

An Algebra of Causal Chains^{*}

Pedro Cabalar and Jorge Fandinno

Department of Computer Science
University of Corunna, SPAIN
{cabalar, jorge.fandinno}@udc.es

Abstract. In this work we propose a multi-valued extension of logic programs under the stable models semantics where each true atom in a model is associated with a set of justifications, in a similar spirit than a set of proof trees. The main contribution of this paper is that we capture justifications into an algebra of truth values with three internal operations: an addition ‘+’ representing alternative justifications for a formula, a commutative product ‘*’ representing joint interaction of causes and a non-commutative product ‘.’ acting as a concatenation or proof constructor. Using this multi-valued semantics, we obtain a one-to-one correspondence between the syntactic proof tree of a standard (non-causal) logic program and the interpretation of each true atom in a model. Furthermore, thanks to this algebraic characterization we can detect semantic properties like redundancy and relevance of the obtained justifications. We also identify a lattice-based characterization of this algebra, defining a direct consequences operator, proving its continuity and that its least fix point can be computed after a finite number of iterations. Finally, we define the concept of *causal stable model* by introducing an analogous transformation to Gelfond and Lifschitz’s program reduct.

1 Introduction

A frequent informal way of explaining the effect of default negation in an introductory class on semantics in logic programming (LP) is that a literal of the form ‘*not p*’ should be read as “there is no way to derive *p*.” Although this idea seems quite intuitive, it is actually using a concept outside the discourse of any of the existing LP semantics: the *ways to derive p*. To explore this idea, [1] introduced the so-called *causal logic programs*. The semantics was an extension of stable models [2] relying on the idea of “justification” or “proof”. Any true atom, in a standard (non-causal) stable model needs to be justified. In a *causal stable model*, the truth value of each true atom captures these possible justifications, called *causes*. Let us see an example to illustrate this.

Example 1. Suppose we have a row boat with two rowers, one at each side of the boat, port and starboard. The boat moves forward *fw* if both rowers strike at a time. On the other hand, if we have a following wind, the boat moves forward anyway. \square

^{*} This research was partially supported by Spanish MEC project TIN2009-14562-C05-04 and Xunta program INCITE 2011.

Suppose now that we have indeed that both rowers stroke at a time when we additionally had a following wind. A possible encoding for this example could be the set of rules Π_1 :

$$\begin{aligned} p : port \quad s : starb \quad w : fwind \\ fwd \leftarrow port \wedge starb \quad fwd \leftarrow fwind \end{aligned}$$

In the only causal stable model of this program, atom fwd was justified by two alternative and independent causes. On the one hand, cause $\{p, s\}$ representing the joint interaction of $port$ and $starb$. On the other hand, cause $\{w\}$ inherited from $fwind$. We label rules (in the above program only atoms) that we want to be reflected in causes. Unlabelled fwd rules are just ignored when reflecting causal information. For instance, if we decide to keep track of the application of these rules, we would handle instead a program Π_2 obtained just by labelling these two rules in Π_1 as follows:

$$a : fwd \leftarrow port \wedge starb \tag{1}$$

$$b : fwd \leftarrow fwind \tag{2}$$

The two alternative justifications for atom fwd become the pair of causes $\{p, s\} \cdot a$ and $\{w\} \cdot b$. The informal reading of $\{p, s\} \cdot a$ is that “the joint interaction of $\{p\}$ and $\{s\}$, the cause $\{p, s\}$, is necessary to apply rule a .” From a graphical point of view, we can represent *causes* as proof trees.

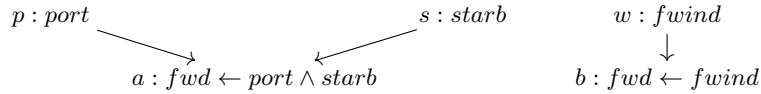


Fig. 1. Proof trees justifying atom fwd in the program Π_2

In this paper, we show that causes can be embedded in an algebra with three internal operations: an addition ‘+’ representing alternative justifications for a formula, a commutative product ‘*’ representing joint interaction of causes (in a similar spirit to the ‘+’ in [3]) and a non-commutative product ‘.’ acting as a concatenation or rule application. Using these operations, we can see that justification for fwd would correspond now to the value $((p * s) \cdot a) + (w \cdot b)$ which means that fwd is justified by the two alternative *causes*, $(p * s) \cdot a$ and $(w \cdot b)$. The former refers to the application of rule a to the join interaction of p and s . Similarly, the later refers to the application of rule b to w . From a graphical point of view, each cause corresponds to one of proof trees in the Figure 1, the right hand side operator of application corresponds to the head whereas the left hand side operator corresponds to the product of its children.

The rest of the paper is organised as follows. Section 2 describes the algebra with these three operations and a quite natural ordering relation on causes. The next section studies the semantics for positive logic programs and shows the correspondence between the syntactic proof tree of a standard (non-causal) logic program and the interpretation of each atom in a *causal model*. Section 4 introduces default negation and stable models. Finally, Section 5 concludes the paper.

2 Algebra of causal values

As we have introduced, our set of *causal values* will constitute an algebra with three internal operations: addition ‘+’ representing alternative causes, product ‘*’ representing joint interaction between causes and rule application ‘.’. We define now *causal terms*, the syntactic counterpart of (causal) values, just as combinations of these three operations over labels (events).

Definition 1 (Causal term). A causal term, t , over a set of labels Lb is recursively defined as one of the following expressions:

$$t ::= l \mid \prod_{t_i \in S} t_i \mid \sum_{t_i \in S} t_i \mid t_1 \cdot t_2$$

where l is a label $l \in Lb$, t_1, t_2 are in their turn causal terms and S is a (possibly empty or possibly infinite) set of causal terms. The set of causal terms over Lb is denoted by \mathcal{T}_{Lb} . \square

As we can see, infinite products and sums are allowed whereas a term may only contain a finite number of concatenation applications. Constants 0 and 1 will be shorthands for the empty sum $\sum_{t \in \emptyset} t$ and the empty product $\prod_{t \in \emptyset} t$, respectively.

We adopt the following notation. To avoid an excessive use of parentheses, we assume that ‘.’ has the highest priority, followed by ‘*’ and ‘+’ as usual, and we further note that the three operations will be associative. When clear from the context, we will sometimes remove ‘.’ so that, for instance, the term $l_1 l_2$ stands for $l_1 \cdot l_2$. As we will see, two (syntactically) different causal terms may correspond to the same causal value. However, we will impose Unique Names Assumption (UNA) for labels, that is, $l \neq l'$ for any two (syntactically) different labels $l, l' \in Lb$, and similarly $l \neq 0$ and $l \neq 1$ for any label l .

To fix properties of our algebra we recall that addition ‘+’ represents a set of alternative causes and product ‘*’ a set of causes that are jointly used. Thus, since both represent sets, they are associative, commutative and idempotent. Contrary, although associative, application ‘.’ is not commutative. Note that the right hand side operator represents the applied rule and left hand one represents a cause that is necessary to apply it, therefore they are clearly not interchangeable. We can note another interesting property: application ‘.’ distributes over both addition ‘+’ and product ‘*’. To illustrate this idea, consider the following variation of our example. Suppose now that the boat also leaves a wake behind when it moves forward. Let Π_3 be the set of rules Π_1 plus the rule $k : wake \leftarrow fwd$ reflecting this new assumption. As we saw, fwd is justified by $p * s + w$ and thus $wake$ will be justified by applying rule $k : wake \leftarrow fwd$ to it, i.e. the value $(p * s + w) \cdot k$. We can also see that there are two alternative causes justifying $wake$, graphically represented in the Figure 2. The term that corresponds which this graphical representation is $(p * s) \cdot k + w \cdot k = (p * s + w) \cdot k$. Moreover, application ‘.’ also distributes over product ‘*’ and $(p * s) \cdot k + w \cdot k$ is equivalent to $(p \cdot k) * (s \cdot k) + (w \cdot k)$. Intuitively, if the joint iteration of p and s is necessary to apply k then both p and s are also necessary to apply it, and conversely. Note that each chain of applications, $(p \cdot k)$, $(s \cdot k)$ and $(w \cdot k)$ corresponds to a path in one of the trees in the Figure 2. Causes can be seen as sets (products) of paths (causal chains).

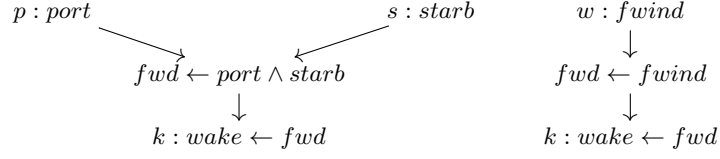


Fig. 2. Proof trees pontificating atom fwd in the program Π_3

Definition 2 (Causal Chain). A causal chain x over a set of labels Lb is a sequence $x = l_1 \cdot l_2 \cdot \dots \cdot l_n$, or simply $l_1 l_2 \dots l_n$, with length $|x| = n > 0$ and $l_i \in Lb$. \square

We denote \mathcal{X}_{Lb} to stand for the set of causal chains over Lb and will use letters x, y, z to denote elements from that set. It suffices to have a non-empty set of labels, say $Lb = \{a\}$, to get an infinite set of chains $\mathcal{X}_{Lb} = \{a, aa, aaa, \dots\}$, although all of them have a finite length. It is easy to see that, by an exhaustive application of distributivity, we can “shift” inside all occurrences of the application operator so that it only occurs in the scope of other application operators. A causal term obtained in this way is a *normal causal term*.

Definition 3 (Normal causal term). A causal term, t , over a set of labels Lb is recursively defined as one of the following expressions:

$$t ::= x \mid \prod_{t_i \in S} t_i \mid \sum_{t_i \in S} t_i$$

where $x \in \mathcal{X}_{Lb}$ is a causal chain over Lb and S is a (possibly empty or possibly infinite) set of normal causal terms. The set of causal terms over Lb is denoted by \mathcal{U}_{Lb} . \square

Proposition 1. Every causal term t can be normalized, i.e. written as an equivalent normal causal term u . \square

In the same way as application ‘ \cdot ’ distributes over addition ‘ $+$ ’ and product ‘ $*$ ’, the latter, in their turn, also distributes over addition ‘ $+$ ’. Consider a new variation of our example to illustrate this fact. Suppose that we have now two port rowers that can strike, encoded as the set of rules Π_4 :

$$\begin{array}{lll} p_1 : port_1 & p_2 : port_2 & s : starb \\ port \leftarrow port_1 & port \leftarrow port_2 & fwd \leftarrow port \wedge starb \end{array}$$

We can see that, in the only causal stable model of this program, atom $port$ was justified by two alternative, and independent causes, p_1 and p_2 , and after applying unlabelled rules to them, the resulting value assigned to fwd is $(p_1 + p_2) * s$. It is also clear that there are two alternative causes justifying fwd : the result from combining the starboard rower strike with each of the port rower strikes, $p_1 * s$ and $p_2 * s$. That is, causal terms $(p_1 + p_2) * s$ and $p_1 * s + p_2 * s$ are equivalent.

Furthermore, as we introduce above, causes can be ordered by a notion of “strength” of justification. For instance, in our example, fwd is justified by two independent

causes, $p * s + w$ while $fwind$ is only justified by w . If we consider the program Π_5 obtained by removing the fact $w : fwind$ from Π_1 then fwd keeps being justified by $p * s$ but $fwind$ becomes false. That is, fwd is “more strongly justified” than $fwind$ in Π_1 , written $w \leq p * s + w$. Similarly, $p * s \leq p * s + w$. Note also that, in this program Π_5 , fwd needs the joint interaction of p and s to be justified but $port$ and $starb$ only need p and s , respectively. That is, p is “more strongly justified” than $p * s$, written $p * s \leq p$. Similarly, $p * s \leq s$. We can also see that in program Π_2 which labels rules for fwd , one of the alternative causes for fwd is $w \cdot b$ and this is “less strongly justified” than w , i.e. $w \cdot b \leq w$ since, from a similar reasoning, $w \cdot b$ needs the application of b to w when w only requires itself. In general, we will see that $a \cdot b \leq a * b \leq X \leq a + b$ where X can be either a or b . We formalize this order relation starting for causal chains. Notice that a causal chain $x = l_1 l_2 \dots l_n$ can be alternatively characterized as a partial function from naturals to labels $x : \mathbb{N} \rightarrow Lb$ where $x(i) = l_i$ for all $i \leq n$ and undefined for $i > n$. Using this characterisation, we can define the following partial order among causal chains:

Definition 4 (Chain subsumption). *Given two causal chains x and $y \in \mathcal{X}_{Lb}$, we say that y subsumes x , written $x \leq y$, if and only if there exists a strictly increasing function $\delta : \mathbb{N} \rightarrow \mathbb{N}$ such that for each $i \in \mathbb{N}$ with $y(i)$ defined, $x(\delta(i)) = y(i)$.* \square

Proposition 2. *Given two finite causal chains $x, y \in \mathcal{X}_{Lb}$, they are equivalent (i.e. both $x \leq y$ and $y \leq x$) if and only if they are syntactically identical.* \square

Informally speaking, y subsumes x , when we can embed y into x , or alternatively when we can form y by removing (or skipping) some labels from x . For instance, take the causal chains $x = abcde$ and $y = ac$. Clearly we can form $y = ac = a \cdot \cancel{b} \cdot c \cdot \cancel{d} \cdot \cancel{e}$ by removing b , d and e from x . Formally, $x \leq y$ because we can take some strictly increasing function with $\delta(1) = 1$ and $\delta(2) = 3$ so that $y(1) = x(\delta(1)) = x(1) = a$ and $y(2) = x(\delta(2)) = x(3) = c$.

Although, at a first sight, it may seem counterintuitive the fact that $x \leq y$ implies $|x| \geq |y|$, as we mentioned, a fact or formula is “more strongly justified” when we need to apply less rules to derive it (and so, causal chains contain less labels) respecting their ordering. In this way, chain ac is a “more strongly justification” than $abcde$.

As we saw above, a cause can be seen as a product of causal chains, that from a graphical point of view correspond to the set of paths in a proof tree. We notice now an interesting property relating causes and the “more strongly justified” order relation: a joint interaction of comparable causal chains should collapse to the weakest among them. Take, for instance, a set of rules Π_6 :

$$a : p \quad b : q \leftarrow p \quad r \leftarrow p \wedge q$$

where, in the unique causal stable model, r corresponds to the value $a * a \cdot b$. Informally we can read this as “we need a and apply rule b to rule a to prove r ”. Clearly, we are repeating that we need a . Term a is redundant and then $a * a \cdot b$ is simply equivalent to $a \cdot b$. This idea is quite related to the definition of *order filter* in order theory. An *order filter* F of a poset P is a special subset $F \subseteq P$ satisfying¹ that for any $x \in F$ and $y \in P$, $x \leq y$

¹ *Order filter* is a weaker notion than *filter* which further satisfies that any pair $x, y \in F$ has a lower bound in F too.

implies $y \in F$. An order filter F is furthermore *generated* by an element $x \in P$ iff $x \leq y$ for all elements $y \in F$, the order filter generated by x is written $\|x\|$. Considering causes as the union of filters generated by their causal chains, the join interaction of causes just correspond to their union. For instance, if we consider the set of labels $Lb = \{a, b\}$ and its corresponding set of causal chains $\mathcal{X}_{Lb} = \{a, b, ab, ba, \dots\}$, then $\|ab\|$ and $\|a\|$ respectively correspond to the set of all chains grater than ab and a in the poset $P = \langle \mathcal{X}_{Lb}, \leq \rangle$. Those are, $\|ab\| = \{ab, a, b\}$ and $\|a\| = \{a\}$. The term $a * ab$ corresponds just to the union of both sets $\|a\| \cup \|ab\| = \|ab\|$. We define a cause as follows:

Definition 5 (Cause). A cause for a set of labels Lb is any order filter for the poset of chains $\langle \mathcal{X}_{Lb}, \leq \rangle$. We will write \mathcal{C}_{Lb} (or simply \mathcal{C} when there is no ambiguity) to denote the set of all causes for Lb . \square

This definition captures the notion of cause, or syntactically a product of causal chains. To capture possible alternative causes, that is, additions of products of causal chains, we notice that addition obeys a similar behaviour with respect to redundant causes. Take, for instance, a set of rules Π_7 :

$$a : p \quad b : p \leftarrow p$$

It is clear, that the cause a is sufficient to justify p , but there are also infinitely many other alternative and redundant causes $a \cdot b, a \cdot b \cdot b, \dots$ that justify p , that is $a + a \cdot b + a \cdot b \cdot b + \dots$. To capture a set of alternative causes we define the idea of causal value, in its turn, as a filter of causes.

Definition 6 (Causal Value). Given a set of labels Lb , a causal value is any order filter for the poset $\langle \mathcal{C}_{Lb}, \subseteq \rangle$. \square

The causal value $\|a\|$, the filter generated by the cause $\|a\|$, is the set containing $\|a\| = \{a, a + b\}$ and all its supersets. That is, $\|a\| = \{\|a\|, \|a * b\|, \|a \cdot b\|, \dots\}$. Furthermore, as we will see later, addition can be interpreted as the union of causal values for its respective operands. Thus, $a + a \cdot b + a \cdot b \cdot b + \dots$ just corresponds to the union of the causal values generated by their addend causes, $\|a\| \cup \|a \cdot b\| \cup \|a \cdot b \cdot b\| + \dots = \|a\|$.

The set of possible causal values formed with labels Lb is denoted as \mathcal{V}_{Lb} . An element from \mathcal{V}_{Lb} has the form of a set of sets of causal chains that, intuitively, corresponds to a set of alternative causes (*sum of products of chains*). From a graphical point of view, it corresponds to a set of alternative proof trees represented as their respective sets of paths. We define now the correspondence between syntactical causal terms and their semantic counterpart, causal values.

Definition 7 (Valuation of normal terms). The valuation of a normal term is a mapping $\epsilon : \mathcal{U}_{Lb} \longrightarrow \mathcal{V}_{Lb}$ defined as:

$$\epsilon(x) \stackrel{\text{def}}{=} \|x\| \text{ with } x \in \mathcal{X}_{Lb}, \quad \epsilon\left(\sum_{t \in S} t\right) \stackrel{\text{def}}{=} \bigcup_{t \in S} \epsilon(t), \quad \epsilon\left(\prod_{t \in S} t\right) \stackrel{\text{def}}{=} \bigcap_{t \in S} \epsilon(t) \quad \square$$

Note that any causal term can be normalized and then this definition trivially extends to any causal term. Furthermore, a causal chain x is mapped just to the causal value generated by the cause, in their turn, generated by x , i.e. the set containing all causes which contain x . The aggregate union of an empty set of sets (causal values) corresponds to \emptyset . Therefore $\epsilon(0) = \bigcup_{t \in \emptyset} \epsilon(t) = \emptyset$, i.e. 0 just corresponds to the absence of justification. Similarly, as causal values range over parts of \mathcal{C} , the aggregate intersection of an empty set of causal values corresponds to \mathcal{C} , and thus $\epsilon(1) = \bigcap_{t \in \emptyset} \epsilon(t) = \mathcal{C}$, i.e. 1 just corresponds to the “maximal” justification.

Theorem 1 (From [4]). $\langle \mathcal{V}_{Lb}, \cup, \cap \rangle$ is the free completely distributive lattice generated by $\langle \mathcal{X}_{Lb}, \leq \rangle$, and the restriction of ϵ to \mathcal{X}_{Lb} is an injective homomorphism (or embedding). \square

The above theorem means that causal terms form a complete lattice. The order relation \leq between causal terms just corresponds to set inclusion between their corresponding causal values, i.e. $x \leq y$ iff $\epsilon(x) \subseteq \epsilon(y)$. Furthermore, addition ‘+’ and product ‘*’ just respectively correspond to the least upper bound and the greater lower bound of the associated lattice $\langle \mathcal{T}_{Lb}, \leq \rangle$ or $\langle \mathcal{T}_{Lb}, +, * \rangle$ where:

$$t \leq u \stackrel{\text{def}}{=} \epsilon(t) \subseteq \epsilon(u) \quad (\Leftrightarrow \quad t * u = t \quad \Leftrightarrow \quad t + u = u)$$

for any normal term t and u . For instance, in our example Π_2 , fw was associated with the causal term $p \cdot a * s \cdot a + w \cdot b$. Thus, the causal value associated with it corresponds to

$$\epsilon(p \cdot a * s \cdot a + w \cdot b) = \llbracket p \cdot a \rrbracket \cap \llbracket s \cdot a \rrbracket \cup \llbracket w \cdot b \rrbracket$$

Causal values are, in general, infinite sets. For instance, as we saw before, simply with $Lb = \{a\}$ we have the chains $\mathcal{X}_{Lb} = \{a, aa, aaa, \dots\}$ and $\epsilon(a)$ contains all possible causes in \mathcal{C} that are supersets of $\{a\}$, that is, $\epsilon(a) = \{\{a\}, \{aa, a\}, \{aaa, aa, a\}, \dots\}$. Obviously, writing causal values in this way is infeasible – it is more convenient to use a representative causal term instead. For this purpose, we define a function γ that acts as a right inverse morphism for ϵ selecting minimal causes, i.e., given a causal value V , it defines a normal term $\gamma(V) = t$ such that $\epsilon(t) = V$ and $\gamma(V)$ does not have redundant subterms. The function γ is defined as a mapping $\gamma : \mathcal{V}_{Lb} \rightarrow \mathcal{U}_{Lb}$ such that for any causal value $V \in \mathcal{V}_{Lb}$, $\gamma(V) \stackrel{\text{def}}{=} \sum_{C \in \underline{V}} \prod_{x \in \underline{C}} x$ where $\underline{V} = \{C \in V \mid \nexists D \in V, D \subset C\}$ and $\underline{C} = \{x \in C \mid \nexists y \in C, y < x\}$ respectively stand for \subseteq -minimal causes of V and \leq -minimal chains of C . We will use $\gamma(V)$ to represent V .

Proposition 3. The mapping γ is a right inverse morphism of ϵ . \square

Given a term t we define its *canonical form* as $\gamma(\epsilon(t))$. Canonical terms are of the form of sums of products of causal chains. As it can be imagined, not any term in that form is a canonical term. For instance, going back, we easily can check that terms $a * ab = ab$ and $a + ab + abb + \dots = a$ respectively correspond to the canonical terms $\gamma(\epsilon(ab * a)) = \gamma(\epsilon(ab)) = ab$ and $\gamma(\epsilon(a + ab + abb + \dots)) = \gamma(\epsilon(a)) = a$. Figure 3 summarizes addition and product properties while Figure 4 is analogous for application properties.

<i>Associativity</i>	<i>Commutativity</i>	<i>Absorption</i>
$t + (u+w) = (t+u) + w$	$t + u = u + t$	$t = t + (t*u)$
$t * (u*w) = (t*u) * w$	$t * u = u * t$	$t = t * (t+u)$

<i>Distributive</i>	<i>Identity</i>	<i>Idempotence</i>	<i>Annihilator</i>
$t + (u*w) = (t+u) * (t+w)$	$t = t + 0$	$t = t + t$	$1 = 1 + t$
$t * (u+w) = (t*u) + (t*w)$	$t = t * 1$	$t = t * t$	$0 = 0 * t$

Fig. 3. Sum and product satisfy the properties of a completely distributive lattice.

<i>Associativity</i>	<i>Absorption</i>	<i>Identity</i>
$t \cdot (u \cdot w) = (t \cdot u) \cdot w$	$t = t + u \cdot t \cdot w$	$t = 1 \cdot t$
	$u \cdot t \cdot w = t * u \cdot t \cdot w$	$t = t \cdot 1$

<i>Addition distributivity</i>	<i>Product distributivity</i>	<i>Annihilator</i>
$t \cdot (u+w) = (t \cdot u) + (t \cdot w)$	$c \cdot (d * e) = (c \cdot d) * (c \cdot e)$	$0 = t \cdot 0$
$(t+u) \cdot w = (t \cdot w) + (u \cdot w)$	$(c * d) \cdot e = (c \cdot e) * (c \cdot e)$	$0 = 0 \cdot t$

Fig. 4. Properties of the application ‘ \cdot ’ operator. Note: c , d and e denote a causes instead of arbitrary causal terms.

For practical purposes, simplification of causal terms can be done by applying the algebraic properties shown in Figures 3 and 4. For instance, the examples from Π_6 and Π_7 containing redundant information can now be derived as follows:

$$\begin{aligned}
a * a \cdot b &= (a * 1 \cdot a \cdot b) && \text{identity for ‘}\cdot\text{’} \\
&= 1 \cdot a \cdot b && \text{absorption for ‘}\cdot\text{’} \\
&= a \cdot b && \text{identity for ‘}\cdot\text{’}
\end{aligned}$$

$$\begin{aligned}
a + a \cdot b + a \cdot b \cdot b + \dots &= a + 1 \cdot a \cdot b + a \cdot b \cdot b + \dots && \text{identity for ‘}\cdot\text{’} \\
&= a + a \cdot b \cdot b + \dots && \text{absorption for ‘}\cdot\text{’} \\
&= a + 1 \cdot a \cdot b \cdot b + \dots && \text{identity for ‘}\cdot\text{’} \\
&\dots && \dots \\
&= a && \text{absorption for ‘}\cdot\text{’}
\end{aligned}$$

Let us see another example involving distributivity. The term $ab * c + a$ can be derived as follows:

$$\begin{aligned}
a \cdot b * c + a &= (a \cdot b + a) * (c + a) && \text{distributivity} \\
&= (1 \cdot a \cdot b + a) * (c + a) && \text{identity for ‘}\cdot\text{’} \\
&= (a + 1 \cdot a \cdot b) * (c + a) && \text{commutativity for ‘}+\text{’} \\
&= a * (c + a) && \text{absorption for ‘}\cdot\text{’} \\
&= a && \text{absorption for ‘}\cdot\text{’}
\end{aligned}$$

3 Positive programs and minimal models

Let us describe now how to use the causal algebra to evaluate causal logic programs. A *signature* is a pair $\langle At, Lb \rangle$ of sets that respectively represent the set of *atoms* (or *propositions*) and the set of labels. As usual, a *literal* is defined as an atom p (positive literal) or its negation $\neg p$ (negative literal). In this paper, we will concentrate on programs without disjunction in the head, leaving the treatment of disjunction for a future study.

Definition 8 (Causal logic program). *Given a signature $\langle At, Lb \rangle$ a (causal) logic program Π is a set of rules of the form:*

$$t : L_0 \leftarrow L_1 \wedge \dots \wedge L_m \wedge \text{not } L_{m+1} \wedge \dots \wedge \text{not } L_n$$

where t is a causal term over Lb , L_0 is a literal or \perp (the head of the rule) and $L_1 \wedge \dots \wedge L_m \wedge \text{not } L_{m+1} \wedge \dots \wedge \text{not } L_n$ is a conjunction of literals (the body of the rule). An empty body is represented as \top . \square

For any rule ϕ of the form $t : L_0 \leftarrow L_1 \wedge \dots \wedge L_m \wedge \text{not } L_{m+1} \wedge \dots \wedge \text{not } L_n$ we define $\text{label}(\phi) = t$. Most of the following definitions are standard in logic programming. We denote $\text{head}(\phi) = L_0$, B^+ (resp. B^-) to represent the conjunction of all positive (resp. negative) literals $L_1 \wedge \dots \wedge L_n$ (resp. $\text{not } L_{m+1} \wedge \dots \wedge \text{not } L_n$) that occur in B . A logic program is *positive* if B^- is empty for all rules ($n = m$), that is, if it contains no negations. Unlabelled rules are assumed to be labelled with the element 1 which, as we saw, is the identity for application ‘ \cdot ’. \top (resp. \perp) represent truth (resp. falsity). If $n = m = 0$ then \leftarrow can be dropped.

Given a signature $\langle At, Lb \rangle$ a *causal interpretation* is a mapping $I : At \longrightarrow \mathcal{V}_{Lb}$ assigning a causal value to each atom. Partial order \leq is extended over interpretations so that given two interpretations I, J we define $I \leq J \stackrel{\text{def}}{=} I(p) \leq J(p)$ for each atom $p \in At$. There is a \leq -bottom interpretation $\mathbf{0}$ (resp. a \leq -top interpretation $\mathbf{1}$) that stands for the interpretation mapping each atom p to 0 (resp. 1). The set of interpretations \mathcal{I} with the partial order \leq forms a poset $\langle \mathcal{I}, \leq \rangle$ with supremum ‘ $+$ ’ and infimum ‘ $*$ ’ that are respectively the sum and product of atom interpretations. As a result, $\langle \mathcal{I}, +, * \rangle$ also forms a complete lattice.

Observation 1 *When $Lb = \emptyset$ the set of causal values becomes $\mathcal{V}_{Lb} = \{0, 1\}$ and interpretations collapse to classical propositional logic interpretations.* \square

Definition 9 (Causal model). *Given a positive causal logic program Π and a causal interpretation I over the signature $\langle At, Lb \rangle$, I is a causal model, written $I \models \Pi$, if and only if*

$$(I(L_1) * \dots * I(L_m)) \cdot t \leq I(L_0)$$

for each rule $\phi \in \Pi$ of the form $\phi = L_0 \leftarrow L_1, \dots, L_m$.

For instance, take rule (1) from Example 1 and let I be an interpretation such that $I(\text{port}) = p$ and $I(\text{starb}) = s$. Then I will be a model of (1) when $(p * s) \cdot a \leq$

$I(fwd)$. In particular, this holds when $I(fwd) = (p * s) \cdot a + w \cdot b$ which was the value we expected for program Π_2 . But it would also hold when, for instance, $I(fwd) = a + b$ or $I(fwd) = 1$. Note that this is important if we had to accommodate other possible additional facts $(a : fwd)$ or even $(1 : fwd)$ in the program. The fact that any $I(fwd)$ greater than $(p * s) \cdot a + w \cdot b$ is also a model clearly points out the need for selecting minimal models. In fact, as happens in the case of non-causal programs, positive programs have a least model (this time, with respect to \leq relation among causal interpretations) that can be computed by iterating an extension of the well-known *direct consequences operator* defined by [5].

Definition 10 (Direct consequences). *Given a positive logic program Π for signature $\langle At, Lb \rangle$ and a causal interpretation I , the operator of direct consequences is a function $T_\Pi : \mathcal{I} \rightarrow \mathcal{I}$ such that, for any atom $p \in At$:*

$$T_\Pi(I)(L_0) \stackrel{\text{def}}{=} \sum \{ (I(L_1) * \dots * I(L_m)) \cdot t \mid (t : L_0 \leftarrow L_1 \wedge \dots \wedge L_m) \in \Pi \}$$

In order to prove some properties of this operator, an important observation should be made: since the set of causal values forms now a lattice, causal logic programs can be translated to *Generalized Annotated Logic Programming* (GAP). GAP is a general framework for multivalued logic programming where the set of truth values must to form an upper semilattice and rules (*annotated clauses*) have the following form:

$$L_0 : \rho \leftarrow L_1 : \mu_1 \ \& \ \dots \ \& \ L_m : \mu_m \quad (3)$$

where L_0, \dots, L_m are literals, ρ is an *annotation* (may be just a truth value, an *annotation variable* or a *complex annotation*) and μ_1, \dots, μ_m are values or annotation variables. A complex annotation is the result to apply a total continuous function to a tuple of annotations. For instance ρ can be a complex annotation $f(\mu_1, \dots, \mu_m)$ that applies the function f to a m-tuple (μ_1, \dots, μ_m) of annotation variables in the body of (3). Given a positive program Π , each rule $\varphi \in \Pi$ of the form

$$t : L_0 \leftarrow L_1 \wedge \dots \wedge L_m \quad (4)$$

is translated to an annotated clause $GAP(\varphi)$ of the form of (3) where μ_1, \dots, μ_m are annotation variables that capture the causal values of each body literal. The head complex annotation corresponds to the function $\rho \stackrel{\text{def}}{=} (\mu_1 * \dots * \mu_m) \cdot t$ that forces the head to inherit the causal value obtained by applying the rule label t to the product of the interpretation of body literals $\mu_1 * \dots * \mu_m$. The translation of a program Π is simply defined as:

$$GAP(\Pi) \stackrel{\text{def}}{=} \{GAP(\varphi) \mid \varphi \in \Pi\}$$

A complete description of GAP restricted semantics, denoted as \models^r , is out of the scope of this paper (the reader is referred to [6]). For our purposes, it suffices to observe that the following important property is satisfied.

Theorem 2. *A positive causal logic program Π can be translated to a general annotated logic program $GAP(\Pi)$ s.t. a causal interpretation $I \models \Pi$ if and only if $I \models^r GAP(\Pi)$. Furthermore, $T_\Pi(I) = R_{GAP(\Pi)}(I)$ for any interpretation I .*

Corollary 1. *Given a positive logic program Π the following properties hold:*

1. *Operator T_Π is monotonic.*
2. *Operator T_Π is continuous.*
3. *$T_\Pi \uparrow^\omega (\mathbf{0}) = \text{lp}(T_\Pi)$ is the least model of Π .*
4. *The iterative computation $T_\Pi \uparrow^k (\mathbf{0})$ reaches the least fixpoint in n steps for some positive integer n .*

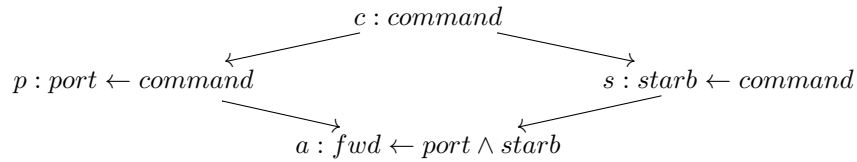
Proof. Directly follows from Theorem 2 and Theorems 1, 2 and 3 in [6].

The existence of a least model for a positive program and its computation using T_Π is an interesting result, but it does not provide any information on the relation between the causal value it assigns to each atom with respect to its role in the program. As we will see, we can establish a direct relation between this causal value and the idea of *proof* in the positive program. Let us formalise next the idea of proof tree.

Definition 11 (Proof tree). *Given a causal logic program Π , a proof tree is a directed acyclic graph $T = \langle V, E \rangle$, where vertices $V \subseteq \Pi$ are rules from the program, and $E \subseteq V \times V$ satisfying:*

- (i) *There is at exactly one vertex without outgoing edges denoted as $\text{sink}(T)$.*
- (ii) *For each rule $\varphi = (t : L_0 \leftarrow B) \in V$ and for each atom $L_i \in B^+$ there is exactly one φ' with $(\varphi', \varphi) \in E$ and this rule satisfies $\text{head}(\varphi') = L_i$.* \square

Notice that condition (ii) forces us to include an incoming edge for each atom in the positive body of a vertex rule. As a result, source vertices must be rules with empty positive body, or just facts in the case of positive programs. Another interesting observation is that, proof *tree* do not require an unique parent for each vertex. For instance, in Example 1, if both *port* and *starb* were obtained as a consequence of some command made by the captain, we could get instead a proof tree, call it T_1 , of the form:



Definition 12 (Proof path). *Given a proof tree $T = \langle V, E \rangle$ we define a proof path for T as a concatenation of terms $t_1 \dots t_n$ satisfying:*

1. *There exists a rule $\varphi \in V$ with $\text{label}(r) = t_1$ such that φ is a source, that is, there is no φ' s.t. $(\varphi', \varphi) \in E$.*
2. *For each pair of consecutive terms t_i, t_{i+1} in the sequence, there is some edge $(\varphi_i, \varphi_{i+1}) \in E$ s.t. $\text{label}(\varphi_i) = t_i$ and $\text{label}(\varphi_{i+1}) = t_{i+1}$.*
3. *$\text{label}(\text{sink}(T)) = t_n$.* \square

Let us write $\text{Paths}(T)$ to stand for the set of all proof paths for a given proof tree T . We define the cause associated to any tree $T = \langle V, E \rangle$ as the causal term $\text{cause}(T) \stackrel{\text{def}}{=} \prod_{t \in \text{Paths}(T)} t$. As an example, $\text{cause}(T_1) = (c \cdot p \cdot a) * (c \cdot s \cdot a)$. Also $(p \cdot a) * (s \cdot a)$ and $w \cdot b$ correspond to each tree in Figure 1.

Theorem 3. *Let Π be a positive program and I be the least model of Π , then for each atom p :*

$$I(p) = \sum_{T \in PT_p} \text{cause}(T)$$

where $PT_p = \{T = \langle V, E \rangle \mid \text{head}(\text{sink}(T)) = p\}$ is a set of proof trees with nodes $V \subseteq \Pi$.

From this result, it may seem that our semantics is just a direct translation of the syntactic idea of proof trees. However, the semantics is actually a more powerful notion that allows detecting redundancies, tautologies and inconsistencies. In fact, the expression $\sum_{T \in PT_p} \text{cause}(T)$ may contain redundancies and is not, in the general case, in normal form. As an example, recall the program Π_6 :

$$a : p \quad b : q \leftarrow p \quad r \leftarrow p \wedge q$$

that has only one proof tree for p whose cause would correspond to $I(r) = a * a \cdot b$. But, by absorption, this is equivalent to $I(r) = a \cdot b$ pointing out that the presence of p in rule $r \leftarrow p \wedge q$ is redundant.

A corollary of Theorem 3 is that we can replace a rule label by a different one, or by 1 (the identity for application ‘ \cdot ’) and we get the same least model, modulo the same replacement in the causal values for all atoms.

Corollary 2. *Let Π be a positive program, I the least model of Π , $l \in Lb$ be a label, $m \in Lb \cup \{1\}$ and Π_m^l (resp. I_m^l) be the program (resp. interpretation) obtained after replacing each occurrence of l by m in Π (resp. in the interpretation of each atom in I). Then I_m^l is the least model of Π_m^l . \square*

In particular, replacing a label by $m = 1$ has the effect of removing it from the signature. Suppose we make this replacement for all atoms in Lb and call the resulting program and least model Π_1^{Lb} and I_1^{Lb} respectively. Then Π_1^{Lb} is just the non-causal program resulting from Π after removing all labels and it is easy to see (Observation 1) that I_1^{Lb} coincides with the least classical model of this program². Moreover, this means that for any positive program Π , if I is its least model, then the classical interpretation:

$$I'(p) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } I(p) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

is the least classical model of Π ignoring its labels.

4 Default negation and stable models

Consider now the addition of negation, so that we deal with arbitrary programs. To illustrate this, we introduce a variation in Example 1 introducing the *qualification problem* from [7]: actions for moving the boat forward can be disqualified if an *abnormal*

² Note that I^{Lb} is Boolean: it assigns either 0 or 1 to any atom in the signature.

situation occurs (for instance, that the boat is anchored, any of the oars are broken, the sail is full of holes, etc.) . As usual this can be represented using default negation as shown in the set of rules Π_8 :

$$\begin{array}{lll}
p : port & s : starb & w : fwind \\
a : fwd \leftarrow port \wedge starb \wedge not\ ab_a & b : fwd \leftarrow fwind \wedge not\ ab_b & \\
ab_a \leftarrow anchored & ab_b \leftarrow anchored & \\
ab_a \leftarrow broken_oar1 & ab_b \leftarrow holed_sail & \\
ab_a \leftarrow broken_oar2 & \dots &
\end{array}$$

The causes that justify an atom should not be a list of not occurred abnormal situations. For instance, in program Π_8 where no abnormal situation occurs, the causal value that justify atom fwd should be $(p \cdot a * s \cdot a) + (w \cdot b)$ as in the program Π_2 where abnormal situations are not included. References to the not occurred abnormal situations (*not anchored*, *not broken_oar1*...) are not mentioned. Default negation does not affect the causes justifying an atom when the default holds. Of course, when the default does not hold, for instance adding the fact *anchored* to the above program, fwd becomes false. Thus, we introduce the following straightforward rephrasing of the traditional program reduct [2].

Definition 13 (Program reduct). *The reduct of a program Π with respect to an interpretation I , written Π^I is the result of the following transformations on Π :*

1. *Removing all rules s.t. $I(B^-) = 0$*
2. *Removing all negative literals from the rest of rules.* □

A causal interpretation I is a *causal stable model* of a causal program Π if I is the least model of Π^I . This definition allows us to extend Theorem 3 to normal programs in a direct way:

Theorem 4 (Main theorem). *Let Π be a causal program and I be causal stable model of Π , then for each atom p :*

$$I(p) = \sum_{T \in PT_p} cause(T) \quad \text{where}$$

$$PT_p = \{T = \langle V, E \rangle \mid head(sink(T)) = p \text{ and } V \subseteq \{(t : q \leftarrow B) \in \Pi \mid I(B^-) \neq 0\}\}. \quad \square$$

That is, the only difference now is that the set of proof trees PT_p is formed with rules whose negative body is not false $I(B^-) \neq 0$ (that is, they would generate rules in the reduct).

Corollary 3. *Let Π be a normal program, I a causal stable model of Π , $l \in Lb$ be a label, $m \in Lb \cup \{1\}$ and Π_m^l (resp. I_m^l) be the program (resp. interpretation) obtained after replacing every occurrence of l by m in Π (resp. in the interpretation of each atom in I). Then I_m^l is a causal stable model of Π_m^l .* □

As in the case of positive programs, replacing a label by $m = 1$ has the effect of removing it from the signature. Then, for any normal program Π , if I is a causal stable model, then the classical interpretation:

$$I'(p) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } I(p) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

is a classical stable model of Π ignoring its labels. It is easy to see that not only the above program Π_8 has an unique causal stable model that corresponds to:

$$\begin{array}{ll} I(\text{port}) &= p & I(\text{ab_f}) &= 0 \\ I(\text{starb}) &= s & I(\text{anchored}) &= 0 \\ I(\text{fwind}) &= w & I(\text{broken_oar1}) &= 0 \\ I(\text{fwd}) &= (p \cdot a * s \cdot a) + (w \cdot b) & \dots &= 0 \end{array}$$

but also the program obtained from it ignoring the labels has an unique standard stable model $\{\text{port}, \text{starb}, \text{fwind}, \text{fwd}\}$ that corresponds to the atoms whose interpretations differ from 0 in the former.

5 Conclusions

In this paper we have provided a multi-valued semantics for normal logic programs whose truth values form a lattice of causal chains. A causal chain is nothing else but a concatenation of rule labels that reflects some sequence of rule applications. In this way, a model assigns to each true atom a value that contains justifications for its derivation from existing rules. We have further provided three basic operations on the lattice: an addition, that stands for alternative, independent justifications; a product, that represents joint interaction of causes; and a concatenation that acts as a chain constructor. We have shown that this lattice is completely distributive and provided a detailed description of the algebraic properties of its three operations.

A first important result is that, for positive programs, there exists a least model that coincides with the least fixpoint of a direct consequences operator, analogous to [5]. With this, we are able to prove a direct correspondence between the semantic values we obtain and the syntactic idea of proof tree. The main result of the paper generalises this correspondence for the case of stable models for normal programs.

Many open topics remain for future study. For instance, ongoing work is currently focused on implementation, complexity assessment, extension to disjunctive programs or introduction of strong negation. Regarding expressivity, an interesting topic is the introduction of new syntactic operators for inspecting causal information like checking the influence of a particular event or label in a conclusion, expressing necessary or sufficient causes, or even dealing with counterfactuals. Another interesting topic is removing the syntactic reduct definition in favour of some full logical treatment of default negation, as happens for (non-causal) stable models and their characterisation in terms of Equilibrium Logic [8]. This would surely simplify the quest for a necessary and sufficient condition for strong equivalence, following similar steps to [9]. It may also allow extending the definition of causal stable models to an arbitrary syntax and to the

first order case, where the use of variables in labels may also introduce new interesting features.

There are also other areas whose relations deserve to be formally studied. For instance, the introduction of a strong negation operator will immediately lead to a connection to Paraconsistency approaches. In particular, one of the main problems in the area of Paraconsistency is deciding which parts of the theory do not propagate or *depend* on an inconsistency. This decision, we hope, will be easier in the presence of causal justifications for each derived conclusion. A related area for which similar connections can be exploited is Belief Revision. In this case, causal information can help to decide which *relevant* part of a revised theory must be withdrawn in the presence of new information that would lead to an inconsistency if no changes are made. A third obvious related area is Debugging in Answer Set Programming, where we try to explain discrepancies between an expected result and the obtained stable models. In this field, there exists a pair of relevant approaches [10, 11] to whom we plan to compare. Finally, as potential applications, our main concern is designing a high level action language on top of causal logic programs with the purpose of modelling some typical scenarios from the literature on causality in Artificial Intelligence.

References

1. Cabalar, P.: Causal logic programming. In Erdem, E., Lee, J., Lierler, Y., Pearce, D., eds.: Correct Reasoning. Volume 7265 of Lecture Notes in Computer Science., Springer (2012) 102–116
2. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R.A., Bowen, K.A., eds.: Logic Programming: Proc. of the Fifth International Conference and Symposium (Volume 2). MIT Press, Cambridge, MA (1988) 1070–1080
3. Artëmov, S.N.: Explicit provability and constructive semantics. *Bulletin of Symbolic Logic* **7**(1) (2001) 1–36
4. Stumme, G.: Free distributive completions of partial complete lattices. *Order* **14** (1997) 179–189
5. van Emden, M.H., Kowalski, R.A.: The semantics of predicate logic as a programming language. *J. ACM* **23**(4) (1976) 733–742
6. Kifer, M., Subrahmanian, V.S.: Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming* **12** (1992)
7. McCarthy, J.: Epistemological problems of artificial intelligence. In Reddy, R., ed.: *IJCAI*, William Kaufmann (1977) 1038–1044
8. Pearce, D.: Equilibrium logic. *Ann. Math. Artif. Intell.* **47**(1-2) (2006) 3–41
9. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. *ACM Trans. Comput. Log.* **2**(4) (2001) 526–541
10. Gebser, M., Pührer, J., Schaub, T., Tompits, H.: A meta-programming technique for debugging answer-set programs. In Fox, D., Gomes, C.P., eds.: *AAAI*, AAAI Press (2008) 448–453
11. Pontelli, E., Son, T.C., El-Khatib, O.: Justifications for logic programs under answer set semantics. *TPLP* **9**(1) (2009) 1–56