TLP **16** (5–6): 670–687, 2016.
doi:10.1017/S1471068416000466

*Deriving conclusions from non-monotonic cause-effect relations**

JORGE FANDINNO

Department of Computer Science
University of Corunna, Corunna, Spain
(*e-mail:* jorge.fandino@udc.es)

submitted 6 May 2016; revised 8 July 2016; accepted 22 August 2016

Abstract

We present an extension of Logic Programming (under stable models semantics) that, not only allows concluding whether a true atom is a cause of another atom, but also *deriving new conclusions* from these causal-effect relations. This is expressive enough to capture informal rules like "if some agent's actions $\mathscr{A}$ have been *necessary* to cause an event E then conclude atom *caused*$(\mathscr{A}, E)$," something that, to the best of our knowledge, had not been formalised in the literature. To this aim, we start from a first attempt that proposed extending the syntax of logic programs with so-called *causal literals*. These causal literals are expressions that can be used in rule bodies and allow inspecting the derivation of some atom A in the program with respect to some query function ψ. Depending on how these query functions are defined, we can model different types of causal relations such as sufficient, necessary or contributory causes, for instance. The initial approach was specifically focused on monotonic query functions. This was enough to cover sufficient cause-effect relations but, unfortunately, necessary and contributory are essentially *non-monotonic*. In this work, we define a semantics for non-monotonic causal literals showing that, not only extends the stable model semantics for normal logic programs, but also preserves many of its usual desirable properties for the extended syntax. Using this new semantics, we provide precise definitions of *necessary* and *contributory* causal relations and briefly explain their behaviour on a pair of typical examples from the Knowledge Representation literature.

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 of reading for default literal *not A* is "there is no way to derive *A*." Although this idea seems quite intuitive and, in fact, several approaches have studied how to syntactically build these derivations or *justifications* (Specht 1993; Pemmasani *et al.* 2004; Pontelli *et al.* 2009; Denecker *et al.* 2015; Schulz and

* This research was partially supported by Spanish Project TIN2013-42149-P.

Toni 2016), it actually resorts to a concept, the *ways to derive A*, outside the scope of the standard LP semantics.

Such information on justifications for atoms can be of great interest for Knowledge Representation (KR), and especially, for dealing with problems related to causality. For instance, in the area of legal reasoning where determining a legal responsibility usually involves finding out which agent or agents have eventually caused a given result, regardless the chain of effects involved in the process. In this sense, 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 and reason about them. As an example, take the assertion:

"If somebody causes an accident, (s)he would receive a fine" (1)

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. Focusing on representing (1) in an elaboration tolerant manner (McCarthy 1998), we should be able to write a single rule whose body only refers to the *agent* involved and the *accident*. For instance, consider the following program

$$accident \leftarrow oil \tag{2}$$

$$oil \leftarrow suzy \tag{3}$$

$$suzy \tag{4}$$

representing that *accident* is an indirect effect of Suzy's actions. We may then represent (1) by the following rule

$$fine(suzy) \leftarrow suzy \ \texttt{necessary for} \ accident \tag{5}$$

that states that Suzy would receive a *fine* whenever the fact *suzy* was necessary to cause the atom *accident*.

With this long term goal in mind, (Cabalar *et al.* 2014a) proposed a multi-valued semantics for LP that extends the stable model semantics (Gelfond and Lifschitz 1988) and where justifications are treated as *algebraic* constructions. In this semantics, *causal stable models* assign, to each atom, one of these algebraic expressions that captures the set of all non-redundant logical proofs for that atom. Recently, this semantics was used in (Fandinno 2015b) to extend the syntax of logic programs with a new kind of literal, called *causal literal*, that allow representing rules like

$$fine(suzy) \leftarrow suzy \ \texttt{sufficient for} \ accident \tag{6}$$

and derive, from a program P_1 containing rules (2-4,6), that $fine(suzy)$ holds. However, the major limitation of this semantics is that causal literals must be monotonic and, therefore, rule (5) cannot be represented. It is easy to see that rule (5) is non-monotonic: in a program P_2 containing rules (2-5), the fact *suzy* is necessary for *accident* is satisfied and, thus, $fine(suzy)$ must hold, but in a program P_3

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

obtained by adding a fact *oil* to this last program, *suzy* is not longer necessary and, thus, $fine(suzy)$ should not be a conclusion.

In this paper, we present a semantics for logic programs with causal literals defined in terms of *non-monotonic* query functions. More specifically, we summarise our contributions as follows. In Section 2, we define the syntax of causal literals and a multi-valued semantics for logic programs whose causal values rely on a completely distributive lattice based on causal graphs. Section 3 shows that positive monotonic program has a least model that can be computed by an extension of the direct consequences operator (van Emden and Kowalski 1976). In Section 4, we define semantics for programs with negation and non-monotonic causal literals and show that it is a conservative extension of the standard stable model semantics. Besides, with a running example, we show how causal literals can be used to derive new conclusion from necessary causal relations and, in Section 5, briefly relate this notion with the actual cause literature. In this section, we also formalise the weaker notion of *contributory cause*, also related to the actual cause literature, and show how causal literals may be used to derive new conclusion from them. In Section 6, we show that our semantics satisfy the usual properties of the stable models semantics for the new syntax. Finally, Section 7 concluded the paper. Proofs of formal results from the paper can be found in an extended version (Fandinno 2016).

2 Causal programs

We start by reviewing some definitions from (Cabalar *et al.* 2014a).

Definition 1 (Term)
Given a set of labels Lb, a *term* t is recursively defined as one of the following expressions

$$t \quad ::= \quad l \;\Big|\; \prod S \;\Big|\; \sum S \;\Big|\; t_1 \cdot t_2$$

where $l \in Lb$ is a label, t_1, t_2 are in their turn terms and S is a (possibly empty and possible infinite) set of terms.

When $S = \{t_1, \ldots, t_n\}$ is a finite set, we will write $t_1 * \ldots * t_n$ and $t_1 + \ldots + t_n$ instead of $\prod S$ and $\sum S$, respectively. When $S = \emptyset$, we denote $\prod S$ and $\sum S$ by 1 and 0, respectively. We assume that application '$\cdot$' has higher priority than product '$*$' and, in its turn, product '$*$' has higher priority than addition '$+$'. *Application* '$\cdot$' represents the application of a rule label to a previous justification. For instance, the justification in program P_1 for atom *suzy* is the fact *suzy* itself. If rules (2-3) in program P_1 are labelled in the following way

$$r_1 : \quad accident \leftarrow oil \tag{7}$$

$$r_2 : \quad oil \leftarrow suzy \tag{8}$$

we may represent the justification of *oil* as $suzy \cdot r_2$, in other words, *oil* is true because of the application of rule r_2 to the fact *suzy*. Similarly, we may represent the justification of *accident* as $suzy \cdot r_2 \cdot r_1$. Addition '$+$' is used to capture alternative

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

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

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

Fig. 1. Properties of the '·'operators: t, u, w are terms, l is a label and c, d, e are terms without '+'. Addition and product distributivity are also satisfied over infinite sums and products.

independent causes: each addend is one of those independent causes. For instance, the justification of *oil*, in program P_3, may be represented as $suzy \cdot r_2 + oil$ and the justification of *accident* as $(suzy \cdot r_2 + oil) \cdot r_1$. As we will see below application distributes over addition, so that, the justification of *accident* can also be written as $suzy \cdot r_2 \cdot r_1 + oil \cdot r_1$, which better illustrates the existence of two alternatives. Product '$*$' represents conjunction or joint causation. For instance, in a program P_4 obtained by adding the fact *billy* to P_3 and replacing rule (8) by

$$r_2 : \quad oil \leftarrow suzy,\ billy \tag{9}$$

the justifications of *oil* will be $(suzy * billy) \cdot r_2 + oil$. Similarly, the justification of *accident* will be $(suzy * billy) \cdot r_2 \cdot r_1 + oil \cdot r_1$. Intuitively, terms without addition '+' represent individual causes while terms with addition '+' represent sets of causes. It is worth to mention that these algebraic expressions are in a one-to-one correspondence with non-redundant proofs of an atom (Cabalar *et al.* 2014a) and that they may also be understood as a formalisation of Lewis' concept of causal chain (Lewis 1973) (see Fandinno 2015b).

Definition 2 (Value)
(Causal) values are the equivalence classes of terms under axioms for a completely distributive (complete) lattice with meet '$*$' and join '+' plus the axioms of Figure 1. The set of values is denoted by $\mathbf{V}_{Lb}$. Furthermore, by $\mathbf{C}_{Lb}$ we denote the subset of causal values with some representative term without sums '+'.

All three operations, '$*$', '+' and '·' are associative. Product '$*$' and addition '+' are also commutative, and they satisfy the usual absorption and distributive laws with respect to infinite sums and products of a completely distributive lattice. The lattice order relation is defined as:

$$t \leqslant u \qquad \text{iff} \qquad t * u = t \qquad \text{iff} \qquad t + u = u$$

An immediate consequence of this definition is that product, addition, 1 and 0 respectively are the greatest lower bound, the least upper bound and the top and the bottom element of the $\leqslant$-relation. Term 1 represents a value which holds by default, without an explicit cause, and will be assigned to the empty body. Term 0 represents the absence of cause or the empty set of causes, and will be assigned to false. Furthermore, applying distributivity (and absorption) of product and application

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

over addition, every term can be represented in *(minimal) disjunctive normal form* in which addition is not in the scope of any other operation and every pair of addends are pairwise $\leqslant$-incomparable. In the following, we will assume that every term is in disjunctive normal form.

This semantics was used in (Fandinno 2015b), to define the concept of causal query, here *m-query*: a monotonic function $\phi : \mathbf{C}_{Lb} \longrightarrow \{0,1\}$. Unfortunately, m-queries are not expressive enough to capture necessary causation for two reasons: (*i*) they are monotonic and (*ii*) they cannot capture relations between sets of causes. We introduced here the following definition which removes these two limitations.

Definition 3 (Causal query)
A *causal query* $\psi : \mathbf{C}_{Lb} \times \mathbf{V}_{Lb} \longrightarrow \{0,1\}$ is a function mapping pairs cause-value into 1 (true) and 0 (false) which is anti-monotonic in the second argument, that is, $\psi(G,t) \leqslant \psi(G,u)$ for every $G \in \mathbf{C}_{Lb}$ and $\{t,u\} \subseteq \mathbf{V}_{Lb}$ such that $t \geqslant u$.

Syntax. We define the semantics of logic programs using its grounding. Therefore, for the remainder of this paper, we restrict our attention to ground logic programs. A *signature* is a triple $\langle At, Lb, \Psi \rangle$ where At, Lb and Ψ respectively represent sets of *atoms* (or *propositions*), *labels* and causal queries. We assume the signature of every program contains a causal query $\psi^1 \in \Psi$ s.t. $\psi^1(G,t) \stackrel{\text{def}}{=} 1$ for every $G \in \mathbf{C}_{Lb}$ and value $t \in \mathbf{V}_{Lb}$.

Definition 4 (Causal literal)
A *(causal) literal* is an expression $(\psi :: A)$ where $A \in At$ is an atom and $\psi \in \Psi$ is a causal query.

A causal atom $(\psi^1 :: A)$ is said to be *regular* and, by abuse of notation, we will use atom A as shorthand for regular causal literals of the form $(\psi^1 :: A)$. We will see below the justification for this notation. A *literal* is either a causal literal $(\psi :: A)$ (*positive literal*), or a negated causal literal $not(\psi :: A)$ (*negative literal*) or a double negated causal literal $not\, not(\psi :: A)$ (*consistent literal*) with $A \in At$ an atom and $\psi \in \Psi$ a causal query.

Definition 5 (Causal program)
A *(causal) program* P is a set of rules of the form:

$$r_i : \quad A \leftarrow B_1, \ldots, B_m \tag{10}$$

where $0 \leqslant m$ is a non-negative integer, $r_i \in Lb$ is a label or $r_i = 1$, A (the *head* of the rule) is an atom and each B_i with $1 \leqslant i \leqslant m$ (the *body* of the rule) is a literal or a term.

A rule r is said to be *positive* iff all literals in its body are positive and it is said to be *regular* if all causal literals in its body are regular. When $m = 0$, we say that the rule is a *fact* and omit the body and sometimes the symbol '$\leftarrow$.' Furthermore, for clarity sake, we also assume that, for every atom $A \in At$, there is a homonymous label $A \in Lb$ and that the label of an unlabelled rule is assumed to be its head. In this sense, a fact A in a program actually stands for the labelled rule $(A : A \leftarrow)$. A program P is *positive* or *regular* when all its rules are positive (i.e. it contains no

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

default negation) or regular, respectively. A *standard program* is a regular program in which the label of every rule is '1 :'.

Semantics. A *(causal) interpretation* is a mapping $I : At \longrightarrow \mathbf{V}_{Lb}$ assigning a value to each atom. For interpretations I and J, we write $I \leqslant J$ when $I(A) \leqslant J(A)$ for every atom $A \in At$. Hence, there is a $\leqslant$-bottom interpretation $\mathbf{0}$ (resp. a $\leqslant$-top interpretation $\mathbf{1}$) that stands for the interpretation mapping every atom A to 0 (resp. 1). For an interpretation I and atom $A \in At$, by $\max I(A)$ we denote the set

$$\max I(A) \stackrel{\text{def}}{=} \{\ G \in \mathbf{C}_{Lb} \mid G \leqslant I(A) \text{ and there is no } G' \in \mathbf{C}_{Lb} \text{ s.t. } G < G' \leqslant I(A)\ \}$$

containing the maximal terms without addition (or individual causes) of A w.r.t. I.

Definition 6 (Causal literal valuation)
The *valuation of a causal literal* of the form $(\psi :: A)$ with respect to an interpretation I, in symbols $I(\psi :: A)$, is given by

$$I(\psi :: A) \stackrel{\text{def}}{=} \sum \{\ G \in \max I(A) \mid \psi(G, I(A)) = 1\ \}$$

We say that I satisfies a causal literal $(\psi :: A)$, in symbols $I \models (\psi :: A)$, iff $I(\psi :: A) \neq 0$.

Notice now that $I(\psi^1 :: A) = I(A)$ for any atom A and, thus, writing a standard atom A as a shorthand for causal literal $(\psi^1 :: A)$ does not modify its intended meaning. Causal literals can be used to represent the body of rule (5). For instance, given a set of labels $\mathscr{A} \subseteq Lb$ representing the actions of some agent $\mathscr{A}$, we may define the query function

$$\psi^{\text{nec}}_{\mathscr{A}}(G, t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if} \quad t \leqslant \sum \mathscr{A} \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

and represent the body of rule (5) by a causal literal of the form $(\psi^{\text{nec}}_{Suzy} :: accident)$ where $Suzy$ is the set of labels $\{suzy\}$. In the sake of clarity, we will usually write $(\mathscr{A}$ `necessary for` $A)$ in rule bodies instead $(\psi^{\text{nec}}_{\mathscr{A}} :: A)$.

If we consider an interpretation I which assigns to the atom *accident* its justification in program P_2, that is, $I(accident) = suzy{\cdot}r_2{\cdot}r_1$, then any term without addition $G \in \mathbf{C}_{Lb}$, satisfies

$$\begin{aligned} \psi^{\text{nec}}_{Suzy}(G, I)(accident) = 1 \quad \text{iff} \quad & suzy{\cdot}r_2{\cdot}r_1 \leqslant \sum\{suzy\} \\ \text{iff} \quad & suzy{\cdot}r_2{\cdot}r_1 \leqslant suzy \\ \text{iff} \quad & suzy{\cdot}r_2{\cdot}r_1 + suzy = suzy \end{aligned}$$

which holds applying identity and associativity of application and absorption w.r.t. addition

$$suzy{\cdot}r_2{\cdot}r_1 + suzy = 1 \cdot suzy \cdot (r_2{\cdot}r_1) + suzy = suzy$$

Similarly, in program P_3, $\psi^{\text{nec}}_{Suzy}(G, I'(accident)) = 1$ iff $suzy{\cdot}r_2{\cdot}r_1 + oil \leqslant suzy$ which does not hold. In other words, Suzy's actions has been necessary in program P_2 but not in program P_3.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

The valuation of a causal term t is the class of equivalence of t. The valuation of non-positive literals is defined as follows

$$I(not(\psi :: A)) \quad \stackrel{\text{def}}{=} \quad \begin{cases} 1 & \text{iff } I(\psi :: A) = 0 \\ 0 & \text{otherwise} \end{cases}$$

$$I(not\, not(\psi :: A)) \quad \stackrel{\text{def}}{=} \quad \begin{cases} 1 & \text{iff } I(\psi :: A) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, for any literal or term L, we write $I \models L$ iff $I(L) \neq 0$.

Definition 7 (Causal model)
Given a rule r of the form (10), we say that an interpretation I *satisfies* r, in symbols $I \models r$, if and only if the following condition holds:

$$\big(I(B_1) * \ldots * I(B_m)\big) \cdot r_i \quad \leqslant \quad I(A) \tag{12}$$

An interpretation I is a *causal model* of P, in symbols $I \models P$, iff I satisfies all rules in P.

Let P_5 be the program containing rules (7) and (8) plus the labelled fact ($suzy$: $suzy \leftarrow$) and P_6 be the program containing rules (7) and (9) plus the labelled facts ($suzy$: $suzy \leftarrow$) and ($billy$: $billy \leftarrow$). Then, it can be checked that these programs respectively have unique $\leqslant$-minimal models I_5 and I_6 which satisfy

$$I_5(accident) = suzy \cdot r_2 \cdot r_1 \qquad I_6(accident) = (suzy * billy) \cdot r_2 \cdot r_1 + oil$$

Let now P_7 and P_8 be the labelled programs respectively obtained by adding the following rule

$$r_3 : \quad fine(suzy) \leftarrow suzy \ \texttt{necessary for} \ accident \tag{13}$$

(resulting of labelling rule (5) with r_3) to programs P_5 and P_6. Then it can be checked that these programs also have unique $\leqslant$-minimal models I_7 and I_8 which respectively agree with I_5 and I_6 in all atoms but in $fine(suzy)$ and, as we have seen above,

$$I_7(\psi^{\text{nec}}_{Suzy} :: accident) = I_7(accident) = suzy \cdot r_2 \cdot r_1 \qquad I_8(\psi^{\text{nec}}_{Suzy} :: accident) = 0$$

Furthermore, by definition, it holds that $I_j(fine(suzy)) = I_j(\psi^{\text{nec}}_{Suzy} :: accident) \cdot r_3$ for $j \in \{7, 8\}$ which implies that

$$I_7(fine(suzy))) = suzy \cdot r_2 \cdot r_3$$
$$I_8(fine(suzy))) = 0 \cdot r_3 = 0$$

That is, Suzy would receive a fine for causing the accident, $I_7 \models fine(suzy)$, w.r.t P_7, but not w.r.t. program P_8 because $I_8 \not\models fine(suzy)$.

It is worth to note that positive programs may contain non-monotonic causal literals that, somehow, play the role of negation and, hence, they may have several $\leqslant$-minimal causal models. Consider, for instance, the following positive program P_9

$$r_1 : \quad p \qquad\qquad r_2 : \quad q \leftarrow \mathscr{A}_1 \ \texttt{necessary for} \ p$$

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

where $\mathscr{A}_1 \stackrel{\text{def}}{=} \{r_1\}$. Program P_9 has two $\leqslant$-minimal causal models. The first one which satisfies $I_9(p) = r_1$ and $I_9(q) = r_1 \cdot r_2$; and a second unintended one which satisfies $I'_9(p) = r_1 + r_2$ and $I'_9(q) = 0$. In the following section, we introduce the notion of *monotonic programs* which have a least model and a well-behaved direct consequences operator (when they are positive). In Section 4, we will see that, in fact, only I_9 is a causal stable model of program P_9.

3 Positive monotonic Programs

A causal query ψ is said to be *monotonic* iff $\psi(G,u) \leqslant \psi(G',w)$ for any values $\{G, G'\} \subseteq \mathbf{C}_{Lb}$ and $\{u, w\} \subseteq \mathbf{V}_{Lb}$ such that $G \leqslant G'$. A causal literal $(\psi :: A)$ is *monotonic* if ψ is monotonic. A program P is *monotonic* iff all causal literals occurring in P are monotonic. We show next that every monotonic program can be reduced to the syntax and semantics of (Fandinno 2015b). For space reasons, we omit here the details of (Fandinno 2015b), which can be found in the extended version (Fandinno 2016).

Definition 8
Given a query ψ (resp. m-query ϕ), its *corresponding m-query (resp. query)* is given by $\phi_\psi(G) \stackrel{\text{def}}{=} \psi(G,1)$ (resp. $\psi_\phi(G,t) \stackrel{\text{def}}{=} \phi(G)$). Similarly, for any program P (resp. m-program Q) its *corresponding m-program Q (resp. program P)* is obtained by replacing every query ψ in P (resp. m-query ϕ in Q) by its corresponding m-query ϕ_ψ (resp.query ψ_ϕ).

Theoram 1
If P is the corresponding program of some positive m-program Q with the syntax of Definition 5 or Q is the corresponding m-program of some positive monotonic program P, then an interpretation I is a model of P iff I is a model of Q.

An immediate consequence of Theorem 1, plus Theorem 3.8 in (Fandinno 2015b), is that positive monotonic programs have a least model that can be computed by iteration of the following extension of the direct consequences operator of van Emden and Kowalski (1976).

Definition 9 (Direct consequences)
Given a causal program P, the operator of *direct consequences* is a function T_P from interpretations to interpretations such that

$$T_P(I)(A) \stackrel{\text{def}}{=} \sum \{ \; \big(I(B_1) * \ldots * I(B_m)\big) \cdot r_1 \;\mid\; (r_i : A \leftarrow B_1, \ldots, B_m) \in P \; \}$$

for any interpretation I and any atom $A \in At$. The iterative procedure is defined as usual

$$T_P^{\uparrow\alpha}(\mathbf{0}) \stackrel{\text{def}}{=} T_P(T_P^{\uparrow\alpha-1}(\mathbf{0})) \qquad \text{if } \alpha \text{ is a successor ordinal}$$

$$T_P^{\uparrow\alpha}(\mathbf{0}) \stackrel{\text{def}}{=} \sum_{\beta<\alpha} T_P^{\uparrow\beta}(\mathbf{0}) \qquad \text{if } \alpha \text{ is a limit ordinal}$$

As usual 0 and ω respectively denote the first limit ordinal and the first limit ordinal that is greater than all integers. Thus, $T_P^{\uparrow 0}(\mathbf{0}) = \mathbf{0}$.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

Corollary 1
Any (possibly infinite) positive monotonic program P has a least causal model I which (i) coincides with the least fixpoint $\mathrm{lfp}(T_P)$ of the direct consequences operator T_P and (ii) can be iteratively computed from the bottom interpretation $I = \mathrm{lfp}(T_P) = T_P^{\uparrow\omega}(\mathbf{0})$.

Corollary 1 guarantees that the least fixpoint of T_P is well-behaved and corresponds to the least model of the program P. In fact, we can check now that the least model I_6 of program P_6 satisfies $I_6(accident) = (suzy * billy)\cdot r_2\cdot r_1 + oil\cdot r_1$. First note, that program P_6 contains facts *suzy*, *billy* and *oil* whose label is the same as the name atom and, thus, $T_{P_6}^{\uparrow 1}(\mathbf{0})(A) = A$ for each atom $A \in \{suzy,\ billy,\ oil\}$. Then, since $T_{P_6}^{\uparrow 1}(\mathbf{0})(suzy) = suzy$, $T_{P_6}^{\uparrow 1}(\mathbf{0})(billy) = billy$ and rule (8) and fact *oil* belong to program P_6, it follows that $T_{P_6}^{\uparrow 2}(\mathbf{0})(oil) = (suzy * billy)\cdot r_2 + oil$. Similarly, we can check that

$$T_{P_6}^{\uparrow 3}(\mathbf{0})(accident) \quad = \quad ((suzy * billy)\cdot r_2 + oil)\cdot r_1 \quad = \quad (suzy * billy)\cdot r_2\cdot r_1 + oil\cdot r_1$$

and, thus, $I_6 = T_{P_6}^{\uparrow 3}(\mathbf{0})$ is the least fixpoint of T_{P_6}. Checking that $T_{P_5}^{\uparrow 3}(\mathbf{0}) = I_5$, that $T_{P_7}^{\uparrow 4}(\mathbf{0}) = I_7$ and that $T_{P_8}^{\uparrow 4}(\mathbf{0}) = I_8$ are the least fixpoint and the least models respectively of programs P_5, P_7 and P_8 is analogous.

It is easy to see that every true atom, according to the standard least model semantics, has a non-zero causal value associated in the causal least model of the program, that is, some associated cause. An interpretation I is *two-valued* when it maps each atom into the set $\{0, 1\}$. By I^{cl}, we denote the two-valued (or "classic") interpretation corresponding to some interpretation I s.t.

$$I^{cl}(A) \quad \stackrel{\text{def}}{=} \quad \begin{cases} 1 & \text{iff } I(A) > 0 \\ 0 & \text{iff } I(A) = 0 \end{cases}$$

Corollary 2
Let P be a regular, positive monotonic program and Q its standard unlabelled version obtained by removing all labels from the rules in P. Let I and J be the least models of P and Q, respectively. Then, $I^{cl} = J$.

4 Non-monotonic causal queries and negation

We introduce now the semantics for programs with non-monotonic causal queries and negation by extending the concept of reduct (Gelfond and Lifschitz 1988) to causal queries.

Definition 10 (Reduct)
For any term t, by ψ^t we denote a query such that

$$\psi^t(G, u) \quad \stackrel{\text{def}}{=} \quad \begin{cases} 1 & \text{iff exists some } G' \leqslant G \text{ s.t. } G' \in \max t \text{ and } \psi(G', t) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The *reduct* of a causal literal $(\psi :: A)$ w.r.t some interpretation I is itself if ψ is monotonic and $(\psi^{I(A)} :: A)$ if ψ is non-monotonic. The reduct of a program P w.r.t.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

an interpretation I, in symbols P^I, is the result of (i) removing all rules whose body contains a non satisfied negative or consistent literal, (ii) removing all the negative and consistent literals for the remaining rules and (iii) replacing the remaining causal literals $(\psi :: A)$ by their reducts $(\psi :: A)^I$.

It is easy to see that the reduct P^I of any program P is a positive monotonic program and, therefore, it has a least causal model.

Definition 11 (Causal stable model)
We say that an interpretation I is a *causal stable model* of a program P iff I is the least model of the positive program P^I.

We can check now that interpretation I_9 is, in fact, the unique causal stable model of program P_9. Let $Q = P_9^{I_9}$ be the reduct of program P_9 w.r.t. I_9 consisting in the following rules

$$r_1 : \quad p \qquad\qquad r_2 : \quad q \leftarrow (\psi :: p)$$

where $\psi(G,t) = 1$ iff there exists some $G' \leqslant G$ s.t. $G' \in \max I_9(p) = r_1$ and $\psi^{\text{nec}}_{\mathscr{A}_1}(G', I_9(p))$ iff $r_1 \leqslant G$ and $r_1 \leqslant \sum \mathscr{A}_1 = r_1$ iff $r_1 \leqslant G$. First note that $T_Q^{\uparrow\alpha}(\mathbf{0})(p) = r_1 = I_9(p)$ for any ordinal $\alpha \geqslant 1$ because r_1 is the only rule with the atom p in the head. Then, note that $T_Q^{\uparrow\alpha}(\mathbf{0})(\psi :: p) = T_Q^{\uparrow\alpha}(\mathbf{0})(p)$ because $r_1 \leqslant G$ for every $G \in \max T_Q^{\uparrow\alpha}(\mathbf{0})(p) = r_1$ (there is only one such $G = r_1$) and, thus,

$$T_Q^{\uparrow\beta}(\mathbf{0})(q) \;=\; T_Q^{\uparrow\alpha}(\mathbf{0})(\psi :: p) \cdot r_2 \;=\; T_Q^{\uparrow\alpha}(\mathbf{0})(p) \cdot r_2 \;=\; r_1 \cdot r_2 \;=\; I_9(q)$$

for any ordinal $\beta \geqslant 2$. Hence, I_9 is a causal stable model of P_9. On the other hand, we can check that I_9' is not a causal stable model of P_9. Let $Q' = P_9^{I_9'}$ be the reduct of program P_9 w.r.t. I_9' consisting in the same rules than program Q, but replacing ψ by ψ' where $\psi'(G,t) = 1$ iff there exists some $G' \leqslant G$ s.t. $G' \in \max I_9'(p) = r_1 + r_2$ and $\psi^{\text{nec}}_{\mathscr{A}_1}(G', I_9'(p))$. As above, $T_{Q'}^{\uparrow\alpha}(\mathbf{0})(p) = r_1 \neq I_9'(p) = r_1 + r_2$ for any ordinal $\alpha \geqslant 1$ and, therefore, I_9 is not a causal stable model of program P_9.

It is worth to mention that, as happened with positive programs, we can establish a correspondence between the causal stable models of regular programs and the standard stable models of their standard version.

Definition 12 (Two-valued equivalence)
Two programs P and Q are said to be *two-valued equivalent* iff for every causal stable model I of P there is a unique causal stable model J of Q such that $I^{cl} = J^{cl}$, and vice-versa.

Theoram 2
Let P be a regular program and Q be its corresponding standard program obtained by removing all labels in P. Then P and Q are two-valued equivalent.

Theorem 2 asserts that, labelling a standard program does not change which atoms are true or false in its stable models, in other words, the causal stable semantics presented here is a conservative extension of the standard stable model semantic.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

5 Contributory cause and its relation with actual causation

Until now, we have considered that an agent is a cause of an event when its actions have been necessary to cause that event. This understanding is similar to the definition of the *modified Halpern-Pearl definition of causality* given by Halpern (2015). However, in some scenarios it makes sense to consider a weaker definition in which those agents whose actions have *contributed* to that event are also considered causes, even if their actions have not been necessary (Pearl 2000). Consider, for instance, the following example from (Hopkins and Pearl 2003).

Example 1
For a firing squad consisting of shooters Billy and Suzy, it is John's job to load Suzy's gun. Billy loads and fires his own gun. On a given day, John loads Suzy's gun. When the time comes, Suzy and Billy shoot the prisoner. The agents who caused the prisoner death would be punished with imprisonment.

In this example, although the actions of any of the agents are not necessary for the prisoner's death, commonsense tells that all three should be considered responsible of it. If we represent Example 1 by the following program P_{10}

$$\begin{array}{lll} r_1 : dead & \leftarrow shoot(suzy), loaded & shoot(suzy) \\ r_2 : dead & \leftarrow shoot(billy) & shoot(billy) \\ r_3 : loaded & \leftarrow load(john) & load(john) \\ r_{\mathscr{A}} : long\_prison(\mathscr{A}) & \leftarrow \mathscr{A} \texttt{ necessary for } dead & \end{array}$$

for $\mathscr{A} \in \{suzy,\ billy,\ john\}$, it can be shown that its unique causal stable model I_{10} satisfies

$$I_{10}(dead) \quad = \quad \big(load(john){\cdot}r_3 * shoot(suzy)\big) \cdot r_1 \quad + \quad shoot(billy){\cdot}r_2$$

Recall that, we assume that every fact has a label with the same name. According to I_{10}, the actions of the three agents appear in the causes of the atom *dead*, but there is no agent whose actions occur in all causes. Then, the causal literal ($\mathscr{A}$ `necessary for` *dead*) is not satisfied for any agent $\mathscr{A}$ and, therefore, for every agent $\mathscr{A} \in \{suzy,\ billy,\ john\}$, it holds that $I_{10}(long\_prison(\mathscr{A})) = 0$. That is, no agent is punished with imprisonment for the prisoner's death. On the other hand, if P_{11} is a program obtained by replacing rules $r_{\mathscr{A}}$ by rules

$$c_{\mathscr{A}} : short\_prison(\mathscr{A}, dead) \leftarrow \mathscr{A} \texttt{ contributed to } dead$$

in program P_{10}, we may expect that $short\_prison(\mathscr{A})$ holds, in its unique causal stable model I_{11}, for any $\mathscr{A} \in \{suzy,\ billy,\ john\}$. We formalise this by defining the following query

$$\psi^{\text{cont}}_{\mathscr{A}}(G,t) \quad \stackrel{\text{def}}{=} \quad \begin{cases} 1 & \text{if } G \leqslant \sum \mathscr{A} \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

In the sake of clarity, we will write ($\mathscr{A}$ `contributed to` *dead*) instead of ($\psi^{\text{cont}}_{\mathscr{A}}$ :: *dead*). It can be checked that $\big(load(john){\cdot}r_3 * shoot(suzy)\big) \cdot r_1 \leqslant load(john)$ and,

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

therefore,

$$I_{11}(john \texttt{ contributed to } dead) = \big(load(john)\cdot r_3 * shoot(suzy)\big)\cdot r_1$$

Consequently, $I_{11}(short\_prison(john)) = \big(load(john)\cdot r_3 * shoot(suzy)\big)\cdot r_1 \cdot c_{john}$. Similarly, it can be shown that

$$I_{11}(short\_prison(suzy)) = \big(load(john)\cdot r_3 * shoot(suzy)\big)\cdot r_1\cdot c_{suzy}$$
$$I_{11}(short\_prison(billy)) = shoot(billy)\cdot r_2 \cdot c_{billy}$$

It is worth to note that contributory causes are non-monotonic when defaults are taken into account. Consider now the following variation of Example 1.

Example 2
Now Suzy also loads her gun as Billy does. However, Suzy's gun was broken and John repaired it.

As in Example 1, John's repairing action is necessary in order for Suzy to be able to fire her gun. However, in this case, it seems too severe to consider that John has contributed to the prisoner's death. This consideration has been widely attributed to the fact that we consider that, by default, things are not broken and that causes must be events that deviate from the norm (Maudlin 2004; Hall 2007; Halpern 2008; Hitchcock and Knobe 2009). If we represent this variation by a program P_{12} containing the following rules[1]

$$\begin{array}{llll} r_1 & : dead & \leftarrow shoot(suzy), un\_broken & shoot(suzy) \\ r_2 & : dead & \leftarrow shoot(billy) & shoot(billy) \\ r_3 & : un\_broken & \leftarrow repair(john) & repair(john) \\ c_{\mathscr{A}} & : short\_prison(\mathscr{A}) & \leftarrow \mathscr{A} \texttt{ contributed to } dead & \end{array}$$

for $\mathscr{A} \in \{suzy,\ billy,\ john\}$, then it is easy to see that

$$I_{12}(dead) = \big(repair(john)\cdot r_3 * shoot(suzy)\big)\cdot r_1 \ + \ shoot(billy)\cdot r_2$$

where I_{12} is the least model of program P_{12} and, thus, $responsible(john, dead)$ will be a conclusion of it. Just note that program P_{12} is the result of replacing atoms *loaded* and $load(john)$ in program P_{11} by *un_broken* and $repair(john)$, respectively. Note also that nothing in program P_{12} reflects the fact that by default guns are *un_broken*. We state that guns are *un_broken* by default adding the following rule

$$1 : un\_broken \leftarrow not\ broken \tag{15}$$

If P_{13} is the result of adding rule (15) to program P_{12} and I_{13} is the least model of P_{13}, then

$$I_{13}(un\_broken) = I_{12}(un\_broken) + 1 = 1$$

[1] We have chosen this representation in order to illustrate the non-monotonicity of contributory cause. However, solving the Frame and Qualification Problems (McCarthy and Hayes 1969; McCarthy 1987) would require the introduction of time and the inertia laws, plus the replacement of rule r_1 by the pair of rules (r_1 : $dead \leftarrow shoot(suzy)$, $not\ ab$) and ($ab \leftarrow broken$). For a detailed discussion of how causality and the inertia laws can combined we refer to (Fandinno 2015a).

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

and, consequently,

$$\begin{aligned} I_{12}(dead) &= \big(1{\cdot}r_3 * shoot(suzy)\big) \cdot r_1 \;+\; shoot(billy){\cdot}r_2 \\ &= \big(r_3 * shoot(suzy)\big) \cdot r_1 \;+\; shoot(billy){\cdot}r_2 \end{aligned}$$

which shows that John is not considered to have contributed to the prisoner's death. Hence, $short\_prison(john)$ is not a conclusion of program P_{13}. It is worth to mention that besides the two syntactic differences between causal queries and m-queries already mentioned, there is a, perhaps, less noticeable difference in the evaluation of causal literals. Note that,

$$\big(repair(john){\cdot}r_3 * shoot(suzy)\big) \cdot r_1 \;\leqslant\; \big(r_3 * shoot(suzy)\big) \cdot r_1$$

and, thus, if we replaced $G \in \max I(A)$ by $G \leqslant I(A)$ in Definition 6 (as done in Fandinno 2015b), it would follow that atom $short\_prison(john)$ would be an unintended conclusion of program P_{13}. It is also worth to mention that, besides (Pearl 2000) approach, the notion of contributory cause is also behind the definitions of actual cause given in (Halpern and Pearl 2005; Hall 2007).

6 Properties of causal logic programs

Theorem 2 established a correspondence for regular programs, but they say nothing about programs with causal queries. For instance, positive program with non-monotonic causal literals may have more than one causal stable model. Consider the following positive program P_{14}

$$\begin{array}{llll} r_1: & p & r_2: & q \leftarrow \mathscr{A}_1 \texttt{ necessary for } p \\ r_3: & q & r_4: & p \leftarrow \mathscr{A}_2 \texttt{ necessary for } q \end{array}$$

obtained by adding rules r_3 and r_4 to program P_9 and where $\mathscr{A}_2 \stackrel{\text{def}}{=} \{r_3\}$. Program P_{14} has two causal stable causal models. The first that satisfies $I_{14}(p) = r_1 + r_3{\cdot}r_4$ and $I_{14}(q) = r_3$. The second $I'_{14}(p) = r_1$ and $I'_{14}(q) = r_3 + r_1{\cdot}r_2$. Let now $Q = P_{14}^{I_{14}}$ be the reduct of program P_{14} w.r.t. I_{14}, which consists in the following rules

$$\begin{array}{llll} r_1: & p & r_2: & q \leftarrow (\psi_1 :: p) \\ r_3: & q & r_4: & p \leftarrow (\psi_2 :: q) \end{array}$$

where $\psi_1(G,t) = 1$ iff there exists some $G' \leqslant G$ such that $G' \in \max I_{14}(p) = r_1 + r_3{\cdot}r_4$ and $\psi^{\text{nec}}_{\mathscr{A}_1}(G', I_{14}(p))$ and $\psi_2(G,t) = 1$ iff there exists $G' \leqslant G$ such that $G' \in \max I_{14}(q) = r_3$ and $\psi^{\text{nec}}_{\mathscr{A}_2}(G', I_{14}(p))$. First, note that $\psi^{\text{nec}}_{\mathscr{A}_1}(G', I_{14}(p))$ iff $I_{14}(p) = r_1 + r_3{\cdot}r_4 \leqslant \sum \mathscr{A}_1 = r_1$ which does not hold. Thus, $\psi_1(G,t) = 0$ for every $G \in \mathbf{C}_{Lb}$ and $t \in \mathbf{V}_{Lb}$. Then, it is clear that the body of rule r_2 is never satisfied and, therefore, $T_Q^{\uparrow\alpha}(\mathbf{0})(q) = r_3$ for any ordinal $\alpha \geqslant 1$. It can also be checked that $\psi_2(r_3, T_Q^{\uparrow\alpha}(\mathbf{0})(q)) = 1$ because there exists $G' = r_3$ such that $G' \in \max I'_{14}(q) = r_3$ and $\psi^{\text{nec}}_{\mathscr{A}_2}(G', I_{14}(q)) = \psi^{\text{nec}}_{\mathscr{A}_2}(r_3, r_3) = 1$ since $r_3 \leqslant \sum \mathscr{A}_2 = r_3$. Hence, since $r_3 \in \max T_Q^{\uparrow\alpha}(\mathbf{0})(q)$ and $\psi_2(r_3, T_Q^{\uparrow\alpha}(\mathbf{0})(q)) = 1$, it follows that $T_Q^{\uparrow\alpha}(\mathbf{0})(\psi_2 :: q) = r_3$ and $T_Q^{\uparrow\beta}(\mathbf{0})(p) = r_1 + T_Q^{\uparrow\alpha}(\mathbf{0})(q){\cdot}r_4 = r_1 + r_3{\cdot}r_4 = I_9(p)$ for any ordinal $\beta \geqslant 2$. Hence, I_{14} is the least model of $P_{14}^{I_{14}}$ and a causal stable model of program P_{14}. Showing that I'_{14} is also a causal stable model of P_{14} is symmetric.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

In the following we revise some desired general properties for a LP semantics. First, causal stable models should also be supported models. Note that the concept of supported model bellow is analogous to the usual concept used in standard LP, but it is stronger in the sense that, not only requires that true atoms are supported, but also all their causes must be supported by a rule and a cause of its body.

Definition 13
An interpretation I is a *(causally) supported model* of a program P iff I is a model of P and for every true atom A and cause $G \in \mathbf{C}_{Lb}$ such that $G \leqslant I(A)$ there is a rule r in P of the form of (10) such that $G \leqslant (I(B_1) * \ldots * I(B_m)) \cdot r_i$.

Proposition 1
Any causal stable model I of a program P is also a supported model of P.

Furthermore, as happen with programs with nested negation under the standard stable models semantics (where stable models may not be minimal models of the program), causal stable models may not be minimal models either. In fact, this may happen even when the nested negation is replaced by a non-monotonic causal literal. Consider, for instance, the following program P_{15}

$$r_1 : p \qquad\qquad r_2 : p \leftarrow not\ (\mathscr{A}_1 \ \texttt{necessary for}\ p)$$

where $\mathscr{A}_1 \stackrel{\text{def}}{=} \{r_1\}$. Program P_{15} has two causal models. One which satisfies $I_{15}(p) = r_1$. The other which satisfies $I'_{15}(p) = r_1 + r_2$. We define now the notion of *normal program* whose causal stable models are also $\leqslant$-minimal models. A program P is *normal* iff no body rule in P contains a consistent literal (double negated literal) nor a negated non-monotonic causal literal. In other words, a program is normal iff it does not contain nested negation nor non-monotonic causal literals in the scope of negation.

Proposition 2
Any causal stable model I of normal program P is also a $\leqslant$-minimal model.

Splitting programs. The intuitive meaning of the causal rule (13) in programs P_7 and P_8 is to cause the atom $fine(suzy)$ whenever the causal query expressed by its body is true with respect to programs P_5 and P_6, respectively. This intuitive understanding can be formalised as a splitting theorem in (Lifschitz and Turner 1994).

Theoram 3 (Splitting)
Let $\langle P_b, P_t \rangle$ a partition of a program P such that no atom occurring in the head of a rule in P_t occurs in P_b. An interpretation I is a causal stable model of P iff there is some causal stable model J of P_b such that I is a causal stable model of $(J \cup P_t)$.

In our running example, the bottom part are $P_{7,b} = P_5$ and $P_{8,b} = P_6$ while their top part $P_{7,t} = P_{8,t}$ is the program containing the rule (13). This result can be generalised to infinite splitting sequences as follows.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

Definition 14
A *splitting sequence* of a program P is a family $(P_\alpha)_{\alpha<\mu}$ of pairwise disjoint sets such that $P = \bigcup_{\alpha<\mu} P_\alpha$ and no atom occurring in the head of a rule in some P_α occurs in the body of a rule in $\bigcup_{\beta<\alpha} P_\beta$. A *solution* of a splitting $(P_\alpha)_{\alpha<\mu}$ is a family $(I_\alpha)_{\alpha<\mu}$ such that align=Center, leftmargin=10pt, itemindent=0.5pt

1. I_0 is a stable model of P_0,
2. I_α is a stable model of $(J_\alpha \cup P_\alpha)$ for any ordinal $0 < \alpha < \mu$ where $J_\alpha = \sum_{\beta<\alpha} I_\beta$.

A splitting sequence is said to be *strict in* α if, in addition, no atom occurring in the head of a rule in P_α occurs (in the head of a rule) in $\bigcup_{\beta<\alpha} P_\beta$ and it is said to be *strict* if it is strict in α for every $\alpha < \mu$.

Theoram 4 (Splitting sequences)
Let $(P_\alpha)_{\alpha<\mu}$ a splitting sequence of some program P. An interpretation I is a causal stable model of P iff there is some solution $(I_\alpha)_{\alpha<\mu}$ of $(P_\alpha)_{\alpha<\mu}$ such that $I = \sum_{\alpha<\mu} I_\alpha$. Furthermore, if such solution is strict in α, then $I_\alpha = I_{|S_\alpha}$ where S_α is the set of all atoms not occurring in the head of any rule in $\bigcup_{\alpha<\beta<\mu} P_\beta$ and $I_{|S_\alpha}$ denotes the restriction if I to S_α.

A program P is said to be *stratified* if there is a some ordinal μ and mapping λ from the set of atoms At into the set of ordinals $\{\alpha < \mu\}$ such that, for every rule of the form (10) and atom B occurring in its body, it satisfies $\lambda(A) \geqslant \lambda(B)$ if B does not occur in the scope of negation nor in a non-monotonic causal literal, and $\lambda(A) > \lambda(B)$ if B does occur under the scope of negation or in a non-monotonic causal literal.

Proposition 3
Every stratified causal program P has a unique causal stable model.

7 Conclusions, related work and open issues

The main contribution of this work is the introduction of a semantics for non-monotonic *causal literals* that allow deriving new conclusions by inspecting the causal justifications of atoms in an *elaboration tolerant* manner. In particular, we have used causal literals to define *necessary* and *contributory causal relations* which are intuitively related to some of the most established definitions of actual causation in the literature (Pearl 2000; Halpern and Pearl 2005; Hall 2007; Halpern 2015). Besides, by some running examples we have shown that causal literals allow, not only to derive whether some event is the cause or not of another event, but also to derive new conclusions from this fact. From a technical point of view, we have shown that our semantics is a conservative extension of the stable model semantics and that satisfy the usual desired properties for an LP semantics (casual stable models are supported models, minimal models in case of normal programs and can be iteratively computed by split table programs). It worth to mention that, besides the syntactic approaches to justifications in LP, the more related approach to our semantics is (Damásio *et al.* 2013), for which a formal comparative can be found in (Cabalar and Fandinno 2016a) and that (Pontelli *et al.* 2009) allows a Prolog

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press

system to reason about justifications of an ASP program, but justifications cannot be inspected inside the ASP program.

Regarding complexity, it has been shown in (Cabalar *et al.* 2014b) that there may be an exponential number of causes for a given atom w.r.t. each causal stable model. Despite that, the existence of stable model for programs containing only monotonic queries evaluable in polynomial time is NP-complete (Fandinno 2015b). For programs containing only necessary causal literals we can prove NP-complete (NP-hard holds even for programs containing a single negated regular literal or positive programs containing a single constraint, see (Fandinno 2016)). The complexity for programs including other non-monotonic causal literals (like contributory) is still an open question. A preliminary prototype extending the syntax of logic programs with causal literals capturing sufficient, necessary and contributory causal relation can be tested on-line at `http://kr.irlab.org/cgraphs-solver/nmsolver`.

In a companion paper (Cabalar and Fandinno 2016b), the causal semantics used here has been extended to disjunctive logic programs, which will be useful for representing non-deterministic causal laws. Interesting topics include extending the complexity assessment for contributory causes, studying an extension to arbitrary theories as with Equilibrium Logic (Pearce 2006) for the non-causal case; and formalise the relation between our notions of necessary and contributory cause with the above definitions of the actual causation and, in particular, with (Vennekens 2011) who has studied it in the context of CP-logic. A promising approach seems to translate structural equations into logic programs in a similar way as it has been done to translate them into the causal theories (Giunchiglia *et al.* 2004; Bochman and Lifschitz 2015).

References

Bochman, A. and Lifschitz, V. 2015. Pearl's causality in a logical setting. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, B. Bonet and S. Koenig, Eds. AAAI Press, 1446–1452.

Cabalar, P. and Fandinno, J. 2016a. Enablers and inhibitors in causal justifications of logic programs. *Theory and Practice of Logic Programming (TPLP), (First View)*, 1–26.

Cabalar, P. and Fandinno, J. 2016b. Justifications for programs with disjunctive and causal-choice rules. *Theory and Practice of Logic Programming (TPLP)*. (to appear).

Cabalar, P., Fandinno, J. and Fink, M. 2014a. Causal graph justifications of logic programs. *Theory and Practice of Logic Programming (TPLP) 14,* 4-5, 603–618.

Cabalar, P., Fandinno, J. and Fink, M. 2014b. A complexity assessment for queries involving sufficient and necessary causes. In *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, E. Fermé and J. Leite, Eds. Lecture Notes in Computer Science, vol. 8761. Springer, 297–310.

Damásio, C. V., Analyti, A. and Antoniou, G. 2013. Justifications for logic programming. In *Logic Programming and Nonmonotonic Reasoning, Twelfth International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 530–542.

Denecker, M., Brewka, G. and Strass, H. 2015. A formal theory of justifications. In *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR*

2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, F. Calimeri, G. Ianni, and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 9345. Springer, 250–264.

Fandinno, J. 2015a. A causal semantics for logic programming. Ph.D. thesis, University of Corunna.

Fandinno, J. 2015b. Towards deriving conclusions from cause-effect relations. In *in Proc. of the 8th International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP 2015, Cork, Ireland, August 31, 2015. Extended version under revison for publication in Fundamenta Informaticae.*

Fandinno, J. 2016. Deriving conclusions from non-monotonic cause-effect relations. *CoRR abs/1608.00867.*

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, 1070–1080.

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., and Turner, H. 2004. Nonmonotonic causal theories. *Artificial Intelligence 153,* 1-2, 49–104.

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

Halpern, J. Y. 2008. Defaults and normality in causal structures. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16-19, 2008*, G. Brewka and J. Lang, Eds. AAAI Press, 198–208.

Halpern, J. Y. 2015. A modification of the halpern-pearl definition of causality. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Q. Yang and M. Wooldridge, Eds. AAAI Press, 3022–3033.

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.

Hopkins, M. and Pearl, J. 2003. Clarifying the usage of structural models for commonsense causal reasoning. In *Proceedings of the AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning.* 83–89.

Lewis, D. K. 1973. Causation. *The journal of philosophy 70,* 17, 556–567.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, P. V. Hentenryck, Ed. MIT Press, 23–37.

Maudlin, T. 2004. Causation, counterfactuals, and the third factor. In *Causation and Counterfactuals*, J. Collins, E. J. Hall, and L. A. Paul, Eds. MIT Press.

McCarthy, J. 1987. Epistemological problems of artificial intelligence. *Readings in artificial intelligence*, 459.

McCarthy, J. 1998. Elaboration tolerance. In *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning, Common Sense.* London, UK, 198–217.

McCarthy, J. and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence Journal 4*, 463–512.

Pearce, D. 2006. Equilibrium logic. *Annals of Mathematics and Artificial Intelligence 47,* 1-2, 3–41.

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

Pemmasani, G., Guo, H., Dong, Y., Ramakrishnan, C. R., and Ramakrishnan, I. V. 2004. Online justification for tabled logic programs. In *Functional and Logic Programming,*

Seventh International Symposium, FLOPS 2004, Nara, Japan, April 7-9, 2004, Proceedings, Y. Kameyama and P. J. Stuckey, Eds. Lecture Notes in Computer Science, vol. 2998. Springer, 24–38.

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

Schulz, C. and Toni, F. 2016. Justifying answer sets using argumentation. *Theory and Practice of Logic Programming (TPLP) 16,* 1, 59–110.

Specht, G. 1993. Generating explanation trees even for negations in deductive database systems. In *Proceedings of the Fifth Workshop on Logic Programming Environments (LPE 1993), October 29-30, 1993, In conjunction with ILPS 1993, Vancouver, British Columbia, Canada*, M. Ducassé, B. L. Charlier, Y. Lin, and L. Ü. Yalçinalp, Eds. IRISA, Campus de Beaulieu, France, 8–13.

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

Vennekens, J. 2011. Actual causation in CP-logic. *Theory and Practice of Logic Programming (TPLP) 11,* 4-5, 647–662.

https://doi.org/10.1017/S1471068416000466 Published online by Cambridge University Press