TLP **17** (1): 49–74, 2017. © Cambridge University Press 2016
doi:10.1017/S1471068416000107 First published online 3 May 2016

*Enablers and inhibitors in causal justifications of logic programs**

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

submitted 21 October 2015; revised 26 January 2016; accepted 26 February 2016

Abstract

In this paper, we propose an extension of logic programming where each default literal derived from the well-founded model is associated to a justification represented as an algebraic expression. This expression contains both causal explanations (in the form of proof graphs built with rule labels) and terms under the scope of negation that stand for conditions that enable or disable the application of causal rules. Using some examples, we discuss how these new conditions, we respectively call *enablers* and *inhibitors*, are intimately related to default negation and have an essentially different nature from regular cause-effect relations. The most important result is a formal comparison to the recent algebraic approaches for justifications in logic programming: *Why-not Provenance* and *Causal Graphs*. We show that the current approach extends both Why-not Provenance and Causal Graphs justifications under the well-founded semantics and, as a byproduct, we also establish a formal relation between these two approaches.

KEYWORDS: causal justifications, well-founded semantics, stable models, answer set programming

1 Introduction

The strong connection between non-monotonic reasoning and logic programming (LP) semantics for default negation has made possible that LP tools became nowadays an important paradigm for knowledge representation and problem-solving in Artificial Intelligence. In particular, *Answer Set Programming* (Marek and Truszczyński 1999; Niemelä 1999) has established as a preeminent LP paradigm for practical non-monotonic reasoning with applications in diverse areas of Artificial Intelligence including planning, reasoning about actions, diagnosis, abduction and beyond. The Answer Set Programming paradigm is based on the *stable models semantics* (Gelfond and Lifschitz 1988) and is also closely related to the other mainly accepted interpretation for default negation, *well-founded* semantics (Van

* This is an extended version of a paper presented at the Logic Programming and Non-monotonic Reasoning Conference (LPNMR 2015), invited as a rapid communication in TPLP. The authors acknowledge the assistance of the conference program chairs Giovambattista Ianni and Miroslaw Truszczynski.

Gelder *et al.* 1991). One interesting difference between these two LP semantics and classical models (or even other non-monotonic reasoning approaches) is that true atoms in LP must be founded or justified by a given derivation. These *justifications* are not provided in the semantics itself, but can be syntactically built in some way in terms of the program rules, as studied in several approaches (Denecker and Schreye 1993; Specht 1993; Pemmasani *et al.* 2004; Gebser *et al.* 2008; Pontelli *et al.* 2009; Oetsch *et al.* 2010; Schulz and Toni 2016).

Rather than manipulating justifications as mere syntactic objects, two recent approaches have considered extended multi-valued semantics for LP where justifications are treated as *algebraic* constructions: *Why-not Provenance* (WnP) (Damásio *et al.* 2013) and *Causal Graphs* (CG) (Cabalar *et al.* 2014a). Although these two approaches present formal similarities, they start from different understandings of the idea of justification. On the one hand, WnP answers the query "why literal L might hold" by providing conjunctions of *hypothetical modifications* on the program that would allow deriving L. These modifications include rule labels, expressions like $not(A)$ with A an atom, or negations '$\neg$' of the two previous cases. As an example, a justification for L like $r_1 \wedge not(p) \wedge \neg r_2 \wedge \neg not(q)$ means that the presence of rule r_1 and the absence of atom p would allow deriving L (hypothetically) if both rule r_2 were removed and atom q were added to the program. If we want to explain why L *actually* holds, we have to restrict to justifications without '$\neg$', that is, those without program modifications (which will be the focus of this paper).

On the other hand, CG-justifications start from identifying program rules as *causal laws* so that, for instance, $(p \leftarrow q)$ can be read as "event q *causes* effect p". Under this viewpoint, (positive) rules offer a natural way for capturing the concept of *causal production*, i.e. a continuous chain of events that has helped to cause or produce an effect (Hall 2004; Hall 2007). The explanation of a true atom is made in terms of graphs formed by rule labels that reflect the ordered rule applications required for deriving that atom. These graphs are obtained by algebraic operations exclusively applied on the positive part of the program. Default negation in CG is understood as absence of cause and, consequently, a false atom has *no justification.*

The explanation of an atom A in CG is more detailed than in WnP, since the former contains graphs that correspond to all relevant proofs of A, whereas in WnP we just get conjunctions that do not reflect any particular ordering among rule applications. However, as explained before, CG does not capture the effect of default negation in a given derivation and, sometimes, this information is very valuable, especially if we want to answer questions of the form "why not".

As in the previous paper on CG (Cabalar *et al.* 2014a), our final goal is to achieve an elaboration tolerant representation of causality that allows reasoning about cause–effect relations. Under this perspective, although WnP is more oriented to program debugging, its possibility of dealing with hypothetical reasoning of the form "why not" would be an interesting feature to deal with counterfactuals, since several approaches to causality (see Section 5) are based on this concept. To understand the kind of problems, we are interested in, consider the following example. A drug d in James Bond's drink causes his paralysis p provided that he was not given an antidote a that day. We know that Bond's enemy, Dr. No, poured

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

the drug:

$$p \leftarrow d, \mathit{not}\ a \tag{1}$$

$$d \tag{2}$$

In this case, it is obvious that d causes p, whereas the absence of a just *enables* the application of the rule. Now, suppose we are said that Bond is daily administered an antidote by the MI6, unless it is a holiday h:

$$a \leftarrow \mathit{not}\ h \tag{3}$$

Adding this rule makes a become an *inhibitor* of p, as it prevents d to cause p by rule (1). But suppose now that we are in a holiday, that is, fact h is added to the program (1)–(3). Then, the inhibitor a is *disabled* and d causes p again. However, we do not consider that the holiday h is a (productive) cause for Bond's paralysis p although, indeed, the latter counterfactually depends on the former: "had not been a holiday h, Bond would have not been paralysed". We will say that the fact h, which disables inhibitor a, is an *enabler* of p, as it allows applying rule (1).

In this work, we propose dealing with these concepts of enablers and inhibitors by augmenting CG justifications with a new negation operator '$\sim$' in the CG causal algebra. We show that this new approach, which we call *Extended Causal Justifications* (ECJ), captures WnP justifications under the well-founded semantics, establishing a formal relation between WnP and CG as a byproduct.

The rest of the paper is structured as follows. The next section defines the new approach. Sections 3 and 4 explain the formal relations to CG and WnP through a running example. Section 5 studies several examples of causal scenarios from the literature and finally, Section 6 concludes the paper. The online appendix contains an auxiliary figure depicting some common algebraic properties and the formal proofs of theorems from the previous sections.

2 Extended causal justifications (ECJ)

A *signature* is a pair $\langle At, Lb\rangle$ of sets that respectively represent *atoms* (or *propositions*) and *labels*. Intuitively, each atom in At will be assigned justifications built with rule labels from Lb. In principle, the intersection $At \cap Lb$ does not need to be empty: we may sometimes find it convenient to label a rule using an atom name (normally, the head atom). Justifications will be expressions that combine four different algebraic operators: a product '$*$' representing conjunction or joint causation; a sum '$+$' representing alternative causes; a non-commutative product '$\cdot$' that captures the sequential order that follows from rule applications; and a non-classical negation '$\sim$' which will precede inhibitors (negated labels) and enablers (doubly negated labels).

Definition 1 (*Terms*)
Given a set of labels Lb, a *term*, t is recursively defined as one of the following expressions $t ::= l \mid \prod S \mid \sum S \mid t_1 \cdot t_2 \mid \sim t_1$, where $l \in Lb$, t_1, t_2 are in their turn terms and S is a (possibly empty and possibly infinite) set of terms. A term is *elementary* if it has the form l, $\sim l$ or $\sim\sim l$ with $l \in Lb$ being a label. □

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

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

Idempotency	*Addition distributivity*	*Product distributivity*
$x \cdot x = x$	$t \cdot (u+w) = (t \cdot u) + (t \cdot w)$ $(t+u) \cdot w = (t \cdot w) + (u \cdot w)$	$c \cdot d \cdot e = (c \cdot d) * (d \cdot e)$ 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 '$\cdot$' operator (c, d, e are terms without '$+$' and x is an elementary term). Distributivity is also satisfied over infinite sums and products.

Pseudo-complement	*De Morgan*	*Weak excl. middle*	*appl. negation*
$t * {\sim}t = 0$ ${\sim}{\sim}{\sim}t = {\sim}t$	${\sim}(t+u) = ({\sim}t * {\sim}u)$ ${\sim}(t * u) = ({\sim}t + {\sim}u)$	${\sim}t + {\sim}{\sim}t = 1$	${\sim}(t \cdot u) = {\sim}(t * u)$

Fig. 2. Properties of the '$\sim$' operator.

When $S = \{t_1, \ldots, t_n\}$ is finite, we simply write $\prod S$ as $t_1 * \cdots * t_n$ and $\sum S$ as $t_1 + \cdots + t_n$. Moreover, when $S = \emptyset$, we denote $\prod S$ by 1 and $\sum S$ by 0, as usual, and these will be the identities of the product '$*$' and the addition '$+$', respectively. We assume that '$\cdot$' has higher priority than '$*$' and, in turn, '$*$' has higher priority than '$+$'.

Definition 2 (*Values*)
A *(causal) value* is each equivalence class of terms under axioms for a completely distributive (complete) lattice with meet '$*$' and join '$+$' plus the axioms of Figures 1 and 2. The set of (causal) values is denoted by $\mathbf{V}_{Lb}$. □

Note that $\langle \mathbf{V}_{Lb}, +, *, \sim, 0, 1 \rangle$ is a completely distributive Stone algebra (a pseudo-complemented, completely distributive, complete lattice which satisfies the weak excluded middle axiom) whose meet and join are, as usual, the product '$*$' and the addition '$+$'. Informally speaking, this means that these two operators satisfy the properties of a Boolean algebra but without negation.

Note also that all three operations, '$*$', '$+$' and '$\cdot$' are associative. Product '$*$' and addition '$+$' are also commutative, and they hold the usual absorption and distributive laws with respect to infinite sums and products of a completely distributive lattice.

The axioms for '$\cdot$' in Figure 1 are directly extracted from the CG algebraic structure. For a more detailed explanation on their induced behaviour, see Cabalar *et al.* (2014a). The new contribution in this paper with respect to the CG algebra is the introduction of the '$\sim$' operator whose meaning is captured by the axioms in Figure 2. As we can see, this operator satisfies De Morgan laws and acts as a complement for the product $t * {\sim}t = 0$. However, it diverges from a classical Boolean negation in some aspects. In the general case, the axioms ${\sim}{\sim}t = t$ (double

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

negation) and $t + {\sim}t = 1$ (excluded middle) are not valid. Instead[1], we can replace a triple negation ${\sim}{\sim}{\sim}t$ by ${\sim}t$, and we have a weak version of the excluded middle axiom ${\sim}t + {\sim}{\sim}t = 1$. The negation of an application is defined as the negation of the product ${\sim}(t \cdot u) \stackrel{\text{def}}{=} {\sim}(t * u)$ which, in turn, is equivalent to ${\sim}(u * t)$, since $*$ is commutative. In other words, under negation, the rule application ordering is disregarded. It is not difficult to see that we can apply the axioms of negation to reach an equivalent expression that avoids its application to other operators. We say that a term is in *negation normal form* if no other operator is in the scope of negation '${\sim}$'. Moreover, a negation normal form term is in *disjuntive normal form* (DNF) if: (1) no sum is in the scope of another operator; (2) only elementary terms are in the scope of application and (3) every product is transitively closed, that is, of the form of $a{\cdot}b * b{\cdot}c * a{\cdot}c$. Without loss of generality, we assume from now that all functions defined over causal terms are applied over their DNF form, although, we will usually write them in negation normal form for short.

The lattice order relation is defined as usual in the following way:

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

Consequently, 1 and 0 are respectively the top and bottom elements with respect to relation $\leqslant$.

Definition 3 (*Labelled logic program*)
Given a signature $\langle At, Lb\rangle$, a *(labelled logic) program* P is a set of rules of the form:

$$r_i : \quad H \leftarrow B_1, \ldots, B_m, \; not\; C_1, \ldots, \; not\; C_n \tag{4}$$

where $r_i \in Lb$ is a label or $r_i = 1$, H (the *head* of the rule) is an atom, and B_i's and C_i's (the *body* of the rule) are either atoms or terms. □

When $n = 0$, we say that the rule is positive, furthermore, if in addition $m = 0$ we say that the rule is a *fact* and omit the symbol '$\leftarrow$'. When $r_i \in Lb$, we say that the rule is labelled; otherwise $r_i = 1$ and we omit both r_i and ':'. By these conventions, for instance, an unlabelled fact A is actually an abbreviation of $(1 : A \leftarrow)$. A program P is *positive* when all its rules are positive, i.e. it contains no default negation. It is *uniquely labelled* when each rule has a different label or no label at all. In this paper, we will assume that programs are uniquely labelled. Furthermore, for the sake of clarity, we also assume that, for every atom $A \in At$, there is an homonymous label $A \in Lb$, and that each fact A in the program actually stands for the labelled rule $(A : A \leftarrow)$. For instance, following these conventions, a possible labelled version for the James Bond's program could be program P_1 below:

$$\begin{array}{lll} r_1 : & p \leftarrow d, not\; a & d \\ r_2 : & a \leftarrow not\; h & h \end{array}$$

[1] This behaviour coincides indeed with the properties for default negation obtained in Equilibrium Logic (Pearce 1996) or the equivalent General Theory of Stable Models (Ferraris *et al.* 2007).

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

where facts d and h stand for rules $(d : d \leftarrow)$ and $(h : h \leftarrow)$, respectively.

An *ECJ-interpretation* is a mapping $I : At \longrightarrow \mathbf{V}_{Lb}$ assigning a value to each atom. For interpretations I and J, we say that $I \leqslant J$, when $I(A) \leqslant J(A)$ for each 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 each atom A to 0 (resp. 1). The value assigned to a negative literal *not* A by an interpretation I, denoted as $I(not\ A)$, is defined as $I(not\ A) \stackrel{\text{def}}{=} {\sim} I(A)$, as expected. Similarly, for a term t, $I(t) \stackrel{\text{def}}{=} [t]$ is the equivalence class of t.

Definition 4 (*Model*)
An interpretation I satisfies a rule like (4) iff

$$\big(I(B_1) * \ldots * I(B_m) * I(not\ C_1) * \ldots * I(not\ C_n) \big) \cdot r_i \ \leqslant \ I(H) \tag{5}$$

and I is a (causal) model of P, written $I \models P$, iff I satisfies all rules in P. □

As usual in LP, for positive programs, we may define a direct consequence operator T_P s.t.

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

for any interpretation I and atom $H \in At$. We also define $T_P \uparrow^{\alpha}(\mathbf{0}) \stackrel{\text{def}}{=} T_P(T_P \uparrow^{\alpha-1}(\mathbf{0}))$ for any successor ordinal α and

$$T_P \uparrow^{\alpha}(\mathbf{0}) \ \stackrel{\text{def}}{=} \ \sum_{\beta<\alpha} T_P \uparrow^{\beta}(\mathbf{0})$$

for any limit ordinal α. As usual, ω denotes the smallest infinite limit ordinal. Note that 0 is considered a limit ordinal and, thus, $T_P \uparrow^0(\mathbf{0}) = \sum_{\beta<0} T_P \uparrow^{\beta}(\mathbf{0}) = \mathbf{0}$.

Theorem 1
Let P be a (possibly infinite) positive logic program. Then, (i) the least fixpoint of the T_P operator, denoted by $\mathrm{lfp}(T_P)$, satisfies $\mathrm{lfp}(T_P) = T_P \uparrow^{\omega}(\mathbf{0})$ and it is the least model of P, (ii) furthermore, if P is finite and has n rules, then $\mathrm{lfp}(T_P) = T_P \uparrow^{\omega}(\mathbf{0}) = T_P \uparrow^{n}(\mathbf{0})$. □

Theorem 1 asserts that, as usual, positive programs have a $\leqslant$-least causal model. As we will see later, this least model coincides with the traditional least model (of the program without labels) when one just focuses on the set of true atoms, disregarding the justifications explaining why they are true. For programs with negation, we define the following reduct.

Definition 5 (*Reduct*)
Given a program P and an interpretation I, we denote by P^I the positive program containing a rule of the form

$$r_i : \ H \leftarrow B_1, \ldots, B_m, \ I(not\ C_1), \ldots, \ I(not\ C_n) \tag{6}$$

for each rule of the form (4) in P. □

Program P^I is positive and, from Theorem 1, it has a *least causal model*. By $\Gamma_P(I)$, we denote the least model of program P^I. The operator Γ_P is anti-monotonic and,

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

consequently, Γ^2_P is monotonic (Proposition 4 in the appendix) so that, by Knaster–Tarski's theorem, it has a least fixpoint $\mathbb{L}_P$ and a greatest fixpoint $\mathbb{U}_P \stackrel{\text{def}}{=} \Gamma_P(\mathbb{L}_P)$. These two fixpoints respectively correspond to the justifications for true and for non-false atoms in the (standard) well-founded model (WFM), we denote as W_P.

For instance, in our running example, $\mathbb{L}_{P_1}(d) = \Gamma^2_{P_1} \uparrow^\alpha (\mathbf{0})(d) = d$ for $1 \leqslant \alpha$ points out that atom d is true because of fact d. Similarly, $\mathbb{L}_{P_1}(h) = h$ and $\mathbb{L}_{P_1}(a) = \sim h{\cdot}r_2$ reveals that atom h is true because of fact h, and that atom a is not true because fact h has inhibited rule r_2.

Furthermore,

$$\mathbb{L}_{P_1}(p) \;=\; \Gamma^2_{P_1} \uparrow^\alpha (\mathbf{0})(p) \;=\; (\sim(\sim h{\cdot}r_2) * d){\cdot}r_1 \;=\; (\sim\sim h * d){\cdot}r_1 + (\sim r_2 * d){\cdot}r_1$$

for $2 \leqslant \alpha$. That is, Bond has been paralysed because fact h has enabled drug d to cause the paralysis by means of rule r_1. This corresponds to the justification $(\sim\sim h * d){\cdot}r_1$. Notice how the real cause d is a positive label (not in the scope of negation), whereas the enabler h is in the scope of a double negation $\sim\sim h$. Justification $(\sim r_2 * d){\cdot}r_1$ means that $d{\cdot}r_1$ would have been sufficient to cause p, had not been present r_2. This example is also useful for illustrating the importance of axiom *appl. negation*. By directly evaluating the body of rule r_1, we have seen that $\Gamma^2_{P_1} \uparrow^2 (\mathbf{0})(p) = (\sim(\sim h \cdot r_2) * d) \cdot r_1$. Then, axiom *appl. negation* allows us to break the dependence between $\sim h$ and r_2 into enablers and inhibitors: $\sim(\sim h \cdot r_2) = \sim(\sim h * r_2) = \sim\sim h + \sim r_2$ and, applying distributivity, we obtain one enabled justification, $(\sim\sim h * d){\cdot}r_1$, and one disabled one, $(\sim r_2 * d){\cdot}r_1$.

In our previous example, the least and greatest fixpoint coincided $\mathbb{L}_{P_1} = \mathbb{U}_{P_1} = \Gamma^2_{P_1} \uparrow^2 (\mathbf{0})$. To illustrate the case where this does not hold consider, for instance, the program P_2 formed by the following negative cycle:

$$r_1 : \; a \leftarrow not\ b \qquad\qquad r_2 : \; b \leftarrow not\ a$$

In this case, the least fixpoint of Γ^2_P assigns $\mathbb{L}_{P_2}(a) = \sim r_2{\cdot}r_1$ and $\mathbb{L}_{P_2}(b) = \sim r_1{\cdot}r_2$, while, in its turn, the greatest fixpoint of Γ^2_P corresponds to $\mathbb{U}_{P_2}(a) = r_1$ and $\mathbb{U}_{P_2}(b) = r_2$. If we focus on atom a, we can observe that it is not concluded to be true, since the least fixpoint $\mathbb{L}_P$ has only provided one disabled justification $\sim r_2{\cdot}r_1$ meaning that r_2 is acting as a disabler for a. But, on the other hand, a cannot be false either since the greatest fixpoint provides an enabled justification r_1 for being non-false (remember that $\mathbb{U}_P$ provides justifications for non-false atoms). As a result, we get that a is left undefined because r_2 prevents it to become true while r_1 can still be used to conclude that it is not false.

To capture these intuitions, we provide some definitions. A *query literal (q-literal)* L is either an atom A, its default negation '*not A*' or the expression '*undef A*' meaning that A is undefined.

Definition 6 (*Causal well-founded model*)
Given a program P, its *causal* WFM $\mathbb{W}_P$ is a mapping from q-literals to values s.t.

$\mathbb{W}_P(A) \stackrel{\text{def}}{=} \mathbb{L}_P(A)$ $\mathbb{W}_P(not\ A) \stackrel{\text{def}}{=} \sim\mathbb{U}_P(A)$ $\mathbb{W}_P(undef\ A) \stackrel{\text{def}}{=} \sim\mathbb{W}_P(A) * \sim\mathbb{W}_P(not\ A)$ □

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

Let l be a label occurrence in a term t in the scope of $n \geqslant 0$ negations. We say that l is an *odd* or an *even* occurrence if n is odd or even, respectively. We further say that l is a *strictly* even occurrence if it is even and $n > 0$.

Definition 7 (*Justification*)
Given a program P and a q-literal L, we say that a term E with no sums is a *(sufficient causal) justification* for L iff $E \leqslant \mathbb{W}_P(L)$. Odd (resp. strictly even) labels[2] in E are called *inhibitors* (resp. *enablers*) of E. A justification is said to be *inhibited* if it contains some inhibitor and it is said to be *enabled* otherwise. □

True atoms will have at least one enabled justification, whereas false atoms only contain disabled justifications. As an example of a query for a plain atom A, take the already seen explanation for p in Bond's example program P_1: $\mathbb{W}_{P_1}(p) = \mathbb{L}_{P_1}(p) = (\sim\sim h * d)\cdot r_1 + (\sim r_2 * d)\cdot r_1$. We have here two justifications for atom p, let us call them $E_1 = (\sim\sim h * d)\cdot r_1$ and $E_2 = (\sim r_2 * d)\cdot r_1$. Justification E_1 is enabled because it contains no inhibitors (in fact, E_1 is the unique real support for p). Moreover, h is an enabler in E_1 because it is strictly even (it is in the scope of double negation), whereas d is a productive cause, since it is not in the scope of any negation. On the contrary, E_2 is disabled because it contains the inhibitor r_2 (it occurs in the scope of one negation). Intuitively, r_2 has prevented $d\cdot r_1$ to become a justification of p. On the other hand, for atom a, we had $\mathbb{W}_{P_1}(a) = \sim h \cdot r_2$ that only contains an inhibited justification (being h the inhibitor), and so, atom a is not true. Now, if we query about the negative q-literal *not a*, we obtain $\mathbb{W}_{P_1}(not\ a) = \sim\mathbb{U}_{P_1}(a)$ which in this case happens to be $\sim\mathbb{L}_{P_1}(a) = \sim(\sim h \cdot r_2) = \sim\sim h + \sim r_2$. That is, q-literal *not a* holds, being enabled by h. Moreover, $\sim r_2$ points out that removing r_2 would suffice to cause *not a* too. It is easy to see that the explanations we can get for q-literals *not A* or *undef A* will have all their labels in the scope of negation (either as inhibitors or as enablers).

To illustrate a query for *undef A*, let us return to program P_2 whose standard WFM left both a and b undefined. Given the values, we obtained in the least and greatest fixpoints, the causal WFM will assign $\mathbb{W}_{P_2}(a) = \sim r_2\cdot r_1$ and $\mathbb{W}_{P_2}(b) = \sim r_1\cdot r_2$, that is, r_2 prevents r_1 to cause a and r_1 prevents r_2 to cause b. Furthermore, the values assigned to their respective negations, $\mathbb{W}_{P_2}(not\ a) = \sim r_1$ and $\mathbb{W}_{P_2}(not\ b) = \sim r_2$, point out that atoms a and b are not false because rules r_1 and r_2 have respectively prevented them to be so. Finally, we obtain that *undef a* is true because

$$\mathbb{W}_P(undef\ a) = \sim\mathbb{W}_{P_2}(a) * \sim\mathbb{W}_{P_2}(not\ a) = (\sim\sim r_2 + \sim r_1) * \sim\sim r_1 = \sim\sim r_2 * \sim\sim r_1$$

that is, rules r_1 and r_2 together have made a undefined. Similarly, b is also undefined because of rules r_1 and r_2, $\mathbb{W}_P(undef\ b) = \sim\sim r_1 * \sim\sim r_2$.

The next theorem shows that the literals satisfied by the standard WFM are precisely those ones containing at least one enabled justification in the causal WFM.

[2] We just mention labels, and not their occurrences because terms are in negation normal form and E contains no sums. Thus, having odd and even occurrences of a same label at a same time would mean that $E = 0$.

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

Theorem 2
Let P be a labelled logic program over a signature $\langle At, Lb\rangle$, where Lb is a finite set of labels and let W_P its (standard) WFM. A q-literal L holds with respect to W_P if and only if there is some enabled justification E of L, that is, $E \leqslant \mathbb{W}_P(L)$ and E does not contain odd negative labels. □

Back to our example program P_1, as we had seen, atom p had a unique enabled justification $E_1 = (\sim\sim h * d)\cdot r_1$. The same happens for atoms d and h whose respective justifications are just their own atom labels. Therefore, these three atoms hold in the standard WFM, W_{P_1}. On the contrary, as we discussed before, the only justification for a, $\mathbb{W}_{P_1}(a) = \sim h\cdot r_2$, is inhibited by h, and thus, a does not hold in W_{P_1}. The interest of an inhibited justification for a literal is to point out "potential" causes that have been prevented by some abnormal situation. In our case, the presence of $\sim h$ in $\mathbb{W}_{P_1}(a) = \sim h\cdot r_2$ points out that an exception h has prevented r_2 to cause a. When the exception is removed, the inhibited justification (after removing the inhibitors) becomes an enabled justification.

In our running example, if we consider a program P_3 obtained by removing the fact h from P_1, then $\mathbb{W}_{P_3}(a) = r_2$ points out that a has been caused by rule r_2 in this new scenario. This intuition about inhibited justifications is formalised as follows.

Definition 8
Given a term t in DNF, by $\varrho_x : \mathbf{V}_{Lb}^{CG} \longrightarrow \mathbf{V}_{Lb}^{CG}$, we denote the function that removes the elementary term x from t as follows:

$$\varrho_x(t) \stackrel{\text{def}}{=} \begin{cases} \varrho_x(u) \otimes \varrho_x(w) & \text{if } t = u \otimes v \text{ with } \otimes \in \{+, *, \cdot\} \\ 1 & \text{if } \sim\sim t \text{ is equivalent to } \sim\sim x \\ 0 & \text{if } t \text{ is equivalent to } \sim x \end{cases}$$

Note that we have assumed that t is in DNF. Otherwise, $\varrho_x(t) \stackrel{\text{def}}{=} \varrho_x(u)$, where u is an equivalent term in DNF. □

Theorem 3
Let P be a program over a signature $\langle At, Lb\rangle$, where Lb is a finite set of labels. Let Q be the result of removing from P all rules labelled by some $r_i \in Lb$. Then, the result of removing r_i from the justifications of some atom A with respect to program P are justifications of A with respect to Q, that is, $\varrho_{\sim r_i}(\mathbb{W}_P(A)) \leqslant \mathbb{W}_Q(A)$.

3 Relation to causal graph justifications

We discuss now the relation between ECJ and CG approaches. Intuitively, ECJ extends CG causal terms by the introduction of the new negation operator '$\sim$'. Semantically, however, there are more differences than a simple syntactic extension. A first minor difference is that ECJ is defined in terms of a WFM, whereas CG defines (possibly) several causal stable models. In the case of stratified programs, this difference is irrelevant, since the WFM is complete and coincides with the unique stable model. A second, more important difference is that CG exclusively considers productive causes in the justifications, disregarding additional information like the

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

inhibitors or enablers from ECJ. As a result, a false atom in CG has *no justification* – its causal value is 0 because there was no way to derive the atom. For instance, in program P_1, the only CG stable model I just makes $I(a) = 0$ and we lose the inhibited justification $\sim h \cdot r_2$ (default r_2 could not be applied). True atoms like p also lose any information about enablers: $I(p) = d \cdot r_1$ and nothing is said about $\sim\sim h$. Another consequence of the CG orientation is that negative literals *not A* are never assigned a cause (different from 0 or 1), since they cannot be "derived" or produced by rules. In the example, we simply get $I(not\ a) = 1$ and $I(not\ p) = 0$.

To further illustrate the similarities and differences between ECJ and CG, consider the following program P_4 capturing a variation of the Yale Shooting Scenario (Hanks and McDermott 1987).

$$
\begin{array}{llll}
d_{t+1} : dead_{t+1} & \leftarrow shoot_t,\ loaded_t,\ not\ ab_t & \overline{loaded}_0 & load_1 \\
l_{t+1} : loaded_{t+1} & \leftarrow load_t & \overline{dead}_0 & water_3 \\
a_{t+1} : ab_{t+1} & \leftarrow water_t & \overline{ab}_0 & shoot_8
\end{array}
$$

plus the following rules corresponding inertia axioms

$$F_{t+1} \leftarrow F_t,\ not\ \overline{F}_{t+1} \qquad \overline{F}_{t+1} \leftarrow \overline{F}_t,\ not\ F_{t+1}$$

for $F \in \{loaded, ab, dead\}$. Atoms of the form $\overline{A}$ represent the strong negation of A and we disregard models satisfying both A and $\overline{A}$. Atom $dead_9$ does not hold in the standard WFM of P_4, and so there is no CG-justification for it. Note here the importance of default reasoning. On the one hand, the default flow of events is that the turkey, Fred, continues to be alive when nothing threats him. Hence, we do not need a cause to explain why Fred is alive. On the other hand, shooting a loaded gun would normally kill Fred, being this a cause of its death. But, in this example, another exceptional situation – *water* spilled out – has *inhibited* this existing threat and allowed the world to flow as if nothing had happened (that is, following its default behaviour).

In the CG-approach, $dead_9$ is simply false by default and no justification is provided. However, a gun shooter could be "disappointed" since another conflicting default (shooting a loaded gun *normally* kills) has not worked. Thus, an expected answer for the shooter's question "why *not* $dead_9$?" is that $water_3$ broke the default, disabling d_9. In fact, ECJ yields the following inhibited justification for $dead_9$:

$$(\sim water_3 * shoot_8 * load_1 \cdot l_2) \cdot d_9 \tag{7}$$

meaning that $dead_9$ could not be derived because inhibitor $water_3$ prevented the application of rule d_9 to cause the death of Fred. Note that inertia rules are not labelled, which, as mentioned before, is syntactic sugar for rules with label 1. Since 1 is the identity of product and application, this has the effect of not being traced in the justifications. Note also that, according to Theorem 3, if we remove fact $water_3$ (the inhibitor) from P_4 leading to a new program P_5, then we get:

$$\mathbb{W}_{P_5}(dead_9) \quad = \quad (shoot_8 * load_1 \cdot l_2) \cdot d_9 \tag{8}$$

which is nothing else but the result of removing $\sim water_3$ from (7). In fact, the only CG stable model of P_5 makes this same assignment (8) which also corresponds

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

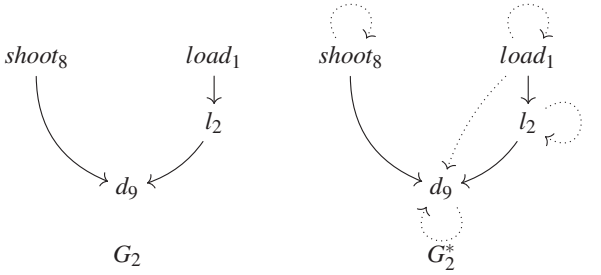

Fig. 3. G_2 is the cause of $dead_9$ in program P_4 while G_2^* is its associated causal graphs, that is, its reflexive and transitive closure.

to the causal graph depicted in Figure 3. In the general case, CG-justifications intuitively correspond to enabled justifications after forgetting all the enablers. Formally, however, there is one more difference in the definition of causal values: CG causal values are defined as ideals for the poset of a type of graphs formed by rule labels.

Definition 9 (*Causal graph*)
Given some set Lb of (rule) labels, a *causal graph* (*c-graph*) $G \subseteq Lb \times Lb$ is a reflexively and transitively closed set of edges. By $\mathbf{G}_{Lb}$, we denote the set of CG. Given two c-graphs G and G', we write $G \leqslant G'$ when $G \supseteq G'$. □

Intuitively, CG, like G_2 in Figure 3, are directed graphs representing the causal structure that has produced some event. Furthermore, $G \leqslant G'$ means that G contains enough information to yield the same effect as 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". Causes will be $\leqslant$-maximal (or $\subseteq$-minimal) CG. Formally, including reflexive and transitive edges allows to capture this intuitive relation simply by the subgraph relation. Note that, since CG are reflexively closed, every vertex has at least one edge (the reflexive one) and, thus, we can omit the set of vertices. Besides, for the sake of clarity, we only depict the minimum set of edges necessary for defining a causal graph (transitive and reflexive reduction). For instance, graph G_2 in Figure 3 is the transitive and reflexive reduction of the causal graph G_2^*.

Definition 10 (*CG Values in Cabalar et al. 2014a*)
Given a set of labels Lb, a *CG causal value* is any ideal (or lower-set) for the poset $\langle \mathbf{G}_{Lb}, \leqslant \rangle$. By $\mathbf{I}_{Lb}^{CG}$, we denote the set of CG causal values. Product '$*$', sums '$+$' and the $\leqslant$-order relation are defined as the set intersection, union and the subset relation, respectively. Application is given by $U \cdot U' \stackrel{\text{def}}{=} \{\, G'' \leqslant G \cdot G' \mid G \in U \text{ and } G' \in U' \,\}$. □

It has been shown in Fandinno (2015a) that CG values can be alternatively characterised as a free algebra generated by rule labels under the axioms of a complete distributive lattice plus the axioms of Figure 1.

Definition 11 (*CG Values in Fandinno 2015a*)
Given a set of labels Lb, a CG term is a term without negation '$\sim$'. *CG causal values* are the equivalence classes of CG terms for a completely distributive (complete)

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

lattice with meet '$*$' and join '$+$' plus the axioms of Figure 1. By $\mathbf{V}_{Lb}^{CG}$, we denote the set of CG causal values. □

Theorem 4 (*Causal values isomorphism from Fandinno 2015a*)
The function $term : \mathbf{I}_{Lb}^{CG} \longrightarrow \mathbf{V}_{Lb}^{CG}$ given by

$$term(U) \;\mapsto\; \sum_{G \in U} \prod_{(v_1,v_2) \in G} v_1 \cdot v_2$$

is an isomorphism between algebras $\langle \mathbf{I}_{Lb}^{CG}, +, *, \cdot, \mathbf{G}_{Lb}, \emptyset \rangle$ and $\langle \mathbf{V}_{Lb}^{CG}, +, *, \cdot, 1, 0 \rangle$. □

Theorem 4 states that CG causal values can be equivalently described either as ideals of CG or as elements of an algebra of terms. Furthermore, by abuse of notation, by G, we also denote the ideal whose maximum element is G, corresponding to $term(G)$ as well. For instance, for the causal graph G_2 in Figure 3, it follows $G_2 = term(G_2) = term(\downarrow G_2)$ with $\downarrow G_2$ the ideal whose maximum element is G_2. Moreover, from the equivalences in Figure 1, it also follows that

$$\begin{aligned} G_2 &= shoot_8 \cdot d_9 \;*\; load_1 \cdot l_2 \;*\; l_2 \cdot d_9 \;*\; \alpha \\ &= shoot_8 \cdot d_9 \;*\; load_1 \cdot l_2 \;*\; l_2 \cdot d_9 \\ &= shoot_8 \cdot d_9 \;*\; load_1 \cdot l_2 \cdot d_9 \\ &= (shoot_8 \;*\; load_1 \cdot l_2) \cdot d_9 \end{aligned}$$

where $\alpha = load_1 \cdot d_9 * shoot_8 \cdot shoot_8 * d_9 \cdot d_9 * load_1 \cdot load_1 * l_2 \cdot l_2 * d_9 \cdot d_9$ is a term that, as we can see, can be ruled out and corresponds to the transitive and reflexive doted edges in G_2^*. That is, justification (8) associated to atom $dead_9$ by the causal WFM of program P_5 actually corresponds to causal graph G_2.

Theorem 4 also formalises the intuition that opens this section: ECJ extends CG causal terms by the introduction of the new negation operator '$\sim$'. We formalise next the correspondence between CG and ECJ justifications.

Definition 12 (*CG mapping*)
We define a mapping $\lambda^c : \mathbf{V}_{Lb} \longrightarrow \mathbf{V}_{Lb}^{CG}$ from ECJ values into CG values in the following recursive way:

$$\lambda^c(t) \stackrel{\text{def}}{=} \begin{cases} \lambda^c(u) \otimes \lambda^c(w) & \text{if } t = u \otimes v \text{ with } \otimes \in \{+, *, \cdot\} \\ 1 & \text{if } t = {\sim}{\sim} l \text{ with } l \in Lb \\ 0 & \text{if } t = \;\; {\sim} l \text{ with } l \in Lb \\ l & \text{if } t = \;\;\; l \text{ with } l \in Lb \end{cases}$$

Note that we have assumed that t is in DNF. Otherwise, $\lambda^c(t) \stackrel{\text{def}}{=} \lambda^c(u)$, where u is an equivalent term in DNF. □

Function λ^c maps every negated label $\sim l$ to 0 (which is the annihilator of both product '$*$' and application '$\cdot$' and the identity of addition '$+$'). Hence λ^c removes all the inhibited justifications. Furthermore λ^c maps every doubly negated label $\sim\sim l$ to 1 (which is the identity of both product '$*$' and application '$\cdot$'). Therefore λ^c removes all the enablers (i.e. doubly negated labels $\sim\sim l$) for the remaining (i.e. enabled) justifications.

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

A CG interpretation is a mapping $\tilde{I} : At \longrightarrow \mathbf{V}_{Lb}^{CG}$. The value assigned to a negative literal *not A* by a CG interpretation $\tilde{I}$, denoted as $\tilde{I}(not\ A)$, is defined as: $\tilde{I}(not\ A) \stackrel{\text{def}}{=} 1$ if $\tilde{I}(A) = 0$; $\tilde{I}(not\ A) \stackrel{\text{def}}{=} 0$ otherwise. A CG interpretation $\tilde{I}$ is a CG model of rule like (4) iff

$$\left(\tilde{I}(B_1) * \ldots * \tilde{I}(B_m) * \tilde{I}(not\ C_1) * \ldots * \tilde{I}(not\ C_n) \right) \cdot r_i \ \leqslant \ \tilde{I}(H) \tag{9}$$

Notice that the value assigned to a negative literal by CG and ECJ interpretations is different. According to Cabalar *et al.* (2014a), a CG interpretation $\tilde{I}$ is a *CG stable model* of a program P iff $\tilde{I}$ is the least model of the program $P^{\tilde{I}}$. In the following, we provide an ECJ-based characterisation of the CG stable models that will allow us to relate both approaches. By $\lambda^c(I)$, we will denote a CG interpretation $\tilde{I}$ s.t. $\tilde{I}(A) = \lambda^c(I(A))$ for every atom A.

Definition 13 (*CG stable models*)
Given a program P, a CG interpretation $\tilde{I}$ is a *CG stable model* of P iff there exists a fixpoint I of the operator Γ_P^2, i.e. $\Gamma_P(\Gamma_P(I)) = I$, such that $\tilde{I} = \lambda^c(I) = \lambda^c(\Gamma_P(I))$. □

Theorem 5
Let P be a program over a signature $\langle At, Lb \rangle$, where Lb is a finite set of labels. Then, the CG stable models (Definition 13) are exactly the causal values and causal stable models defined in Cabalar *et al.* (2014a). □

Theorem 5 shows that Definition 13 is an alternative definition of CG causal stable models. Furthermore, it settles that every causal model corresponds to some fixpoint of the operator Γ_P^2. Therefore, for every enabled justification there is a corresponding CG-justification common to all stable models. In order to formalise this idea, we just take the definition of causal explanation from Cabalar *et al.* (2014b).

Definition 14 (*CG-justification*)
Given an interpretation I, we say that a c-graph G is a *(sufficient) CG-justification* for an atom A iff $term(G) \leqslant \tilde{I}(A)$. □

Since $term(\cdot)$ is a one-to-one correspondence, we can define its inverse $graph(v) \stackrel{\text{def}}{=} term^{-1}(v)$ for all $v \in \mathbf{V}_{Lb}^{CG}$.

Theorem 6
Let P be a program over a signature $\langle At, Lb \rangle$, where Lb is a finite set of labels. For any enabled justification E of some atom A w.r.t. $\mathbb{W}_P$, i.e. $E \leqslant \mathbb{W}_P(A)$, there is a CG-justification $G \stackrel{\text{def}}{=} graph(\lambda^c(E))$ of A with respect to any stable model $\tilde{I}$ of P. □

As happens between the (standard) well-founded and stable model semantics, the converse of Theorem 6 does not hold in general. That is, we may get a justification that is common to all CG-stable models but does not occur in the ECJ WFM. For instance, let P_6 be the program consisting on the following rules:

$r_1 : \ a \leftarrow not\ b \qquad r_2 : \ b \leftarrow not\ a,\ not\ c \qquad c \qquad r_3 : \ c \leftarrow a \qquad r_4 : \ d \leftarrow b,\ not\ d$

The (standard) WFM of program P_6 is two-valued and corresponds to the unique (standard) stable model $\{a, c\}$. Furthermore, there are two causal explanations of

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

c with respect to this unique stable model: the fact c and the pair of rules $r_1 \cdot r_3$. Note that when c is removed, $\{a, c\}$ is still the unique stable model, but all atoms are undefined in the WFM. Hence, $r_1 \cdot r_3$ is a justification with respect to the unique stable model of the program, but not with respect to its WFM.

4 Relation to why-not provenance

An evident similarity between ECJ and WnP approaches is the use of an alternating fixpoint operator (Van Gelder 1989) which has been actually borrowed from WnP. However, there are some slight differences. A first one is that we have incorporated from CG the non-commutative operator '$\cdot$' which allows capturing not only which rules justify a given atom, but also the dependencies among these rules. The second is the use of a *non-classical* negation '$\sim$' that is crucial to distinguish between productive causes and enablers. This distinction cannot be represented with the classical negation '$\neg$' in WnP since double negation can always be removed. Apart from the interpretation of negation in both formalisms, there are other differences too. As an example, let us compare the justifications we obtain for $dead_9$ in program P_5. While for ECJ, we obtained (8) (or graph G_2 in Figure 3), the corresponding WnP justification has the form:

$$\begin{aligned} &l_2 \wedge d_9 \wedge load_1 \wedge shoot_8 \\ &\quad \wedge not(ab_1) \wedge not(ab_2) \wedge \ldots \wedge not(ab_7) \wedge not(water_0) \wedge \ldots \wedge not(water_6) \end{aligned} \tag{10}$$

A first observation is that the subexpression $l_2 \wedge d_9 \wedge load_1 \wedge shoot_8$ constitutes, informally speaking, a "flattening" of (8) (or graph G_2), where the ordering among rules has been lost. We get, however, new labels of the form $not(A)$ meaning that atom A is required not to be a program fact, something that is not present in CG-justifications. For instance, (10) points out that *water* can not be spilt on the gun along situations $0, \ldots, 7$. Although this information can be useful for debugging (the original purpose of WnP), its inclusion in a causal explanation is obviously inconvenient from a knowledge representation perspective, since it explicitly *enumerates all the defaults* that were applied (no water was spilt at any situation) something that may easily blow up the (causally) irrelevant information in a justification.

An analogous effect happens with the enumeration of exceptions to defaults, like inertia. Take program P_7 obtained from P_4 by removing all the performed actions, i.e. facts $load_1$, $water_3$ and $shoot_7$. As expected, Fred will be alive, $\overline{dead}_t$, at any situation t by inertia. ECJ will assign no cause for $dead_t$, not even any inhibited one, i.e. $\mathbb{W}_P(\overline{dead}_t) = 1$ and $\mathbb{W}_P(dead_t) = 0$ for any t. The absence of labels in $\mathbb{W}_P(\overline{dead}_t) = 1$ is, of course, due to the fact that inertia axioms are not labelled, as they naturally represent a default and not a causal law. Still, even if inertia were labelled, say, with in_k per each situation k, we would obtain a *unique cause* for $\mathbb{W}_P(\overline{dead}_t) = in_1 \cdot \ \ldots \ \cdot in_t$ for any $t > 0$ while maintaining no cause for $\mathbb{W}_P(dead_t) = 0$. However, the number of minimal WnP justifications of $dead_t$ grows quadratically, as it collects *all the plans* for killing Fred in t steps loading and

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

shooting once. For instance, among others, all the following:

$$d_9 \wedge \neg not(load_0) \wedge r_2 \wedge \neg not(shoot_1) \wedge not(water_0) \wedge \quad not(ab_1)$$
$$d_9 \wedge \neg not(load_0) \wedge r_2 \wedge \neg not(shoot_2) \wedge not(water_0) \wedge not(water_1) \wedge not(ab_1) \wedge not(ab_2)$$
$$d_9 \wedge \neg not(load_1) \wedge r_2 \wedge \neg not(shoot_3) \wedge not(water_0) \wedge$$
$$\wedge \ not(water_1) \ \wedge not(water_2) \wedge \quad not(ab_1) \quad \wedge not(ab_2) \wedge not(ab_3)$$
$$\ldots$$

are WnP-justifications for $dead_9$. The intuitive meaning of expressions of the form $\neg not(A)$ is that $dead_9$ can be justified by adding A as a fact to the program. For instance, the first conjunction means that it is possible to justify $dead_9$ by adding the facts $load_0$ and $shoot_1$ and not adding the fact $water_0$. We will call these justifications, which contain a subterm of the form $\neg not(A)$, *hypothetical* in the sense that they involve some hypothetical program modification.

Definition 15 (*Provenance values*)
Given a set of labels Lb, a *provenance term* t is recursively defined as one of the following expressions $t ::= l \mid \prod S \mid \sum S \mid \neg t_1$, where $l \in Lb$, t_1 is in its turn a provenance term and S is a (possibly empty and possible infinite) set of provenance terms. *Provenance values* are the equivalence classes of provenance terms under the equivalences of the Boolean algebra. We denote by $\mathbf{B}_{Lb}$ the set of provenance values over Lb. □

Informally speaking, with respect to ECJ, we have removed the application '$\cdot$' operator, whereas product '$*$' and addition '$+$' hold the same equivalences as in Definition 2 and negation '$\sim$' has been replaced by '$\neg$' from Boolean algebra. Thus, '$\neg$' is classical and satisfies all the axioms of '$\sim$' plus $\neg\neg t = t$. Note also that, in the examples, we have followed the convention from Damásio *et al.* (2013) of using the symbols '$\wedge$' and '$\vee$' to respectively represent meet and join. However, in formal definitions, we will keep respectively using '$*$' and '$+$' for that purpose. We define a mapping $\lambda^p : \mathbf{V}_{Lb} \longrightarrow \mathbf{B}_{Lb}$ in the following recursive way:

$$\lambda^p(t) \stackrel{\text{def}}{=} \begin{cases} \lambda^p(u) \otimes \lambda^p(w) & \text{if } t = u \otimes v \text{ with } \otimes \in \{+, *\} \\ \lambda^p(u) * \lambda^p(w) & \text{if } t = u \cdot v \\ \neg\lambda^p(u) & \text{if } t = \sim u \\ l & \text{if } t = l \text{ with } l \in Lb \end{cases}$$

Definition 16 (*Provenance*)
Given a program P, the WnP program $\mathfrak{P}(P) \stackrel{\text{def}}{=} P \cup P'$, where P' contains a labelled fact of the form $(\sim not(A) : A)$ for each atom $A \in At$ not occurring in P as a fact. We will write $\mathfrak{P}$ instead of $\mathfrak{P}(P)$ when the program P is clear by the context. We denote by $Why_P(L) \stackrel{\text{def}}{=} \lambda^p(\mathbb{W}_{\mathfrak{P}}(L))$ the WnP of a q-literal L. We also say that a justification is hypothetical when $not(A)$ occurs oddly negated in it, non-hypothetical otherwise. □

Theorem 7

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

Let P be program over a finite signature $\langle At, Lb\rangle$. Then, the provenance of a literal according to Definition 16 is equivalent to the provenance defined by Damásio *et al.* (2013). □

Theorem 8
Let P be program over a finite signature $\langle At, Lb\rangle$. $\mathbb{W}_P$ is the result of removing all non-hypothetical justification from $\mathbb{W}_{\mathfrak{P}}$ and each occurrence of the form $\sim\sim not(A)$ for the remaining ones, that is, $\mathbb{W}_P = \varrho(\mathbb{W}_{\mathfrak{P}})$, where ϱ is the result of removing every label of the form $not(A)$, that is ϱ is the composition of $\varrho_{not(A_1)} \circ \varrho_{not(A_2)} \circ \ldots \circ \varrho_{not(A_n)}$ with $At = \{A_1, A_2, \ldots, A_n\}$. □

On the one hand, Theorem 7 shows that the provenance of a literal can be obtained by replacing the negation '$\sim$' by '$\neg$' and '$\cdot$' by '$*$' in the causal WFM of the augmented program $\mathfrak{P}$. On the other hand, Theorem 8 asserts that non-hypothetical justifications of a program and its augmented one coincide when subterms of the form $\sim\sim not(A)$ are removed from justifications of the latter. Consequently, we can establish the following correspondence between the ECJ justifications and the non-hypothetical WnP justifications.

Theorem 9
Let P be program over a finite signature $\langle At, Lb\rangle$. Then, the ECJ justifications of some atom A (after replacing '$\cdot$' by '$*$' and '$\sim$' by '$\neg$') correspond to the WnP justifications of A (after removing every label of the form $not(B)$ with $B \in At$), that is, $\lambda^p(\mathbb{W}_P)(A) = \varrho(Why_P)(A)$, where ϱ is the result of removing every label of the form $not(A)$ as in Theorem 8. □

Theorem 9 establishes a correspondence between non-hypothetical WnP-justifications and (flattened) ECJ justifications. In our running example, (7) is the unique causal justification of $dead_9$, while (11) (below) is its unique non-hypothetical WnP justification.

$$\neg water_3 \wedge shoot_8 \wedge load_1 \wedge l_2 \wedge d_9 \wedge \\ \wedge not(dead_1) \wedge \ldots \wedge not(dead_9) \wedge not(ab_1) \wedge \ldots \wedge not(ab_8) \tag{11}$$

It is easy to see that, by applying λ^p to (7), we obtain

$$\lambda^p\big((\sim water_3 * shoot_8 * load_1 \cdot l_2) \cdot d_9\big) \;=\; \neg water_3 \wedge shoot_8 \wedge load_1 \wedge l_2 \wedge d_9 \tag{12}$$

which is just the result of removing all labels of the form '$not(A)$' from (11). The correspondence between the ECJ justification (8) and the WnP justification (10) for program P_5 can be easily checked in a similar way.

Hypothetical justifications are not directly captured by ECJ, but can be obtained using the augmented program $\mathfrak{P}$ as stated by Theorem 7. As a byproduct, we establish a formal relation between WnP and CG.

Theorem 10
Let P be a program over a finite signature $\langle At, Lb\rangle$. Then, every non-hypothetical and enabled WnP-justification D of some atom A (after removing every label of the form $not(B)$ with $B \in At$) is a justification with respect to every CG stable

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

model $\tilde{I}$ (after replacing '$\cdot$' by '$*$' and '$\sim$' by '$\neg$'), that is $D \leqslant Why_P(A)$ implies $\varrho(D) \leqslant \lambda^P(\tilde{I})(A)$, where ϱ is the result of removing every label of the form $not(B)$ as in Theorem 8. □

Note that, as happened between the ECJ and CG justifications, the converse of Theorem 10 does not hold in general due to the well-founded versus stable model difference in their definitions. As an example, the explanation for atom c at program P_6 has a unique WnP justification c as opposed to the two CG justifications, c and $r_1 \cdot r_3$.

5 Contributory causes

Intuitively, a *contributory cause* is an event that has helped to produce some effect. For instance, in program P_5, it is easy to identify both actions, $load_1$ and $shoot_8$, as events that have helped to produce $dead_9$ and, thus, they are both contributory causes of Fred's death. We may define the above informal concept of contributory cause as: any non-negated label l that occurs in a maximal enabled justification of some atom A. Similarly, a *contributory enabler* can be defined as a doubly negated label $\sim\sim l$ that occurs in a maximal enabled justification of some atom A. These definitions correctly identify $load_1$ and $shoot_8$ as contributory causes of $dead_9$ in program P_5 and d as a contributory cause of p in program P_1. Fact h is considered a contributory enabler of p. These definitions will also suffice for dealing with what Hall (2007) calls *trouble cases*: *non-existent threats*, *short-circuits*, *late-preemption* and *switching examples*.

It is worth to mention that, in the philosophic and Artificial Intelligence literature, the concept of contributory cause is usually discussed in the broader sense of *actual causation* which tries to provide an *unique everyday-concept* of causation. Pearl (2000) studied actual and contributory causes relying on *causal networks*. In this approach, it is possible to conclude cause–effect relations like "A has been an actual (resp. contributory) cause of B" from the behaviour of structural equations by applying, under some *contingency* (an alternative model in which some values are fixed), the *counterfactual dependence* interpretation from Hume (1748): "had A not happened, B would not have happened". Consider the following example which illustrates the difference between contributory and actual causes under this approach.

Example 1 (*Firing squad*)
Suzy and Billy form a two-man firing squad that responds to the order of the captain. The shot of any of the two riflemen would kill the prisoner. Indeed, the captain gives the order, both riflemen shoot together and the prisoner dies. □

On the one hand, the captain is an actual cause of the prisoner's death: "had the captain not given the order, the riflemen would not have shot and the prisoner would not have died". On the other hand, each rifleman alone is not an actual cause: "had one rifleman not shot, the prisoner would have died anyway because of the other rifleman". However, each rifleman's shot is a contributory cause because, under the contingency where the other rifleman does not shoot, the prisoner's death manifests counterfactual dependence on the first rifleman's shot. Later approaches

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

like Halpem and Pearl (2001; 2005), Hall (2004) and Hall (2007) have not made this distinction and consider the captain and the two riflemen as actual causes of the prisoner's death, while Halpern (2015) considers the captain and the conjunction of both riflemen's shoots, but not each of them alone, as actual causes. We will focus here on representing the above concept of contributory cause and leave to the reader whether this agrees with the concept of cause in the every-day discourse or not.

As has been slightly discussed in the introduction, Hall (2004; 2007), has emphasised the difference between two types of causal relations: *dependence* and *production*. The former relies on the idea that "counterfactual dependence between wholly distinct events is sufficient for causation". The latter is characterised by being *transitive*, *intrinsic* (two processes following the same laws must be both or neither causal) and *local* (causes must be connected to their effects via sequences of causal intermediates).

These two concepts can be illustrated in Bond's example by observing the difference between pouring the drug (atom d), which is a cause under both understandings, and being a holiday (atom h), which is not considered a cause under the production viewpoint, although it is considered a cause under the dependence one.

In this sense, all the above approaches to actual causation, but Hall (2004), can be classified in the dependence category. ECJ and CG do not consider h a productive cause of d because the *default* (or *normal*) behaviour of rule (1) is that "d causes p". This default criterion is also shared by Hall (2007), Halpern (2008), Hitchcock and Knobe (2009) and Halpern and Hitchcock (2011). Note that, ECJ (but not CG) captures the fact that d counterfactually depends on h, as it considers it an enabler. In Hall (2004), the author relies on intrinsicness for rejecting h as a productive cause of d: any causal structure (justification) including h and p would have to include the absence of the antidote (atom a), and it would be enough that Bond had taken the antidote by another reason to break the counterfactual dependence between h and p. By applying the above contributory cause definition to the WnP justification $h \wedge d \wedge r_1$ of Bond's paralysis (atom p) in program P_1, we can easily identify that h is being considered a cause in WnP, thus, a causal interpretation of WnP clearly follows the dependence-based viewpoint. On the other hand, the unique CG justification $d{\cdot}r_1$ only considers d as a cause, which illustrates the fact that CG is mostly related to the concept of production. ECJ combines both understandings, and what is a cause under the dependence viewpoint is either an enabler or a cause under the production viewpoint.

In order to illustrate how ECJ can be used for representing the so-called *non-existent threat* scenarios, consider a variation of Bond's example where today is not a holiday and, thus, Bond takes the antidote. The poured drug d is a threat to Bond's safety, represented as s, but that threat is prevented by the antidote. We may represent this scenario by program P_8 below:

$$\begin{array}{lll} r_1: & p \leftarrow d, not\ a & d \\ r_2: & a \leftarrow not\ h & \\ r_3: & s \leftarrow not\ p & \end{array}$$

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

The causal WFM of program P_8 assigns

$$\mathbb{W}_{P_8}(s) \;=\; {\sim}{\sim}r_2{\cdot}r_3 \;+\; {\sim}d{\cdot}r_3 \;+\; {\sim}r_1{\cdot}r_3$$

which recognises rule r_2 (taking the antidote) as a contributory enabler of Bond's safety. The difficulty in this kind of scenarios consists in avoiding the wrong recognition of r_2 as an enabler when the threat d does not exist. If we remove fact d from P_8 to get the new program P_9, then we obtain that $\mathbb{W}_{P_9}(d) = 0$ and, consequently, $\mathbb{W}_{P_9}(s) = r_3$. Intuitively, in the absence of any threat, Bond is just safe because that is his default behaviour as stated by rule r_3.

Short-circuit examples consist in avoiding the wrong recognition of an event as a contributory enabler that provokes a threat that eventually prevents itself. Consider the program P_{10} below:

$$\begin{array}{lll} r_1: & p \leftarrow a, \mathit{not}\ f & a \\ r_2: & f \leftarrow c, \mathit{not}\ b & c \\ r_3: & b \leftarrow c & \end{array}$$

Here, c is a threat to p, since it may cause f through rule r_2. However, c eventually prevents r_2, since it also causes b through rule r_3. The causal WFM of program P_{10} assigns

$$\mathbb{W}_{P_{10}}(p) \;=\; (a * {\sim}{\sim}r_3){\cdot}r_1 \;+\; (a * {\sim}r_2){\cdot}r_1 \;+\; (a * {\sim}c){\cdot}r_1$$

which correctly avoids considering c as a contributory enabler of p and recognises r_3 as the enabler of p. Note that c is actually considered an inhibitor due to justification $(a * {\sim}c){\cdot}r_1$ pointing out that, had c not happened, then $a{\cdot}r_1$ would have been an enabled justification. But then, $(a * {\sim}{\sim}r_3){\cdot}r_1$ would stop being a justification since $(a * {\sim}{\sim}r_3){\cdot}r_1 + a{\cdot}r_1 = a{\cdot}r_1$.

To illustrate *late-preemention,* consider the following example from Lewis (2000).

Example 2 (*Rock throwers*)
Billy and Suzy throw rocks at a bottle. Suzy throws first and her rock arrives first. The bottle shattered. When Billy's rock gets to where the bottle used to be, there is nothing there but flying shards of glass. Who has caused the bottle to shatter? □

The key of this example is to recognise that Suzy, and not Billy, has caused the shattering. The usual way of representing this scenario in the actual causation literature is by introducing two new fluents *hit_suzy* and *hit_billy* in the following way (Hall 2007; Halpern and Hitchcock 2011; Halpern 2014; Halpern 2015):

$$\mathit{hit\_suzy} \;\leftarrow\; \mathit{throw\_suzy} \tag{13}$$

$$\mathit{hit\_billy} \;\leftarrow\; \mathit{throw\_billy}, \neg\mathit{hit\_suzy} \tag{14}$$

$$\mathit{shattered} \;\leftarrow\; \mathit{hit\_suzy} \tag{15}$$

$$\mathit{shattered} \;\leftarrow\; \mathit{hit\_billy} \tag{16}$$

It is easy to see that such a representation mixes, in law (14), both the description of the world and the narrative fact asserting that Suzy threw first. This may easily lead to a problem of elaboration tolerance. For instance, if we have N shooters

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

and they shoot sequentially, we would have to modify the equations for all of them in an adequate way, so that the last shooter's equation would have the negation of the preceding $N-1$ and so on. Moreover, all these equations would have to be *reformulated* if we simply change the shooting order. On the other hand, we may represent this scenario by a program P_{11} consisting of the following rules

$$\begin{array}{lll} s_{t+1} : \mathit{shattered}_{t+1} \leftarrow \mathit{throw}(A)_t, \mathit{not}\ \ \mathit{shattered}_t & & \mathit{throw}(\mathit{suzy})_0 \\ \overline{\mathit{shattered}}_0 & & \mathit{throw}(\mathit{billy})_1 \end{array}$$

with $A \in \{\mathit{suzy}, \mathit{billy}\}$, plus the following rules corresponding to the inertia axioms

$$\begin{array}{rcl} \mathit{shattered}_{t+1} & \leftarrow & \mathit{shattered}_t, \mathit{not}\ \ \overline{\mathit{shattered}}_{t+1} \\ \overline{\mathit{shattered}}_{t+1} & \leftarrow & \overline{\mathit{shattered}}_t, \mathit{not}\ \ \mathit{shattered}_{t+1} \end{array}$$

Atom $\mathit{shattered}_2$ holds in the standard WFM of P_{11} and its justification corresponds to

$$\mathit{throw}(\mathit{suzy})_0 \cdot s_1 \ + \ (\sim \mathit{throw}(\mathit{suzy}) * \mathit{throw}(\mathit{billy})_1) \cdot s_2 \ + \ (\sim s_1 * \mathit{throw}(\mathit{billy})_1) \cdot s_2$$

On the one hand, the first addend points out that fact $\mathit{throw}(\mathit{suzy})_0$ has caused $\mathit{shattered}_2$ by means of rule s_1. On the other hand, the second addend indicates that $\mathit{throw}(\mathit{billy})_1$ has not caused it because Suzy's throw has prevented it. Finally, the third addend means that $\mathit{throw}(\mathit{billy})_1$ would have caused the shattering if it were not for rule s_1. This example shows how our semantics is able to recognise that it was Suzy, and not Billy, who caused the bottle shattering. Furthermore, it also explains that Billy did not cause it because Suzy did it first.

Finally, consider the following example from Hall (2000).

Example 3 (*The engineer*)
An engineer is standing by a switch in the railroad tracks. A train approaches in the distance. She flips the switch, so that the train travels down the right-hand track, instead of the left. Since the tracks reconverge up ahead, the train arrives at its destination all the same; let us further suppose that the time and manner of its arrival are exactly as they would have been, had she not flipped the switch. □

This has been a controversial example. In Hall (2000), the author has argued that the switch should be considered a cause of the arrival because *switch* has contributed to the fact that the train has travelled down the right-hand track. In a similar manner, it seems clear that the train travelling down the right-hand track has contributed to the train arrival. If causality is considered to be a transitive relation, as Hall (2000) does, the immediate consequence of the above reasoning is that flipping the *switch* has contributed to the train *arrival*. In Hall (2007), he argues otherwise and points out that commonsense tells that the *switch* is not a cause of the *arrival*. Halpern and Pearl (2005) had considered *switch* a cause of *arrival* depending on whether the train travelling down the tracks is represented by one or two variables in the model. Although our understanding of causality is closer to the one expressed in Hall (2007), it is not the aim of this work to go more in depth in this discussion, but to show instead how both understandings can be represented in

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

ECJ. Consider the following program P_{12}

$$\begin{array}{lll} r_1 : arrival \leftarrow right & & \overline{switch} \leftarrow not\ switch \\ r_2 : arrival \leftarrow left & & switch \\ r_3 : right \leftarrow train, not\ \overline{switch} & & train \\ r_4 : left \leftarrow train, not\ switch & & \end{array}$$

where $\overline{switch}$ represents the strong negation of *switch*. The unlabelled rule capture the idea that the switch behaves classically, that is, it must be activated or not. The literal *not* $\overline{switch}$ in the body of rule r_3 points out that the *switch* position is an enabler and not a cause of the track taken by the train. This representation can be arguable, but the way in which the rule has been written would be expressing that if a train is coming, then a train will cross the right track by default unless $\overline{switch}$ prevents it. In that sense, the only productive cause for *right* (a train in the right track) is *train* (a train is coming), whereas the switch position just enables the causal rule to be applied. A similar default r_4 is built for the left track, flipping the roles of *switch* and $\overline{switch}$.

The causal WFM of program P_{12} corresponds to

$$\mathbb{W}_{P_{12}}(arrival) \quad = \quad (train * {\sim}{\sim}switch){\cdot}r_3{\cdot}r_1 \ + \ (train * {\sim}switch){\cdot}r_4{\cdot}r_2$$

It is easy to see that *switch* is a doubly negated label occurring in the maximal enabled justification $E_1 = (train * {\sim}{\sim}switch){\cdot}r_3{\cdot}r_1$ and, thus, we may identify it as a contributory enabler of *arrival*, but not its productive cause. On the other hand, by looking at the inhibited justification $E_2 = (train * {\sim}switch){\cdot}r_4{\cdot}r_2$, we observe that *switch* is also preventing rules r_4 and r_2 to produce the same effect, *arrival*, that is helping to produce in E_1.

If we want to ignore the way in which the train arrives, one natural possibility is using the same label for all the rules for atom *arrival*, reflecting in this way that we do not want to trace whether r_1 or r_2 has been actually used. Suppose we label r_2 with r_1 instead, leading to the new program P_{13}

$$\begin{array}{lll} r_1 : arrival \leftarrow right & & \overline{switch} \leftarrow not\ switch \\ r_1 : arrival \leftarrow left & & switch \\ r_3 : right \leftarrow train, not\ \overline{switch} & & train \\ r_3 : left \leftarrow train, not\ switch & & \end{array}$$

whose causal WFM corresponds to

$$\mathbb{W}_{P_{12}}(arrival) \quad = \quad (train * {\sim}{\sim}switch){\cdot}r_3{\cdot}r_1 \ + \ (train * {\sim}switch){\cdot}r_3{\cdot}r_1 \quad = \quad train{\cdot}r_3{\cdot}r_1$$

As we can see, this justification does not consider *switch* at all as a cause of the arrival (nor even a contributory enabler, as before). In other words, *switch* is *irrelevant* for the train *arrival*, which probably coincides with the most common intuition.

However, we do not find this solution fully convincing yet, because the explanation we obtain for *right*, $\mathbb{W}_{P_{13}}(right) = (train * {\sim}{\sim}switch){\cdot}r_3$ is showing that *switch* is just acting as an enabler, as we commented before. If we wanted to represent *switch* as a contributory cause of *right*, we would have more difficulties to simultaneously keep

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

switch irrelevant in the explanation of *arrival*. One possibility we plan to explore in the future is allowing the declaration of a given atom or fluent, like our *switch*, as *classical* so that we include both, the rule:

$$switch \leftarrow not\ not\ switch$$

in the logic program[3] and the axiom $\sim\sim switch = switch$ in the algebra. The latter immediately implies $switch + \sim switch = 1$ (due to the weak excluded middle axiom).

Then, P_{13} could be simply expressed as

$$\begin{array}{lll} r_1 : arrival \leftarrow right & & train \\ r_1 : arrival \leftarrow left & & switch \\ r_3 : right \leftarrow train, switch & & \\ r_3 : left \leftarrow train, not\ switch & & \end{array}$$

and the justification of *right* and *left* would become

$$\mathbb{W}_P(right) = (train * switch)\cdot r_3 \qquad \mathbb{W}_P(left) = (train * \sim switch)\cdot r_3$$

pointing out that *switch* is a cause (resp. an inhibitor) of the train travelling down the right (resp. left) track. Then, the justification of arrival would be

$$\mathbb{W}_P(arrival) = (train * switch)\cdot r_3\cdot r_1 + (train * \sim switch)\cdot r_3\cdot r_1 = train\cdot r_3\cdot r_1$$

We leave the study of this possibility for a future deeper analysis.

6 Conclusions and other related work

In this paper, we have introduced a unifying approach that combines causal production with enablers and inhibitors. We formally capture inhibited justifications by introducing a "non-classical" negation '$\sim$' in the algebra of CG. An inhibited justification is nothing else but an expression containing some negated label. We have also distinguished productive causes from enabling conditions (counterfactual dependences that are not productive causes) by using a double negation '$\sim\sim$' for the latter. The existence of enabled justifications is a sufficient and necessary condition for the truth of a literal. Furthermore, our justifications capture, under the well-founded semantics, both Causal Graph and WnP justifications. As a byproduct we established a formal relation between these two approaches.

We have also shown how several standard examples from the literature on actual causation can be represented in our formalism and illustrated how this representation is suitable for domains which include dynamic defaults – those whose behaviour are not predetermined, but rely on some program condition – as for instance the inertia axioms. As pointed out by Maudlin (2004), causal knowledge can be structured by a combination of *inertial laws* – how the world would evolve if nothing intervened – and *deviations* from these inertial laws.

[3] This implication actually corresponds to a *choice rule* $0\{switch\}1$, commonly used in Answer Set Programming.

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press

In addition to the literature on actual causes cited in Section 5, our work also relates to papers on reasoning about actions and change (Lin 1995; McCain and Turner 1997; Thielscher 1997). These works have been traditionally focused on using causal inference to solve representational problems (such as, the frame, ramification and qualification problems) without paying much attention to the derivation of cause–effect relations. Focusing on LP, our work obviously relates to explanations obtained from Answer Set Programming debugging approaches (Denecker and Schreye 1993; Specht 1993; Pemmasani *et al.* 2004; Gebser *et al.* 2008; Pontelli *et al.* 2009; Oetsch *et al.* 2010; Schulz and Toni 2016). The most important difference of these works with respect to ECJ, and also WnP and CG, is that the last three provide fully algebraic semantics in which justifications are embedded into program models. A formal relation between Pontelli *et al.* (2009) and WnP was established in Damásio *et al.* (2013) and so, using Theorems 7 and 9, it can be directly extended to ECJ, but at the cost of flattening the graph information (i.e. losing the order among rules).

Interesting issues for future study are incorporating enabled and inhibited justifications to the stable model semantics and replacing the syntactic definition in favour of a logical treatment of default negation, as done for instance with the Equilibrium Logic (Pearce 1996) characterisation of stable models. Other natural steps would be the consideration of syntactic operators, for capturing more specific knowledge about causal information as done in (Fandinno 2015b) capturing sufficient causes in the CG approach, and also the representation of non-deterministic causal laws, by means of disjunctive programs or the incorporation of probabilistic knowledge. A prototype that allows to compute extended justifications for logic programs can be tried on-line at `http://kr.irlab.org/cgraphs-solver/solver`.

Acknowledgements

We are thankful to Carlos Damásio for his suggestions and comments on earlier versions of this work. We also thank the anonymous reviewers for their help to improve the paper. This research was partially supported by Spanish Project TIN2013-42149-P.

Supplementary materials

For supplementary material for this article, please visit http://dx.doi.org/10.1017/S1471068416000107

References

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. Cambridge University Press, Cambridge, United Kingdom.

Cabalar, P., Fandinno, J. and Fink, M. 2014b. A complexity assessment for queries involving sufficient and necessary causes. In *Proc. Logics in Artificial Intelligence - 14th European*

Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24–26, 2014, E. Fermé and J. Leite, Eds. Lecture Notes in Computer Science, vol. 8761. Springer, Berlin, Germany, 297–310.

Damásio, C. V., Analyti, A. and Antoniou, G. 2013. Justifications for logic programming. In *Proc. Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15–19, 2013*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, Berlin, Germany, 530–542.

Denecker, M. and Schreye, D. D. 1993. Justification semantics: A unifiying framework for the semantics of logic programs. In *Proc. Logic Programming and Non-monotonic Reasoning, 2nd International Workshop, LPNMR 1993, Lisbon, Portugal, June 1993*. The MIT Press, Cambridge, Massachusetts, United States, 365–379.

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

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

Ferraris, P., Lee, J. and Lifschitz, V. 2007. A new perspective on stable models. In *Proc. of the 20th International Joint Conference on Artificial Intelligence, IJCAI 2007, Hyderabad, India, January 6–12, 2007*, M. M. Veloso, Ed. AAAI Press, Menlo Park, California, United States, 372–379.

Gebser, M., Pührer, J., Schaub, T. and Tompits, H. 2008. A meta-programming technique for debugging answer-set programs. In *Proc. of the 23rd AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13–17, 2008*, D. Fox and C. P. Gomes, Eds. AAAI Press, Menlo Park, California, United States, 448–453.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the 5th International Conference and Symposium, Seattle, Washington, August 15–19*, R. A. Kowalski and K. A. Bowen, Eds. MIT Press, Cambridge, Massachusetts, United States, 1070–1080.

Hall, N. 2000. Causation and the price of transitivity. *The Journal of Philosophy 97,* 4, 198–222. The Journal of Philosophy, Inc, New York, New York, United States.

Hall, N. 2004. Two concepts of causation. In *Causation and Counterfactuals*, J. Collins, N. Hall, and L. A. Paul, Eds. Cambridge, MA: MIT Press, Cambridge, Massachusetts, United States, 225–276.

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 11th International Conference, KR 2008, Sydney, Australia, September 16–19, 2008*, G. Brewka and J. Lang, Eds. AAAI Press, Menlo Park, California, United States, 198–208.

Halpern, J. Y. 2014. Appropriate causal models and stability of causation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 14th International Conference, KR 2014, Vienna, Austria, July 20–24, 2014*, C. Baral, G. D. Giacomo, and T. Eiter, Eds. AAAI Press, Menlo Park, California, United States.

Halpern, J. Y. 2015. A modification of the halpern-pearl definition of causality. In *Proc. of the 24th International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25–31, 2015*, Q. Yang and M. Wooldridge, Eds. AAAI Press, Menlo Park, California, United States, 3022–3033.

Halpern, J. Y. and Hitchcock, C. 2011. Actual causation and the art of modeling. *CoRR abs/1106.2652*, 2015.

Halpern, J. Y. and Pearl, J. 2001. Causes and explanations: A structural-model approach. Part I: Causes. In *Proc. of the 17th Conference in Uncertainty in Artificial Intelligence, UAI 2001, University of Washington, Seattle, Washington, USA, August 2–5*, Morgan Kaufmann, San Francisco, CA, 194–202.

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. Oxford University Press, Oxford, United Kingdom.

Hanks, S. and McDermott, D. V. 1987. Nonmonotonic logic and temporal projection. *Artificial Intelligence 33,* 3, 379–412. Philadelphia, Pennsylvania, United States.

Hitchcock, C. and Knobe, J. 2009. Cause and norm. *Journal of Philosophy 11*, 587–612. The Journal of Philosophy, Inc, New York, New York, United States.

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

Lewis, D. K. 2000. Causation as influence. *The Journal of Philosophy 97,* 4, 182–197. The Journal of Philosophy, Inc, New York, New York, United States.

Lin, F. 1995. Embracing causality in specifying the indirect effects of actions. In *Proc. of the 14th International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20–25 1995, 2 Volumes*. Morgan Kaufmann, San Francisco, CA, 1985–1993.

Marek, V. W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, K. R. Apt, V. W. Marek, M. Truszczyński, and D. Warren, Eds. Artificial Intelligence. Springer, Berlin, Germany, 375–398.

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, Cambridge, Massachusetts, United States, 419–443

McCain, N. and Turner, H. 1997. Causal theories of action and change. In *Proc. of the 14th National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27–31, 1997, Providence, Rhode Island*, B. Kuipers and B. L. Webber, Eds. AAAI Press/MIT Press, Menlo Park/Cambridge, California/Massachusetts, United States, 460–465.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273. Springer, Berlin, Germany.

Oetsch, J., Pührer, J. and Tompits, H. 2010. Catching the ouroboros: On debugging non-ground answer-set programs. In *Proc. CoRR abs/1007.4986*, 2010.

Pearce, D. 1996. A new logical characterisation of stable models and answer sets. In *Non-Monotonic Extensions of Logic Programming, NMELP 1996, Bad Honnef, Germany, September 5–6, 1996, Selected Papers*, J. Dix, L. M. Pereira, and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer, Berlin, Germany, 57–70.

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 *Proc. Functional and Logic Programming, 7th International Symposium, FLOPS 2004, Nara, Japan, April 7–9, 2004*, Y. Kameyama and P. J. Stuckey, Eds. Lecture Notes in Computer Science, vol. 2998. Springer, Berlin, Germany, 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. Cambridge University Press, Cambridge, United Kingdom.

Schulz, C. and Toni, F. 2016. Justifying answer sets using argumentation. In *Theory and Practice of Logic Programming TPLP 16,* 1, 59–110. Cambridge University Press, Cambridge, United Kingdom.

Specht, G. 1993. Generating explanation trees even for negations in deductive database systems. In *Proc. of the 15th 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.

Thielscher, M. 1997. Ramification and causality. *Artificial Intelligence 89,* 1–2, 317–364. Morgan Kaufmann, San Francisco, CA.

Van Gelder, A. 1989. The alternating fixpoint of logic programs with negation. In *Proc. of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29–31, 1989, Philadelphia, Pennsylvania, USA*, A. Silberschatz, Ed. ACM Press, Inc, New York, New York, United States, 1–10.

Van Gelder, A., Ross, K. A. and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM (JACM) 38,* 3, 620–650. ACM Press, Inc, New York, New York, United States.

https://doi.org/10.1017/S1471068416000107 Published online by Cambridge University Press