An Algebra of Causal Chains*

Pedro Cabalar and Jorge Fandinno

Department of Computer Science
University of Corunna, SPAIN
{cabalar, jorge.fandino}@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 '$\cdot$' 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 fwd if both rowers strike at a time. On the other hand, if we have a following wind, the boat moves forward anyway. □

* 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:

$$p : port \quad s : starb \quad w : fwind$$
$$fwd \leftarrow port \wedge starb \qquad fwd \leftarrow fwind$$

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 used to apply rule a." From a graphical point of view, we can represent *causes* as proof trees.

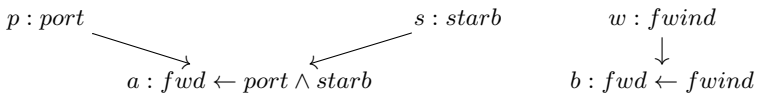


Fig. 1. Proof trees justifying atom fwd in the program Π_2

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 '$\cdot$' acting as a concatenation or rule application. Using these operations, we can see that justification for fwd would correspond now to the value $((p * s) \cdot a) + (w \cdot b)$. Addition '$+$' represents alternative justification, and each addend, $(p * s) \cdot a$ and $(w \cdot b)$, corresponds to a *cause* (alternative justification) for fwd. Product '$*$' and application '$\cdot$' work together to construct a cause. Right hand operator of application '$\cdot$' corresponds to the applied rule, for instance rule a, while the left hand operator corresponds to a product '$*$' of causes used to apply it, $p * s$. From a graphical point of view, each cause corresponds to one of proof trees in the Figure 1, the right hand operator of application corresponds to the head of a proof three, and the left hand 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 '$\cdot$'. We define now "causal terms" the syntactic counterpart of "causal values" just as combination 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 $\mathcal{T}_{Lb}$. □

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 '$\cdot$' 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 '$\cdot$' 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 notice 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 '$\cdot$' is not commutative. Note that, right hand operator represents the applied rule while left hand one represents the cause used to apply it, and then they are clearly not interchangeable. We can note another interesting property: application '$\cdot$' 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

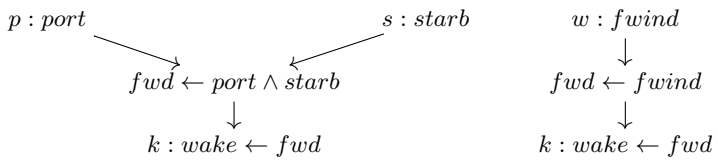


Fig. 2. Proof trees pontificating atom fwd in the program Π_3

is $(p * s) \cdot k + w \cdot k = (p * s + w) \cdot k$. Moreover, application '$\cdot$' also distributes over product '$*$' and $(p * s) \cdot k + w \cdot k$ is equivalent to $(p \cdot k) * (s \cdot k) + (w \cdot k)$. 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).

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

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 with having a non-empty set of labels, say $Lb = \{a\}$, to get an infinite set of chains $\mathcal{X}_{Lb} = \{a, aa, aaa, \ldots\}$, 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 $\mathcal{U}_{Lb}$. □

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

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:

$$p_1 : port_1 \quad p_2 : port_2 \quad s : starb$$
$$port \leftarrow port_1 \qquad port \leftarrow port_2 \qquad fwd \leftarrow port \wedge starb$$

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 rowers 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 continue 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 needs 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, $a \cdot b \leq a * b \leq (a, b) \leq a + b$. 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 $x : \mathbb{N} \longrightarrow Lb$ 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 $x \leq y$, iff there exists a strictly increasing function $\delta : \mathbb{N} \longrightarrow \mathbb{N}$ such that for each $i \in \mathbb{N}$ with $y(i)$ defined, $x\big(\delta(i)\big) = y(i)$.* □

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

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 $x = abcde$ and $y = ac$. Clearly we can form $y = ac = a \cdot \not{b} \cdot c \cdot \not{d} \cdot \not{e}$ 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 stronger 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:

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

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 $F \subseteq P$ satisfying[1] that for any $x \in F$ and $y \in P$, $x \leq y$ implies $y \in F$. An order filter F is furthermore *generated* by an element $x \in P$ iff $x \leq y$ for all element $y \in F$, 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, $||a \cdot b||$ is the set of all terms grater than $a \cdot b$, that is $\{a \cdot b, a * b, a, b, a + b\}$ while $||a|| = \{a, a + b\}$. Term $a * a \cdot b$ corresponds just to the union of both sets $||a|| \cup ||a \cdot b|| = ||a \cdot b||$. Thus, 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.* □

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 obey a similar behaviour with respect to redundant causes. Take, for instance, a set of rules Π_7:

$$a : p \qquad b : p \leftarrow p$$

It is clear, that the cause a is sufficient to justify p, but there are also infinitely many other alternative and redundant causes $a \cdot b$, $a \cdot b \cdot b$, ... that justify p, that is $a + a \cdot b + a \cdot b \cdot b + \ldots$. 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 $\langle \mathcal{C}_{Lb}, \subseteq \rangle$.* □

The causal value $|||a|||$, the filter generated by the cause $||a||$, is the set containing $||a|| = \{a, a + b\}$ and all its super sets. That is, $|||a||| = \{||a||, ||a * b||, ||a \cdot b||, \ldots\}$. Thus, $a + a \cdot b + a \cdot b \cdot b + \ldots$ just corresponds to the union of the causal values generated by their addend causes, $|||a||| \cup |||a \cdot b||| \cup |||a \cdot b \cdot b||| + \ldots = |||a|||$.

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:*

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

[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.

Note that, any causal term can be normalized and then this definition trivially extend for 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).* □

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:

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

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

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

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 unfeasible – 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 $\gamma : \mathcal{V}_{Lb} \longrightarrow \mathcal{U}_{Lb}$ 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.* □

Given a term t we define its *canonical form* as $\gamma(\epsilon(t))$. Canonical terms are in 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 $a + ab + abb + \cdots = a$ respectively corresponds $\gamma(\epsilon(ab * a)) = \gamma(\epsilon(ab)) = ab$ and $\gamma(\epsilon(a + ab + abb + \dots)) = \gamma(\epsilon(a)) = a$ in canonical form. Figure 3 summarizes addition and product properties while Figure 4 is analogous for application properties.

Associativity	*Commutativity*	*Absorption*
$t + (u{+}w) = (t{+}u) + w$	$t + u = u + t$	$t = t + (t * u)$
$t * (u * w) = (t * u) * w$	$t * u = u * t$	$t = t * (t{+}u)$

Distributive	*Identity*	*Idempotence*	*Annihilator*
$t + (u * w) = (t{+}u) * (t{+}w)$	$t = t + 0$	$t = t + t$	$1 = 1 + t$
$t * (u{+}w) = (t * u) + (t * w)$	$t = t * 1$	$t = t * t$	$0 = 0 * t$

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

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

Addition distributivity	*Product distributivity*	*Annihilator*
$t \cdot (u{+}w) = (t{\cdot}u) + (t{\cdot}w)$	$c \cdot (d * e) = (c \cdot d) * (c \cdot e)$	$0 = t \cdot 0$
$(t + u) \cdot w = (t{\cdot}w) + (u{\cdot}w)$	$(c * d) \cdot e = (c \cdot e) * (c \cdot e)$	$0 = 0 \cdot t$

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 showed in Figures 3 and 4. For instance, the examples from $\mathit{\Pi}_6$ and $\mathit{\Pi}_7$ containing redundant information can now be derived as follows:

$$
\begin{array}{rcll}
a * a \cdot b & = & (a * 1 \cdot a \cdot b) & \text{identity for '}\cdot\text{'} \\
& = & 1 \cdot a \cdot b & \text{absorption for '}\cdot\text{'} \\
& = & a \cdot b & \text{identity for '}\cdot\text{'} \\
 & & & \\
a + a \cdot b + a \cdot b \cdot b + \ldots & = & a + 1 \cdot a \cdot b + a \cdot b \cdot b + \ldots & \text{identity for '}\cdot\text{'} \\
& = & a + a \cdot b \cdot b + \ldots & \text{absorption for '}\cdot\text{'} \\
& = & a + 1 \cdot a \cdot b \cdot b + \ldots & \text{identity for '}\cdot\text{'} \\
& \ldots & & \ldots \\
& = & a & \text{absorption for '}\cdot\text{'}
\end{array}
$$

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

$$
\begin{array}{rcll}
a \cdot b * c + a & = & (a \cdot b + a) * (c + a) & \text{distributivity} \\
& = & (1 \cdot a \cdot b + a) * (c + a) & \text{identity for '}\cdot\text{'} \\
& = & (a + 1 \cdot a \cdot b) * (c + a) & \text{commutativity for '}+\text{'} \\
& = & a * (c + a) & \text{absorption for '}\cdot\text{'} \\
& = & a & \text{absorption for '}*\text{'}
\end{array}
$$

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:*

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

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

For any rule ϕ of the form $t : L_0 \leftarrow L_1 \wedge \ldots \wedge L_m \wedge$ *not* $L_{m+1} \wedge \ldots$ *not* L_n we define $label(\phi) = t$. Most of the following definitions are standard in logic programming. We denote $head(\phi) = L_0$, B^+ (resp. B^-) to represent the conjunction of all positive (resp. negative) literals $L_1 \wedge \ldots \wedge L_n$ (resp. *not* $L_{m+1} \wedge \ldots \wedge$ *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. $\bot$) 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.* □

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*

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

for each rule $\varphi \in \Pi$ in the form $\varphi = L_0 \leftarrow L_1, \ldots, L_m$.

For instance, take rule (1) from Example 1 and let I be an interpretation such that $I(port) = p$ and $I(starb) = s$. 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 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 $T_\Pi : \mathcal{I} \longrightarrow \mathcal{I}$ such that, for any atom $p \in At$:*

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

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) [6]. GAP is a general a 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:

$$L_0 : \rho \leftarrow L_1 : \mu_i \;\&\; \ldots \;\&\; L_m : \mu_m \tag{3}$$

where $L_0, \ldots, L_m$ are literals, ρ is an *annotation* (may be just a truth value, an *annotation variable* or a *complex annotation*) and $\mu_1, \ldots, \mu_m$ 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 $f(\mu_1, \ldots, \mu_m)$ that applies the function f to a m-tuple $(\mu_1, \ldots, \mu_m)$ of annotation variables in the body of (3). Given a positive program Π, each rule $\varphi \in \Pi$ in the form

$$t : L_0 \leftarrow L_1 \wedge \ldots \wedge L_m \tag{4}$$

is translated to an annotated clause $GAP(\varphi)$ in the form of (3) where the annotation variables $\mu_1, \ldots, \mu_m$ in this case capture the causal values of the body literals and the complex annotation is defined as $\rho \stackrel{\text{def}}{=} (\mu_1 * \ldots * \mu_m) \cdot t$. 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 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. *$T_\Pi \uparrow^\omega (\mathbf{0}) = lfp(T_\Pi)$ 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.*

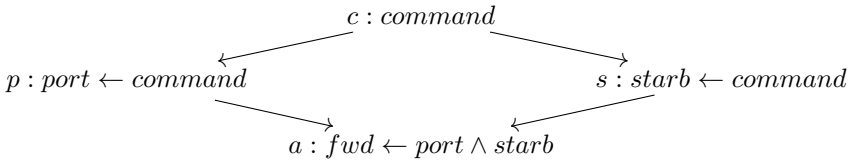
Proof. Directly follows from Theorem 2 and Theorems 1, 2 and 3 in [6].

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 most one vertex without outgoing edges denoted as $sink(T)$ when defined.
(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 $head(\varphi') = L_i$. □

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, although we talk about proof *tree*, we do not require an unique parent for each vertex, so we actually have a rooted, directed acyclic graph. 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 $t_1 \ldots t_n$ satisfying:*

1. *There exists a rule $\varphi \in V$ with $label(r) = t_1$ 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. $label(\varphi_i) = t_i$ and $label(\varphi_{i+1}) = t_{i+1}$.*
3. *$label(sink(T)) = t_n$.* □

Let us write $Paths(T)$ 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 $cause(T) \stackrel{\text{def}}{=} \prod_{t \in Paths(T)} t$. As an example, $cause(T_1) = (c \cdot p \cdot a) * (c \cdot s \cdot a)$. 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:*

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

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

From this result, it may seem that our semantics is 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 $\sum_{T \in PT_p} cause(T)$ may contain redundancies and is not, in the general case, in normal form. As an example, remember the program Π_6:

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

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^l_m (resp. I^l_m) 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^l_m is the least model of Π^l_m.* □

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 program[2]. 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.

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

4 Default negation and stable models

Consider now the addition of negation, so that we deal with arbitrary programs. In order to achieve a similar behaviour for default negation to that provided by stable models in the non-causal case, 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 Theorems 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:*

$$I(p) = \sum_{T \in PT_p} cause(T) \qquad \textit{where}$$

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

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). To illustrate the effect of default negation, suppose that, in Example 1, the actions for moving the boat forward can be disqualified if an *exceptional situation* occurs (for instance, that the boat is anchored). This can be easily represented using default negation as shown in the set of rules Π_8:

$$p : port \quad s : starb \quad w : fwind$$
$$a : fwd \leftarrow port \wedge starb \wedge \textit{not } ab \qquad b : fwd \leftarrow fwind \wedge \textit{not } ab$$
$$c : ab \leftarrow anchored$$

As expected, in the only stable model of the program Π_8 atom fwd is justified by $(p \cdot a * s \cdot a) + (w \cdot b)$ as in the program Π_2. That is, 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.

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 the existing rules. We have further provided three basic operations on the lattice: an addition, that stands for alternative, independent justifications; a product, that represents joint interaction of causes; and a concatenation that 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 [7]. This would surely simplify the quest for a necessary and sufficient condition for strong equivalence, following similar steps to [8]. 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 immediate 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 [9, 10] 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. In: Causal Logic Programming. 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. Pearce, D.: Equilibrium logic. Ann. Math. Artif. Intell. **47**(1-2) (2006) 3–41
8. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4) (2001) 526–541
9. 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
10. Pontelli, E., Son, T.C., El-Khatib, O.: Justifications for logic programs under answer set semantics. TPLP **9**(1) (2009) 1–56

A. Proofs

Proof of Proposition 1. We proceed by induction on the number of operators:

- If $t = l \in Lb$ then it is a normal term.
- For $t = \sum_{u \in S} u$ (dually for $t = \prod_{u \in S} u$) suppose by induction hypothesis that each causal subterm can be normalized. Thus let $U' = \{u' = normal(u) \mid u \in U\}$, then $t' = \sum_{u' \in U'} u'$ is the desired normal term.
- For $t \cdot \big(\sum_{u \in U} u\big)$ (resp. $t \cdot \big(\prod_{u \in U} u\big)$), applying left distributivity of '$\cdot$' with respect to '$+$' it follows that $t \cdot \big(\sum_{u \in U} u\big) = \sum_{u \in U} t \cdot u$ and by induction hypothesis each sum argument $t \cdot u$ has less operators and can be normalized. Thus $\sum_{u \in U} normal(t \cdot u)$ is the desired normal term.
- For $\big(\sum_{u \in U} u\big) \cdot l$ (resp. $\big(\prod_{u \in U} u\big) \cdot l$), applying right distributivity of '$\cdot$' with respect to '$+$' it follows that $\big(\sum_{u \in U} u\big) \cdot l = \sum_{u \in U} u \cdot l$ and by induction hypothesis each addend $u \cdot l$ is a term with less operators and can be normalized. Thus $\sum_{u \in U} normal(t \cdot l)$ is the desired normal term.

□

Proof of Proposition 3. $\gamma(V) = \sum_{C \in \underline{V}} \prod_{x \in \underline{C}} x$, thus $\epsilon\big(\gamma(V)\big) = \bigcup_{C \in \underline{V}} \bigcap_{x \in \underline{C}} \epsilon(x)$. Since each $x \in \mathcal{X}_{Lb}$ it follows that $\epsilon\big(\gamma(V)\big) = \bigcup_{C \in \underline{V}} \bigcap_{x \in \underline{C}} \{C' \in \mathcal{C} \mid x \in C'\} = \bigcup_{C \in \underline{V}} \{C' \in \mathcal{C} \mid \underline{C} \subseteq C'\}$. Furthermore $C'' \in V$ iff there is some $C \in \underline{V}$ s.t. $C \subseteq C''$ (possible C') iff $C'' \in \bigcup_{C \in \underline{V}} \{C' \in \mathcal{C} \mid \underline{C} \subseteq C'\}$, and thus $V = \epsilon\big(\gamma(V)\big)$. □

Proof of Theorem 2. For a rule $\phi \in \Pi$ in the form of (4) and an interpretation I, $I \models \phi$ if and only if $\big(I(L_1) * \ldots * I(L_m)\big) \cdot t \leq I(L_0)$ while $I \models^r GAP(\phi)$ if for all $\mu_i \leq I(L_i)$ implies that $\rho \leq I(L_0)$. Clearly, when each $\mu_i = I(L_i)$ follows $\rho \leq I(L_0)$ becomes $(\mu_1 * \ldots * \mu_n) \cdot t = \big(I(L_1) * \ldots * I(L_n)\big) \cdot t \leq I(L_0)$. That is, $I \models^r GAP(\phi)$ implies $I \models \phi$. Furthermore, if we take any $\mu'_i < I(L_i)$ follows $(\mu'_1 * \ldots * \mu'_n) \cdot t < (\mu_1 * \ldots * \mu_n) \cdot t \leq I(L_0)$, since it is a monotonic mapping. That is, $I \models \phi$ also implies $I \models^r GAP(\phi)$. To show that above translation gives *acceptable* programs, it is only necessary to note that the body of each translated rule $GAP(\phi)$ is only variable annotated. The correspondence between T_Π and $R_{GAP(\Pi)}$ can be done following the same above reasoning about satisfaction. □

Lemma 1. *Let Π be a positive logic program. For each atom p*

$$T_\Pi \uparrow^n (\mathbf{0})(p) = \sum_{T \in PT_p^n} cause(T)$$

where $PT_p^n = \{T = \langle V, E\rangle \mid V \subseteq \Pi,\ source(T) = (\pi : B \to p)$ and $height(T) \leq n\}$ and $height(T) = \max\{|x| \mid cause(T) = \prod_{x \in X} x\}$.

Proof. We proceed by induction on n. For $n = 0$ there are no proof trees, and so, $T_\Pi \uparrow^0 (\mathbf{0})(L_0) = 0$ for each literal L_0. When $n > 0$, by definition of T_Π

$$T_\Pi \uparrow^n (\mathbf{0})(L_0) = \sum_{(t : L_0 \leftarrow B) \in \Pi} \prod_{L_i \in B} T_\Pi \uparrow^{n-1} (\mathbf{0})(L_i) \cdot t$$

By induction hypothesis

$$T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \leq \sum_{T \in PT^{n-1}_{L_i}} cause(T)$$

$$\prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \leq \prod_{L_i \in B} \sum_{T \in PT^{n-1}_{q}} cause(T_q)$$

and applying distributivity

$$\prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \leq \sum_{\varphi \in \Phi} \prod_{L_i \in B} cause\big(\varphi(L_i)\big)$$

where $\Phi = \{\varphi \mid \varphi(q) \in PT^{n-1}_{q}\}$. Therefore

$$\prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \cdot t \leq \sum_{\varphi \in \Phi} \prod_{L_i \in B} cause\big(\varphi(L_i)\big) \cdot t$$

Let $\varphi(L_i) = \langle V_{\varphi(L_i)}, E_{\varphi(L_i)}\rangle$ and $T_\varphi = \langle V_\varphi, E_\varphi\rangle$ s.t.

$$V_\varphi = \{t : L_0 \leftarrow B\} \cup \bigcup_{L_i \in B} V_{\varphi(L_i)}$$
$$E_\varphi = \Big\{\Big(sink\big(\varphi(L_i)\big), (t : L_0 \leftarrow B)\Big) \mid L_i \in B\Big\} \cup \bigcup E_{\varphi(L_i)}$$

clearly $cause(T_\varphi) = \prod_{L_i \in B} cause\big(\varphi(L_i)\big) \cdot t$ and $height(T_\varphi) \leq n$, i.e. $T_\varphi \in PT^n_{L_0}$. Applying the above steps to each rule in the form $t : L_0 \leftarrow B$ we can see that

$$T_{\Pi}\uparrow^{n}(\mathbf{0})(L_0) = \sum_{(t:L_0 \leftarrow B) \in \Pi} \prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \cdot t \leq \sum_{T \in PT^n_{L_0}} cause(T)$$

Now, let $T \in PT^n_{L_0}$ i.e. $source(T) = (t : L_0 \leftarrow B)$ and $height(T) \leq n$. For each $L_i \in B$ there is a subtree $T_{L_i} \in PT^{n-1}_{L_i}$ and by induction hypothesis, $cause(T_{L_i}) \leq T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i)$ for each $L_i \in B$. Clearly

$$\prod_{L_i \in B} cause(T_{L_i}) \leq \prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i)$$

following

$$cause(T) = \prod_{L_i \in B} cause(T_{L_i}) \cdot t \prod_{L_i \in B} T_{\Pi}\uparrow^{n-1}(\mathbf{0})(L_i) \cdot t \leq T_{\Pi}\uparrow^{n}(\mathbf{0})(L_0)$$

i.e.

$$T_{\Pi}\uparrow^{n}(\mathbf{0})(L_0) \geq \sum_{T \in PT^n_p} cause(T)$$

Finally the case $n = \omega$ can be proved as follows:

$$T_{\Pi} \uparrow^{\omega} (\mathbf{0})(L_0) = \sum_{k<\omega} T_{\Pi} \uparrow^{k} (\mathbf{0})(L_0) = \sum_{k\leq\omega} \sum_{T\in PT^k_{L_0}} T =$$

$$= \sum_{T\in\bigcup_{k\leq w} PT^k_{L_0}} T = \sum_{T\in PT^{\omega}_{L_0}} T$$

Proof of Theorem 3. Since I is the least model of Π we conclude $T_{\Pi} \uparrow^{\omega} (\mathbf{0}) = I$ (Corollary 1) and the result then directly follows from Lemma 1. □

Proof of Corollary 2. From Theorem 3, $I(p) = \sum_{T\in PT_p} cause(T)$ where $PT_p = \{T \mid source(T) = t : p \leftarrow B\}$. Thus $I^l_m(p) = \sum_{T\in PT_p} cause(T^l_m)$ is the least model of Π^l_m where T^l_m is the tree obtained by replacing l by m in each tree node. □

Proof of Theorem 4. Since $V \subseteq \{(t : q \leftarrow B) \in \Pi \mid I(B^-) > 0\}$ for each tree $T = \langle V, E\rangle$ we conclude that $V \subseteq \Pi^I$. Therefore, $I(p) = \sum_{T\in PT_{p,I}} cause(T)$ (Theorem 3) is the least model of Π^I i.e. I is a causal stable model of Π. □