A causal semantics for Logic Programming

Jorge Fandiño García

Tese de doutoramento UDC / 2015

Director: Pedro Cabalar

Programa de Doutoramento en Computación

UNIVERSIDADE DA CORUÑA

A causal semantics for Logic Programming

Jorge Fandiño García

Doctoral Thesis UDC / 2015

Supervisor: Pedro Cabalar

Thesis committee:

Georg Gottlob
Torsten Schaub
Francesca Toni
Manuel Ojeda Aciego
Concepción Vidal Martín

UNIVERSIDADE DA CORUÑA

D. José Pedro Cabalar Fernández, Profesor Titular de Universidade na área de Ciencias da Computación e Intelixencia Artificial da Universidade da Coruña

CERTIFICA

Que a presente memoria titulada **A Causal Semantics for Logic Programming** foi realizada baixo a súa dirección e constiúe a Tese que presenta **Joge Fandiño García** para optar ó grado de Doutor pola Universidade da Coruña.

Asina: Dr. José Pedro Cabalar Fernández
Director da Tese

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Pedro Cabalar. In fact, this dissertation has its roots in his Phd Tesis and posterior works in the formalization of causality and it could not be conceived without his advice, direction and support.

I also would like to thank Michael Fink, for his warm welcome during my stay in the Knowledge-Based System Group. The development of great part of the work presented here could not be possible without his advice and supervision. Besides, I own a great dealt of the content of this dissertation to comments by Felicidad Aguado, Vladimir Lifschitz, Jesús Medina, David Pearce, Manuel Ojeda, Claudia Schulz, Carlos Viegas Damásio, Joost Vennekens, Concepción Vidal, and the anonymous reviewers of previous published work. I am also thankful to the committee members: Georg Gottlob, Torsten Schaub, Francesca Toni, Manuel Ojeda, Concepción Vidal, Luis Fariñas and Felicidad Aguado.

Very thanks to Antonio Blanco, director of the Computer Science Department, Álvaro Barreiro, director of the AI Lab, and the members of the AI Lab, Gilberto Pérez, Javier Parapar, Martín Diéguez and Daniel Valcarce, for their help and support during these years. I am also due to Thomas Eiter, Stefan Woltran, Reinhard Pichler, Hans Tompits and the rest of the members of the KRS and DBI groups at the Technical University of Vienna for they welcome and support during my stay in there. I would like to thank all my teachers that have directed my academic path. Special mention to Ramón Otero, José Graciano Pardo and María De La Concepción Pérez Yglesias.

As a personal acknowledgements, I want to thank the support of my friends and the unconditional support and advice of my family along my whole life, without not only this dissertation, but also myself, cannot be conceived.

This research has been supported by grants from the University of Corunna (Predoctoral Grant), the Governments of Galicia and the European Union (Predoctoral Grant INCITE 2011), the Government of Spain (Projects TIN2009-14562-C05-04 and TIN2013-42149-P) and a research stay grant from Inditex and University of Corunna.

Abstract

In this dissertation, we extend some of the major Logic Programming semantics, so that causal information is included into the models of a program. Technically, our approach consists in a multivalued semantics where atoms are associated to their causes instead of just assigning them a truth value, true or false. These causes are represented by *causal values*: sets of pairwise incomparable graphs that form a completely distributive lattice. Causal values can be useful for providing justifications explaining why a given atom holds in a model, but it is perhaps more interesting that they can be inspected into the body of program rules by a new kind of *causal literal*, allowing, for instance, to reason with statements of the form "*A* has caused *B*." In particular, we define *causal semantics* that are conservative extensions of the *least model*, the *well-founded model*, the *stable model* and the *answer set semantics* and show that, for several of the causal problems we investigate, reasoning under these semantics is computationally as hard as in their standard counterparts. We also provide methods and tools for computing reasoning tasks under these semantics and show how they can be useful for representing some traditional scenarios of the literature.

Resumen

En esta tesis, extendemos algunas de las principales semánticas de programación lógica, de manera que la información causal asociada con un átomo sea incluida en los modelos de cada programa lógico. Técnicamente, nuestra aproximación consiste en una semántica multivaluada donde cada átomo está asociado con sus causas en lugar de con un simple valor de verdad, verdadero o falso. Estas causas están representadas por *valores causales*: conjuntos de gráficos incomparables entre sí que forman un retículo completamente distributivo. Los valores causales pueden ser útiles para proporcionar justificaciones que expliquen por que un átomo dado es cierto en un modelo. Sin embargo, la característica más interesante es que estos valores puedan ser inspeccionados en el cuerpo de las reglas del programa por un nuevo tipo de *literal causal*, lo que permite, por ejemplo, razonar con enunciados de la forma "*A* ha causado *B*." En concreto, definimos *semánticas causales* que son extensiones de la *least model*, la *well-founded model*, la *stable model* y la *answer set semantics* y demostramos que, para varios de los problemas causales que investigamos, el razonamiento en estas semánticas no es computacionalmente más costoso que en sus correspondientes versiones estándar. También proporcionamos métodos y herramientas para el cálculo de las soluciones de diversas tareas de razonamiento bajo estas semánticas y mostramos cómo pueden ser útiles para la representación de algunos de los ejemplos tradicionales de la literatura. El apéndice C contiene un resumen extendido en castellano.

Resumo

Nesta tese, ampliamos algunhas das principais semánticas da programación lóxica de tal xeito que a información causal asociada cun átomo sexa incluída nos modelos de cada programa. Tecnicamente, a nosa aproximación consiste nunha semántica multivaluada, onde cada átomo está asociado ás súas causas no canto dun simple valor de verdade, verdadeiro ou falso. Estas causas son representadas por *valores causais*: conxuntos de grafos incomparables entre eles que forman un retículo completamente distributivo. Os valores causais poden ser útiles para proporcionar xustificacións que expliquen por que un átomo dado é certo nun modelo. Sen embargo, a característica máis interesante da nosa aproximación é que estes valores poden ser inspeccionados no corpo das regras do programa mediante un novo tipo de *literal causal*, permitindo, por exemplo, razóar con enunciados da forma "*A* causou *B*." En concreto, definimos *semánticas causais* que son extensións da *least model*, a *well-founded model*, a *stable model* e *answer set semantics* e amosamos que, para moitos dos problemas causais que temos investigado, as tarefas de razoamento nestas semántica non son computacionalmente máis custosas que nas correspondentes versións estándar. Tamén proporcionamos métodos e ferramentas para calcular as solucións de varias tarefas de razoamento sobre estas semánticas e amosamos como poden ser útiles para representar algúns dos exemplos tradicionais da literatura. O Apéndice D contén un resumo ampliado en galego.

CONTENTS

1 INTRODUCTION 23
- 1.1 Historical Overview of Causality . . . 27
- 1.2 Motivation . . . 39
- 1.3 Goals And Structure . . . 41

2 BACKGROUND 47
- 2.1 Answer Set Programming . . . 47
- 2.2 Abstract Algebra . . . 57
- 2.3 General Annotated Logic Programs . . . 61
- 2.4 Complexity: The Polynomial Hierarchy . . . 63

3 SUFFICIENT CAUSES 67
- 3.1 Causes as Graphs . . . 69
- 3.2 Causal Graphs and Logical Proofs . . . 73
- 3.3 Causal Values: Representing Alternative Causes . . . 77
- 3.4 Algebraic Properties of Causal Values . . . 83

4 CAUSAL SEMANTICS FOR LABELLED LOGIC PROGRAMS 91
- 4.1 Causal Semantics for Positive Programs . . . 91
- 4.2 Causal Answer Set Semantics . . . 97
- 4.3 Causal Well-founded Semantics . . . 106

5 ACTIONS DOMAINS EXAMPLES 109
- 5.1 Yale Shooting Scenario: Three Possible Representations . . . 110
- 5.2 The Block's World Scenario . . . 115
- 5.3 Lin's Suitcase Scenario and the Indirect Effects of Actions . . . 117
- 5.4 The Gear Wheels Scenario . . . 119

6 CAUSAL LITERALS 123
- 6.1 Formalization of Causal Literals . . . 124
- 6.2 Encoding in General Annotated Logic Programs . . . 128
- 6.3 Non-Monotonic Causal Literals for Action Domains . . . 132

7 QUERIES AND COMPLEXITY ASSESSMENT 135

7.1 Representing Causal Values 137
7.2 Membership Results 143
7.3 Completeness results 150
7.4 HasCaused Predicate 157

8 RELATED WORK 161
8.1 Why-not Provenance Justifications 163
8.2 Off-line Justifications for ASP 178
8.3 ABAS Justifications 181
8.4 Actual Causation 186

9 IMPLEMENTATION 207

10 CONCLUSIONS 221

Appendices 236

A PROOFS 237

B IMPLEMENTATION OF EXAMPLES 293

C RESUMEN 323

D RESUMO 333

LIST OF FIGURES

Figure 1 Proof for atom *bomb* corresponding to Example 1.2. 25

Figure 2 Relations inside the Polynomial Hierarchy. 65

Figure 3 Graphs G_1 and G_2 associated to the proofs of *bomb* in Example 1.2. 69

Figure 4 A circuit connecting four switches and an alarm. 70

Figure 5 Sufficient explanations for the alarm firing in Example 3.1. 71

Figure 6 Causal graph corresponding to graphs $z \rightarrow s$ and $z \rightarrow s \rightarrow s$. 72

Figure 7 Causal graph corresponding to graph G_1 in Figure 5. . . 72

Figure 8 Proof for atom *bomb* corresponding to Program 3.1. . . . 74

Figure 9 Causal graphs associated to the proofs of the literal *bomb* in Example 3.2. 74

Figure 10 Proof for atom *bomb* corresponding to graph G_1 w.r.t. Program 3.3 75

Figure 11 Proof for atom *bomb* corresponding to graph G_3 w.r.t. Program 3.3 75

Figure 12 Causal graphs associated to the proofs of the literal *bomb* in Example 3.2 with $\delta(A)$ a different label for each atom A. 76

Figure 13 Equations of the $(\cdot)$ operator (t, u, w are arbitrary causal terms). 79

Figure 14 Equations of the $(\cdot)$ operator (c,d,e terms without '$+$' and l is a label). 79

Figure 15 Equations of a completely distributive lattice. 79

Figure 16 Derived properties of the $(\cdot)$ and $(*)$ operations over causal graphs. 80

Figure 17 Iterative application $T_P^i(current(b))$ and $T_P^i(current(c))$. . 93

Figure 18 Iterative application $T_P^i(d)$. No simplification has been performed. 93

Figure 19 Iterative application $T_P^i(d)$ and $T_P^i(alarm)$. Simplification has been performed. 94

Figure 20 Iterative application $T_P^i(spinning(1))$ and $T_P^i(spinning(2))$. 95

Figure 21 Sufficient causes of t in Program 4.9. 105
Figure 22 Sufficient cause of *dead* in the Yale Shooting scenario. . . 113
Figure 23 Sufficient causes G_1 and G_2 of the dead in the Yale Shooting scenario with two shooters. 114
Figure 24 Sufficient cause G_1 of *open* in the suitcase scenario. . . . 118
Figure 25 Sufficient causes of $spinning(b)$ in the two variation of the gear wheel scenario. 121
Figure 26 Sufficient cause G_1 of *bomb* in the suitcase scenario. . . . 124
Figure 27 Sufficient cause G_1 of *bomb* in the suitcase scenario. . . . 126
Figure 28 Completeness results for deciding different types of causation in causal logic programs. *The third row refers to deciding sufficient cause w.r.t. programs which are not head labelled. 136
Figure 29 The t-graph associated to $I(p_3)$ in Program 7.1. 139
Figure 30 $I(\gamma)$ for all I and $I(\varrho)$ for $x_1 \vee \overline{x}_m$. 152
Figure 31 Off-line justifications of *bomb* in Program 3.1. 178
Figure 32 Off-line justifications of *bomb* in Program 3.1. 179
Figure 33 Off-line justifications of b w.r.t. $\{b,e,f\}$ in Program 8.6. . 180
Figure 34 Cause of b w.r.t. $\{b,e,f\}$ in Program 8.6. 180
Figure 35 Off-line justifications of *alarm* in Program 3.2. 181
Figure 36 Negative ABA Justification of *laserSurgey* with respect to Program 8.7. 183
Figure 37 ABA Justification of *intraocularLens* with respect to Program 8.7. 184
Figure 38 Positive dependences of *intraocularLens* with respect to Program 8.7. 184
Figure 39 Sufficient cause *intraocularLens* with respect to Program 8.7.184
Figure 40 ABA Argument for *intraocularLens* with respect to Program 8.7. 185
Figure 41 ABA argument A of *alarm* in Program 3.2. 185
Figure 42 ABA justification of *alarm* in Program 3.2. 186
Figure 43 Sufficient cause G_{suzy} of *shattered* with respect to Program 8.10 and G_{john} obtained after adding the labelled fact $\$: throw(john,0)$. 190
Figure 44 Graphs G_1 and G_2 associated to *dead* in Program 8.18. . . 194
Figure 45 A and C are joint productive causes of E. 202
Figure 46 E depends on C which is not productive causes of it. . . 203
Figure 47 E depends on C which is not productive causes of it. . . 203
Figure 48 The train is deliver to the right-hand track. 204

Figure 49 The train is deliver to the left-hand track. 205
Figure 50 The cause of *prison*(6,*suzy*). 211
Figure 51 Causes of *open*(6). 212
Figure 52 *cgraphs* flowchart. 219
Figure 53 Prueba del átomo *bomb* correspondiente con el Ejemplo C.2. 325
Figure 54 Proba do átomo *bomb* correspondente co Exemplo D.2. . . 335

LIST OF EXAMPLES

Example 1.1 Suitcase scenario 24
Example 1.2 Ex. 1.1 continued 24
Example 1.3 Ex. 1.2 continued 25
Example 1.4 Ex. 1.2 continued 28
Example 1.5 Rock Throwers 29
Example 1.6 Firing Squad 30
Example 1.7 Match 31
Example 1.8 Desert Traveller 40

Example 2.1 The gear wheel (simplified) 48
Example 2.2 School bus 53

Example 3.1 Alarm circuit 70
Example 3.2 Ex. 1.2 continued 73

Example 5.1 Yale Shooting Scenario 110
Example 5.2 Two shooters 112
Example 5.3 Block's World 115
Example 5.4 The gear wheel 119

Example 8.1 Dr. Smith 182
Example 8.2 Three men Firing Squad 193
Example 8.3 Ex. 8.2 continued 194
Example 8.4 Poison 196
Example 8.5 Water Plant 199
Example 8.6 The Engineer 204

1 INTRODUCTION

Causality is a concept firmly settled in commonsense reasoning, present in all kind of human daily scenarios. It has appeared in quite different cultures, both geographically and temporally distant, and is one of the central aims of many studies in physical, behavioural, social and biological sciences. However, its formalisation has proved to be an elusive matter that generates disagreement among experts from different fields. The importance of a reasonable formalisation of causality is revealed by the fact that causal intuitions not only affect people's common sense reasoning, but are also implicitly present in science or in formal reasoning. For instance, as commented by Pearl [2000] in his seminal book "Causality," Newton's second law of mechanics describes how a force applied to an object will *change* its state of motion:

> *"The alteration of motion is ever proportional to the motive force impressed; and is made in the direction of the right line in which that force is impressed."*

Something captured by the well-known mathematical equation:

$$a = \frac{f}{m} \tag{1}$$

The way in which (1) has been written contains some implicit knowledge about the cause-effect relation between force f and acceleration a that is not reflected in the semantics of the equation. In fact, according to the laws of Algebra, this equation can be equivalently rewritten both as $f = m \cdot a$ and $m = f/a$. We say that "the ratio f/a helps us to *determine* the mass" or, alternatively, that "the calculated mass *explains* why a given force has impressed the observed acceleration," but not that "it *causes* the mass." Similarly, the equation $f = m \cdot a$ helps us to *plan* what we have to do in order to impress a desired acceleration to an object of a given mass, but it does not mean that forces are caused by accelerations. These cause-effect considerations are usually taken into account when physicists use (1), but, as explained by Pearl, "such distinctions are not supported by the equations of physics." Pearl [1988] has further observed that something similar happens when we are formalising a wide class of abstract knowledge in a logical formalism. To illustrate this fact, consider the following scenario introduced by Lin [1995]:

Example 1.1 (Suitcase scenario)**.** *A suitcase has two locks and a spring-loaded mechanism that will become open when both locks are in the up position.* □

A straightforward representation of this domain could be the implication:

$$up(a) \wedge up(b) \ \supset \ open \tag{2}$$

stating that, when both locks are up, the suitcase will be opened. A classical theory consisting of the conjunction of the above implication plus the facts that both locks are moved up:

$$up(a) \ \wedge \ up(b)$$

will lead us to conclude that the suitcase will be opened. A *left-to-right* reading of *material implication* may mistakenly lead us to recognise some causal information in (2). However, as happened with (1), we can also write (2) equivalently as:

$$up(a) \wedge \neg open \ \supset \ \neg up(b) \tag{3}$$

or

$$up(b) \wedge \neg open \ \supset \ \neg up(a) \tag{4}$$

In these cases, the intuitive reading must change: we can *explain* that one switch is up and the suitcase is closed by concluding that the other switch is down, but *we do not consider* that $up(a) \wedge \neg open$ will cause $\neg up(b)$.

Furthermore, in some cases, explaining why some event has happened may be as important as predicting that it will happen. For instance, consider the following variation of the suitcase scenario.

Example 1.2 (Ex. 1.1 continued)**.** *The suitcase is connected to a mechanism that provokes a bomb explosion when it is open.* □

We may represent this scenario by adding an implication of the form:

$$open \ \supset \ bomb \tag{5}$$

People are usually interested on assigning responsibility of some occurred event to some given set of facts, especially in those cases that such event has important consequences as the explosion of a bomb. When an explanation for the *bomb* is required, we may use the formula (5) again in its left-to-right reading to determine that moving up both locks explains the bomb explosion. Moreover, if a detailed explanation is required, we can build a *deduction proof*, like the one shown in Figure 1.

$$\cfrac{\cfrac{\cfrac{\cfrac{\top}{up(a)} \qquad \cfrac{\top}{up(b)}}{up(a) \wedge up(b)}}{open}\;(2)}{bomb}\;(5)$$

Figure 1: Proof for atom *bomb* corresponding to Example 1.2.

A judge will not be satisfied with just a physical explanation of the bomb explosion, but will be more concerned with the application of a legal law that informally states the following:

Example 1.3 (Ex. 1.2 continued)**.** *Whoever causes a bomb explosion will be punished with imprisonment.* □

The formalisation of this sentence is a *challenge* problem for *Knowledge Representation* (KR) because it talks about "causing an explosion" without explicitly describing the *possible ways* in which *bomb* can be eventually caused. If we look for an *elaboration tolerant* solution, the formalisation of this law *should not* vary if a new way of causing *bomb* is included in the theory. A formalism is *elaboration tolerant* to the extent that it is convenient to modify a set of facts expressed in the formalism to take into account new phenomena or changed circumstances [McCarthy, 1998]. In order to obtain a solution that fulfils this criterion, we would need a kind of modal predicate "hascaused(A,B)" where *A* and *B* could be formulas in their turn, and then encode the law in Example 1.3 as:

$$hascaused(up(a), bomb) \supset prison \tag{6}$$

That is, if $up(a)$ is a cause of the *bomb*, then we may conclude that the agent performing $up(a)$ should go to prison. The problem obviously comes when trying to fix a meaning to predicate "hascaused," since its truth depends on the formalisation of the rest of the theory. For instance, we can easily derive facts for this predicate that follow from direct effects by adding formulas like:

$$up(a) \wedge up(b) \ \supset \ hascaused(up(a), open) \tag{7}$$
$$up(a) \wedge up(b) \ \supset \ hascaused(up(b), open) \tag{8}$$

In a similar way we may add a formula of the form:

$$open \ \supset \ hascaused(open, bomb) \tag{9}$$

so that we conclude a cause for the indirect effect *bomb*. Unfortunately, this method does not account for the transitive behaviour of causes: $up(a)$ and $up(b)$ *eventually* cause *bomb*. Of course, we could further add:

$$up(a) \wedge up(b) \ \supset \ hascaused(up(a), bomb) \tag{10}$$
$$up(a) \wedge up(b) \ \supset \ hascaused(up(b), bomb) \tag{11}$$

but, in the general case, we may have many indirect steps and even interplay with the inertia default or with recursive fluents. In other words, we would need a *complete analysis* of the theory as a whole in order to conclude the right axioms for predicate "hascaused." For instance, under the above representation, adding a new mechanism that opens the suitcase cannot be accomplished by just adding new facts, but we also have to include formulas that relates this mechanism to the bomb explosion, that is, we will be forced to add new direct effect formulas that causally connect these new ways of opening the suitcase and the *bomb*. This is an instance of the so called *ramification problem*, identified by Kautz [1986], which consists in being forced to represent the *indirect effects* of actions as *direct effects*.

A second drawback of (2), comes from assigning a left-to-right directional reading to *material implication*. It is important to notice that the difference between (2), (3) and (4) is just in their writing, but they are all equivalent, since material implication is just a case of disjunction, in this case:

$$open \ \vee \ \neg up(a) \ \vee \ \neg up(b)$$

Using the correct causal reading of (2) is now *crucial* since we fix the truth of predicate "hascaused" depending on that. To be precise, from (2) we want to conclude that:

$$(2) \ \wedge \ up(a) \ \wedge \ up(b) \ \models \ hascaused(up(a), open) \tag{12}$$

A similar reading for (3) would lead us to conclude:

$$(3) \land up(a) \land \neg open \models hascaused(\neg open, \neg up(b)) \tag{13}$$

something that *should not hold* when we replace (3) by (2) in (13): the intended causal reading of (2) should go from the antecedent (the locks) to the consequent (open) and not the other way around. Unfortunately, both formulas are classically equivalent, and so, this distinction cannot be made.

Contrarily to classical implication, in Logic Programming (LP), *rules* are directional. For instance, (2) would be written as the LP rule:

$$open \leftarrow up(a), up(b) \tag{14}$$

The implication symbol ($\supset$) is replaced by an arrow ($\leftarrow$), the conjunction symbol ($\land$) by a comma (,) and the position of antecedent and consequent are reversed. We may read (14), in a top-down fashion, as "to obtain *open* we may move *up* the two locks" or, in a bottom-up manner, as "moving up both locks allows deriving that the suitcase will be opened." We may also assign (14) the causal reading "moving *up* both locks will *cause* the opening of the suitcase." On the other hand, following this same rewriting, (3) will lead us to a different rule of the form:

$$\overline{up}(b) \leftarrow up(a), \overline{open} \tag{15}$$

where classical negation of an atom $\neg\alpha$ is rewritten here as $\overline{\alpha}$. Rule (15) would be read as "to move *up* lock *b* we may move *up* the lock *a* and close somehow the suitcase" or "moving *up* the first lock while closing the suitcase will cause the second lock to move down," something clearly wrong with respect to our causal understanding of Example 1.1. It is clear that the readings of (14) and (15) are very different, and indeed all LP semantics will treat them as two very different formulas. This distinction makes LP rules a suitable tool for representing *causal laws*.

1.1 HISTORICAL OVERVIEW OF CAUSALITY

We started this dissertation saying that causality has appeared in quite different cultures, both geographically and temporally distant. In particular, in Western

Philosophy, its roots may be traced back prior to Aristotle[1] who defined four kinds of causes. However, the concept of *cause* has evolved along time, and the modern scientific understanding of causality is a more reduced concept that only refers to Aristotle's efficient cause. In this dissertation, we will focus on the study of this modern understanding and, to be precise, on the concept of *sufficient cause*. Section 1.1.1 presents an overview of the concept of causality in Modern Philosophy while Section 1.1.2 overviews the approaches followed in Artificial Intelligence (AI).

1.1.1 Causality in Modern Philosophy

The roots of one of the more prominent approaches to causality in Philosophy can be traced back to Hume [1748], who defined cause using the idea of counterfactual:

> *"we may define a cause to be an event followed by another, [...], where if the first event hadn't occur the second wouldn't have occurred either."*

For instance, continuing with Example 1.1, the counterfactual "had lock a not been lifted, the suitcase would have not been open" identifies the action of moving up the lock a as a cause of opening the suitcase. We may reason analogously to conclude that lifting the second lock is also a cause of opening the suitcase. However, as Mill [1843] has observed, in general, an event may be counted as a cause of another event without being *necessary*. As an example, imagine that our suitcase has a second independent opening mechanism available.

Example 1.4 (Ex. 1.2 continued)**.** *The suitcase has also a second opening mechanism that works when a key is turned regardless of the locks movements.* □

Now, the key is turned after lifting the locks. Then, the suitcase would be open regardless whether the locks had been lifted or not and, thus, under Hume's definition, we would fail to identify the action of lifting any of the locks as a cause of opening the suitcase. This situation is representative of a more general set of examples where the following is an usual scenario [Lewis, 2000]:

1 In Phaedo, Plato discusses what cause means and mentions a previous book of Anaxagoras saying "that it is Mind that directs and is the cause of everything."

Example 1.5 (Rock Throwers)**.** *Suzy throws a rock at a bottle. The rock hits the bottle, shattering it. Suzy's friend, Billy, throws a rock at the bottle a couple of seconds later. Who has caused the bottle to shatter?* □

Intuitively, Suzy is the *actual cause* of the bottle shattering, whereas Billy is just a *preempted backup* that would have shattered the bottle had Suzy not done it. However, it is clear that the bottle shattering does not depend counterfactually on any of the throws, "had Suzy not thrown, the bottle would have shattered anyway because of Billy's throw" and vice-versa.

Lewis [1973] proposed a new definition of causation: an event will be a cause of another whenever there is a *causal chain* leading from the cause to the effect.

> *"Let c, d, e, … be a finite sequence of actual particular events such that d causally depends on c, e on d, and so on throughout. Then this sequence is a causal chain."*

That is, a *causal chain* is a finite sequence of actual events where each element counterfactually depends on its precedent. In this sense, Suzy's throw is connected to the bottle shattering by a finite sequence of events:

$$throw \cdot e_1 \cdot e_2 \ldots \cdot e_n \cdot shattered$$

in which each one counterfactually depends on the previous. Events e_1, e_2, …, e_n may be the position of the rock in the trajectory, the impact of the rock to the bottle, the shock wave that propagates over the bottle, and so on. On the other hand, Billy's throw is not connected to the shattering by such causal chain, something that allows distinguishing actual causes from just preempted backups. As a second example of causal chain, consider Example 1.2 in which (7) allows concluding that $up(a)$ has caused $open$, and (9) allows concluding that $open$ has caused $bomb$. Then, we may build the following causal chain:

$$up(a) \cdot open \cdot bomb$$

for concluding that $up(a)$ has caused $bomb$. In a similar manner, we may also build the following causal chain:

$$up(b) \cdot open \cdot bomb$$

for concluding that $up(b)$ has also caused $bomb$.

A further refinement of this idea of causality has been introduced by Lewis [1986]. In that work, he identified as *intrinsically alike* two processes that obey the same laws. Consider the following example from [Pearl, 2000, page 207]:

Example 1.6 (Firing Squad). *Suzy and Billy form a two-man firing squad that responds to the order of the captain. The shot of any of the two riflemen would kill the prisoner. Indeed, the captain gives the order, both riflemen shoot together and the prisoner dies.* □

In this example, both Suzy and Billy are usually considered *contributory causes* of the prisoner's death. Suzy's shot (when Billy is shooting) is intrinsically alike to Suzy's shot in another scenario where she shoots alone. Therefore, if Suzy's shot is a cause in the second scenario it should also be a cause in the first one. Thus, an event C should be regarded as a cause of an event E if it is connected by a process (causal chain) which is intrinsically alike to another process in which C is considered to be a cause of E. Lewis named this criterion as *quasi-dependence*. A more formal definition of intrinsicness was given by Hall [2007] as follows:

> *"Intrinsicness: Let S be a structure of events consisting of event E, together with all of its causes back to some earlier time t. Let S' be a structure of events that intrinsically matches S in relevant respects, and that exists in a world with the same laws. Let E' be the event in S' that corresponds to E in S. Let C be some event in S distinct from E, and let C' be the event in S' that corresponds to C. Then C' is a cause of E'."*

Note that, in Example 1.5, Billy's throw (when Suzy is throwing) is not intrinsically alike to the same action in another scenario where he is throwing alone because, in the former, his rock will not hit the bottle (which is already broken), whereas, in the latter, it will do so.

Another early attempt of formal definition of *contributory cause* was developed by Mackie [1965] who relied on the concept of the so called *INUS condition*:

> *"an Insufficient but Necessary part of a condition which is itself Unnecessary but Sufficient for the result."*

When Mackie says insufficient and unnecessary, he does not actually mean that an INUS condition *must be* insufficient and unnecessary, but rather that it *may actually be*. Consequently, Mackie's sentence may be rephrased just as "a necessary element in a sufficient set of conditions, NESS" as proposed by Wright [1988].

The basic idea behind the INUS condition consists in writing the laws of the investigated effect in *minimal disjunctive normal form*. A Boolean formula is in disjunctive normal form iff it is a disjunction of one or more conjunctions of one or more literals. It is in minimal disjunctive normal form if, in addition, there does not exist another equivalent formula in disjunctive normal form with less conjunctions or with less conjuncts in a conjunction. For instance, in our running example, the condition for the prisoner's death can be expressed in minimal disjunctive normal form as:

$$shoot(suzy) \ \vee \ shoot(billy)$$

An important observation is that the disjuncts, in our example, can be conjunctions of literals in the general case. To better illustrate this fact, let us consider a twist of our running example where both riflemen also have to load the gun before shooting it. In this case, the prisoner's death can be explained by:

$$load(suzy) \wedge shoot(suzy) \ \vee \ load(billy) \wedge shoot(billy)$$

Each conjunction, $load(suzy) \wedge shoot(suzy)$ and $load(billy) \wedge shoot(billy)$, is a sufficient condition for the prisoner's death. We may also assign a causal reading to these conditions, so that each of these conjuncts are alternative and independent *sufficient causes* for the prisoner's death. Similarly, the concept of *contributory cause* may be defined relying on the idea of INUS condition, that is, the four conjuncts, $load(suzy)$, $shoot(suzy)$, $load(billy)$ and $shoot(billy)$, are INUS conditions, and so, contributory causes for the prisoner's death.

In fact, the idea of contributory cause defined by an INUS condition has been shared by researches from several disciplines, such as Wright [1988] in law studies, Rothman [1976] and [Rothman and Greenland, 1998, page 53] in epidemiology or [Hoover, 1990, page 218] in economics. However, among other drawbacks that we will see later, this approach does not take into consideration the influence of *default knowledge* in the usual understanding of causality. For instance, we all know that cooling water below 0 degree Celsius will cause it to freeze but, in fact, this is only true under the default assumption of atmospheric pressure. At higher pressure, water will freeze at higher temperature, whereas, at lower pressures, we can have water vapour even below 0°C. Another example is the asymmetry that people assign to scratching a match and the presence of oxygen when identifying the causes of a fire.

Example 1.7 (Match). *Scratching a match would produce fire. Of course, if there is not enough oxygen present, the fire cannot be started.* □

Hitchcock and Knobe [2009] attribute the asymmetry between *match* and *oxygen* to the fact that the existence of *oxygen* is assumed to be the norm, or default behaviour. In fact, psychological tests have revealed that people tend to consider as causes those conditions that are somehow "abnormal" in the system, that is, opposed to what they consider to be its default behaviour. In particular, Kahneman and Tversky [1982], Mandel et al. [1985], Kahneman and Miller [1986], Alicke [1992], Roese [1997], Cushman et al. [2008] and Knobe and Fraser [2008] have run experimental tests that reveal such tendency with respect to statistical, moral and social abnormalities.

Our understanding of causality will borrow some technical features from both, Lewis and Mackie's approach, but, intuitively, it will be closer to that stated by Maudlin [2004], who considers that *causes* are those events that *divert* a system from following its *default behaviour*. A second point of agreement with Maudlin's approach will be the role played by causal laws, which are *prior* knowledge at the basis of causal connections.

> *"What links causation and counterfactuals [...] is natural law. Laws play one role in determining which counterfactuals are true, and another role in securing causal connections."*

Maudlin explains his concept of causality as a generalization of the Newtonian concepts of force and inertia: forces play the role of causes, in the particular case that the default behaviour of the system is inertial.

> *"The structure of Newton's laws is particularly suited to identify causes. There is a sense, I think, in which the continuation of inertial motion in a Newtonian universe* is not caused. *[...] Or at least, if there is any cause of a body at rest remaining at rest in a Newtonian universe it is a sort of* second-class *cause: The first-class Newtonian causes are forces (or the source of forces), and what they cause, the first-class form of a Newtonian effect, is a change or deviation from an inertial state of motion. There is no doubt that for Newton, once the first law is in place, one can ask what causes the Earth to orbit the Sun (rather than travel at constant speed in a straight line), and that the cause in this case is the gravitational force produced on the Earth by the Sun.*
>
> *[...]*
>
> *Let us denominate laws* quasi-Newtonian *if they have this form: There are, on the one hand,* inertial *laws that describe how some entities behave when nothing acts on them, and then there are laws of* deviation *that specify in what condition, and in what ways, the behaviour will deviate from the inertial behaviour."*

It is clear that Maudlin's concept of *inertial laws* expressed here is broader than the Newtonian's concept of the word. In fact, Maudlin's concept of inertial laws covers the whole idea of *default knowledge* in its usual meaning in KR. Of course, not all laws have to be quasi-Newtonian. However, as he points out:

> *"Our natural desire is to think of the world in quasi-Newtonian terms [...] [T]his is because it is much easier to think in these terms, to make approximate predictions on the basic of scanty data, and so on. And often circumstances allow us to think in quasi-Newtonian terms even when the underlying laws are not quasi-Newtonian [...] Indeed, the search for quasi-Newtonian laws does much to explain the aims of the special sciences."*

As an example of approximate reasoning, we all ignore the fine-grained physical concepts involved in turning on a light, and just assume that, to do so, we have to push the switch. Even if the laws that dominate our world are the rules of quantum physics, we reason in quasi-Newtonian terms, assuming that, if nothing happens, the light will not start to shine on its own.

1.1.2 Causality in Artificial Intelligence

In the AI literature, causality has been mostly addressed in two different ways that may be classified as *sufficiency-based* and *(contingent) necessity-based*. The former considers laws as *prior* knowledge that express a *sufficient* condition to cause an effect, and has emerged in the area of Reasoning about Actions and Change as a successful technical solution for solving some representational problems. The latter, which follows work by Pearl [2000], focuses on obtaining the *actual causes* of an event B by applying Hume's counterfactual "had A not happened, B would not have happened" under some possible contingency.

Necessity-based approaches

Pearl [2000] continues the research line followed by the above philosophical approaches: the study of the concept of *actual causation*[2]. As Halpern and Hitchcock [2011] write:

2 See [Halpern and Hitchcock, 2011] for a more actualised overview of the state of the art.

"Pearl's account was novel in [...] his use of structural equations as a tool for modelling causality. In the philosophical literature, causal structures were often represented using so called neuron diagrams, but these are not (and were never intended to be) all-purpose representational tools."

A central concept in Pearl's approach is the idea of *causal beam*. A causal beam is an alternative model where all the variables outside of some sufficient condition are fixed to a value that does not support the current value of the effect. Then, we may identify the *actual* and *contributory causes* of the effect by applying the Hume counterfactual analysis. Consider again Example 1.5. Halpern [2014] represents this scenario in the language of structural equations by introducing two extra variables $hit(suzy)$ and $hit(billy)$. The equations defining these variables are:

Structural Model 1.1.

$$\begin{array}{lcl} shattered & = & hit(suzy) \quad \vee \quad hit(billy) \\ hit(billy) & = & throw(billy) \wedge \neg hit(suzy) \\ hit(suzy) & = & throw(suzy) \\ throw(suzy) & = & 1 \\ throw(billy) & = & 1 \end{array}$$

From this model, we may conclude that Suzy is the responsible for breaking the bottle by applying the Hume's counterfactual "had suzy not thrown, the bottle would have not shattered" under the contingency where Billy does not throw. In Chapter 8, we will overview Halpern and Pearl's formal definition of the concept of actual cause [Halpern and Pearl, 2001, 2005, Halpern, 2008], and revise in detail this and other traditional problems of the actual causation literature.

From a KR point of view, this representation is mostly focused on distinguishing that Suzy is the actual cause of the shattering without putting too much attention on other representational problems. For instance, time is not explicitly represented in these equations, but instead implicitly incorporated in the equation defining $hit(billy)$ by asserting that billy does not hit the bottle if Suzy did. In line with this point of view, it is also interesting to separate knowledge about a given history (who throws the stone first) from the general knowledge about the system behaviour (which events may or may not cause an effect). The solution usually adopted in this approach mixes both sources of knowledge in the same equations. As a result, we may easily face a problem of *elaboration tolerance*. If we have N shooters and they shoot sequentially we would have to modify the equations for all of them so that the last shooter's equation would have the negation of the preceding N-1 and so on. All these equations would have to be reformulated if we change the shooting order.

Sufficiency-based Approaches

In contrast to necessity-based approaches, the use of causality in the area of Reasoning about Actions and Change was mostly focused on solving some problems of elaboration tolerance rather than answering questions like "*A* has caused *B*." The roots of causality in this area were settled in work by McCarthy [1959], who stated that one of the goals of AI was the development of programs capable of performing *common-sense reasoning* tasks like prediction, explanation or planning. This seminal work already included a definition of *causal assertions*, which consisted in delays of an unspecified number of situations between the condition (or cause) and its resulting effect. Most research in the area of Actions and Change, including the study of causal approaches, was motivated by the search for a solution to several representational problems, being the most significant one, the so called *frame problem* also identified by McCarthy and Hayes [1969]. The *frame problem*, consists in the unfeasibility of explicitly representing the persistence of all unaffected facts when an action is performed. The usual solution to this problem is the so called *commonsense law of inertia*

> *"A fluent remains unchanged under no evidence of the contrary."*

The commonsense law of inertia expresses a *default* behaviour, which derives new information *under the lack of information*. These new conclusions may be withdrawn in the presence of further evidence. As we have seen in the previous section, *default knowledge* has been recognised as a crucial factor in formalising causality. The need of default knowledge, and as a result, the need of deriving new information that may be withdrawn (in the presence of further evidence) is called *non-monotonicity* (as opposed to *classical logic monotonicity*). A logic is said to be *monotonic* if the more sentences we add to a theory, the more conclusions we obtain. The need of non-monotonicity moved the focus for representing dynamic domains from *classical logic* to new *non-monotonic* formalisms such as predicate *Circumscription* [McCarthy, 1980], *Default Logic* [Reiter, 1980] and modal non-monotonic logics [McDermott and Doyle, 1980]. Hanks and McDermott [1987] showed that previous approaches to solve the frame problem relying on minimal abnormality, were not adequate for representing the so called Yale Shooting Scenario. For overcoming this obstacle, Lifschitz [1987] and Haugh [1987] relied on the concept of causality in the so called *causal minimisations*.

However, the real interest about causality in action theories was raised by the study of the *ramification problem* (the unfeasibility of representing the *indirect effects* of the actions as *direct effects* of them). Lin's work was the first solution that successfully dealt with the frame and the ramification problems in *Situation Calculus* by incorporating a predicate $caused(F,V,S)$ whose meaning is that fluent F is caused to change its value to V in situation S, and it is an exception to the inertia [Lin, 1995]. In Lin's approach, an action domain is described by a Situation Calculus theory which is circumscribed in such a way that only those models with a minimal extension of the predicate $caused(F,V,S)$ are selected. Predicate $caused(F,V,S)$ expresses an exception to inertia, so that, minimal models are those with less exceptions to it. Then, a selection of those models that obey the law of inertia gives us the conclusions of the action domain.[3] For instance, in Lin's Situation Calculus, (2) would be rephrased as:

$$up(1,S) \wedge up(2,S) \supset caused(open,true,S) \tag{16}$$

This formula plus the facts[4] $up(1,s_2)$ and $up(2,s_2)$ allow us to conclude the fact $caused(open,true,s_2)$. An additional predicate $holds(F,S)$ is used to capture that fluent F holds in situation S, which is connected to predicate $caused(F,V,S)$ by a pair of axioms of the form:

$$\forall F,\ caused(F,true,S) \supset holds(F,S) \tag{17}$$

$$\forall F,\ caused(F,false,S) \supset \neg holds(F,S) \tag{18}$$

These axioms state that any fluent F must hold value V when it is caused to do so. The minimisation of the extension of predicate $caused(F,V,S)$ ensures that no other value is caused in those minimal models. The combination of minimising the extension of the predicate *caused* together with the selection of those minimal models that satisfy inertia allows concluding that fluents persist unless caused otherwise. In our running example, we will conclude $holds(open,s_3)$, $holds(open,s_4)$, and so on, because *open* is not caused to change.

3 A further elaboration due to Lin and Soutchanski [2011] adds a second minimisation of the extension of *caused* after selecting the minimal models satisfying the inertia axiom in order to rule out undesired models in theories with positive cycles.

4 Note that we represent here situations as natural numbers deviating from the standard formalisation of situations in Situation Calculus.

Looking at the definition of Lin's law of inertia, we can see that the main role of predicate *caused*(F,V,S) is allowing that a fluent F does not follow its default inertial behaviour. This fits with our understanding of *causes* as "events that *divert* a system from following its *default* behaviour." In this sense, the default behaviour of fluents is to remain unchanged, and causes are those events that break it. On the other hand, despite of the existence of predicate *caused*(F,V,S), Lin's approach does not allow answering the question "which are the causes that explain why the suitcase is open?" In fact, other solutions to the *frame problem* not labelled as "causal", like those by Gustafsson and Doherty [1996], Sandewall [1996] or Shanahan [1999], share similar technical constructions. Another difference, is that our understanding assumes that *all facts must have a cause*. In this sense, we may read the predicate *caused*(F,V,S) as "fluent F is caused to have value V in situation S" emphasizing the "in situation S" statement. That is, when *caused*(F,V,S) does not hold, it does not mean that F is not caused, but rather than F is following its default behaviour, and so the causes of its current state are the same of its previous state. This idea that "all facts must have a cause" is similar to the understanding of causality in the *Causal Explanations Theories* approach developed by McCain and Turner [1997]. However, in *Causal Explanations Theories* no distinction can be made between a system following its default behaviour from one that has deviated from it.

Another approach to solve the *frame* and the *ramification problem* was developed by Turner [1997], who showed how they could be successfully solved in *Default Logic* by making an appropriate use of inference rules to capture causal direction (see Lifschitz [2015] for an actualised view). For that time, Gelfond and Lifschitz [1991] had already introduced the *answer set semantics* for logic programs and shown that it essentially corresponds to a subset of *Default Logic*. This fact immediately allows logic programs to be used as a successful representational tool for action domains. In fact, as has been pointed out by Lifschitz [2002],

> *"[The] limitations of the language of logic programs play a positive role in this case by eliminating some of the 'bad' representational choices that are available when properties of actions are described in default logic."*

Nowadays, the *answer set semantics* is the core of the *Answer Set Programming* (ASP) paradigm [Marek and Truszczyńki, 1999, Niemelä, 1999] which has emerged as prominent tool for KR and *Non-Monotonic Reasoning* (NMR), not only from a theoretical point of view, but also irrupting in the industrial world due to the existence of efficient solvers, as those proposed by Niemelä et al.

[2000], Leone et al. [2006] and Gebser et al. [2011], that can be used for a fast computation of solutions [Baral, 2003, Brewka et al., 2011]. In this sense, ASP offers three main features of interest for the purposes of this dissertation:

- the ability for capturing *causal directionality*, inherited from LP rules,
- the capability for representing and reasoning with general *default knowledge* due to the *default negation* operator *not*,
- the existence of efficient solvers for computation of solutions.

These three features are the main motivations for choosing the *answer set semantics* as the underlying semantics that will be extended for representing and reasoning with causal explanations. Focusing on the capability for representing default knowledge, it is worth to mention that a literal of the form *not* A is usually read, in an informal way, as "there is no way of deriving A." We will identify later the causes of a literal A with the ways of deriving it, so that we may also read *not* A as "there is no cause for A." For instance, following with Example 1.5, we may represent the general knowledge about the system behaviour (which events may or may not cause the shattering) by the following rules:

$$\begin{array}{lcl} \mathit{shattered}(T+1) & \leftarrow & \mathit{throw}(X,T),\ \mathit{not}\ \mathit{shattered}(T) \\ \mathit{shattered}(T+1) & \leftarrow & \mathit{shattered}(T) \end{array}$$

whereas the knowledge about the given history (who throws the stone first) may be represented by the following set of facts:

$$\begin{array}{l} \mathit{throw}(\mathit{suzy},2) \\ \mathit{throw}(\mathit{billy},4) \end{array}$$

This separation of behaviour and narrative provides a higher degree of *elaboration tolerance*: we may add extra shooters or change the shooting order just by changing the set of facts, without changing the rules capturing the general knowledge about the system. Unfortunately, the existing LP semantics will not allow us to obtain an answer for the question: who has caused the bottle shattering?

Besides the above considerations for choosing the answer set semantics, it is also worth to mention that many of the works done in the area of Reasoning about Actions and Change that we have mentioned can be successfully encoded into ASP. For instance, Lee [2012] identifies a wide class of *canonical formulas* in which the *stable model semantics* and *Circumscription* coincide and translates, into ASP, both *Situation Calculus* and the *Event Calculus* of Kowalski and Sergot [1989]. Similarly, a wide class of action descriptions in the action language $\mathcal{C}$ [Giunchiglia and Lifschitz, 1998] has been encoded in ASP by Lifschitz and Turner [1999], while *Causal Explanation Theories* [McCain and Turner, 1997] has been recently encoded by Ferraris et al. [2012].

1.2 MOTIVATION

As we have seen in the previous section, the AI literature has addressed causality in two different and complementary manners: on the one hand, the necessity-based approaches have been focused on determining the actual causes of an event without putting too much attention to problems of elaboration tolerance; on the other hand, the sufficiency-based approaches have been focused on using causality for solving some problems of elaboration tolerance without putting much attention to conclude facts of the form "A has caused B." The core of this dissertation will be focused on representing and reasoning with the *causal explanations* for events. As a starting point, we will represent systems as logic programs, and we will read rules of the form $A \leftarrow B$ as "event B causes effect A." Causal explanations will consist on minimal disjunctive formulas, in a similar manner as Makie's approach, that will be transitively propagated by rules. We will also borrow Lewis' causal chains to represent causal ordering among events, inside the disjunctive normal form. Relying on this idea, we may provide the following definitions:

Definition 1.1 (Sufficient explanation). *Given a formula in minimal disjunctive normal form F, a conjunction of events C is a* sufficient explanation *iff C entails some conjunction of F, that is, iff $C \models F$.* □

Definition 1.2 (Necessary explanation). *Given a formula in minimal disjunctive normal form F, a conjunction of events C is a* necessary explanation *iff every conjunction of F entails C, that is, iff $F \models C$.* □

Definition 1.3 (Contributory explanation). *Given a formula in minimal disjunctive normal form, a conjunction of events C is a* contributory explanation *iff some minimal conjunction D of F entails C.* □

Note that, as Pearl [2000, page 314] pointed out: the limitations in the formalisation of Mackie's INUSS condition easily collided with the lack of distinction between "A implies B" and "not B implies not A" from classical implication. Pearl illustrates this problem with the following example.

Example 1.8 (Desert Traveller). *A desert traveller has two enemies. The first poisons his canteen, and the second, unaware of the first, shoots and empties the canteen. A week later, the traveller is found dead and the two enemies confess to action and intention. A jury must decide whose actions was the cause of the traveller's death.*

We may capture this scenario, ignoring the intermediate facts, drink and dehydration, as the following logic program.

Program 1.2.

$$\begin{array}{ll} death \leftarrow \overline{shoot}, poison & \qquad shoot \\ death \leftarrow shoot & \qquad poison \end{array}$$

Intuitively, it was the enemy who shot the canteen, and not the other who poison the water, who has killed the traveller. We may assign a minimal disjunctive Boolean formula to *death* by interpreting this logic program as a classical theory, obtaining the following definition of *death* in disjunctive normal form:

$$\neg shoot \wedge poison \;\vee\; shoot$$

This formula is not in minimal disjunctive normal form because it can be rewritten as:

$$poison \;\vee\; shoot$$

Unfortunately, in this rewriting, both actions are symmetric and, hence, both or none of them must be causes of the death. As we will see later in Chapter 8 (page 190), our approach will assign the value $shoot \cdot death$ making clear who actually killed the traveller.

A second drawback of this approach consists in formally capturing intermediate events. For instance, applying the same procedure to the suitcase scenario of Example 1.2 we will obtain the following definition:

$$up(a) \wedge up(b)$$

in which we have lost the role of the event *open* in the explosion of the bomb. Lewis' idea of *causal chain* may help us to explain how lifting the locks is connected to the bomb explosion. In our running example, there are two causal chains that explain the bomb: $up(a) \cdot open \cdot bomb$ and $up(b) \cdot open \cdot bomb$. However, Lewis' causal chains do not allow distinguishing whether either $up(a)$ and $up(b)$ are joint or alternative causes of the *bomb* explosion. For instance, these same causal chains will appear in an example in which rule (14) is replaced by:

$$open \leftarrow up(a) \tag{19}$$
$$open \leftarrow up(b) \tag{20}$$

We overcome this issue by *formally defining* Lewis' idea of causal chain, which we combine with the idea of minimal disjunctive normal form, such that, we may assign the following *causal value* to the *bomb* explosion:

$$\big(up(a) * up(b)\big) \cdot open \cdot bomb \tag{21}$$

We use the product $(*)$ symbol instead of conjunctions $(\wedge)$ for avoiding the confusion of causal values with logical formulas.[5] Furthermore, we use the symbol $(\cdot)$, which we will call *application*, for capturing the idea of causal chain. The algebraic expression (21) can be easily read as the proof depicted in Figure 1, where application $(\cdot)$ is replaced by a derivation (horizontal line) and product $(*)$ separates each subderivation. Finally, we use the sum $(+)$ symbol, instead of disjunction symbol $(\vee)$, for representing alternative, independent causes. To illustrate the idea of alternative causes consider the variation introduced by Example 1.4. Here, there is a second alternative, independent cause for the *bomb* explosion that is initiated by the *key*. This situation will be formally captured by the following causal value:

$$\big(up(a) * up(b)\big) \cdot open \cdot bomb \ \ + \ \ key \cdot open \cdot bomb \tag{22}$$

It is easy to apply Definitions 1.1, 1.2 and 1.3 to causal values, so that we can see that each addend of (22) is a sufficient explanation of the *bomb*, the fact $open$ is a necessary one, and each of the literals appearing in (22) is a contributory explanation.

1.3 GOALS AND STRUCTURE

Representing causes as causal values is a useful representation, close to the idea of proof, but still lacks of a formal definition. In particular, the goals of this dissertation are:

5 As we have seen above, logical equivalences of Boolean formulas lead to undesired outcomes, in particular *causal values* will not satisfy the law of excluded middle.

i) Formally defining the concept of causal value and the concepts of sufficient, necessary and contributory causes with respect to it. These three concepts will be close to their respective concepts of explanations given before. In particular, the concept of *sufficient explanation* will be closely related to the concept of *logical proof*, while the concept of *sufficient cause* will be close to the concept of *non-redundant logical proof*, that is, causes will be explanations that do not contain redundant, or unnecessary information.

ii) Studying the properties of causal values and how they can be manipulated by means of algebraic operations. In particular, we will show that causal values can be manipulated by means of three algebraic operations $(\cdot)$, $(*)$ and $(+)$. Furthermore, causal values can be alternatively described as either sets of causes or as a elements of a term (or free) algebra with these three operations.

iii) Providing a procedure for obtaining the causal information of a system represented as a logic program. For that purpose we define a causal semantics for logic programs, and we show that, for positive programs, causal values can be computed by an extension of the direct consequences operator for standard LP.

iv) Incorporating the possibility of obtaining causal information in presence of *default knowledge*. As we commented above, *default knowledge* will be represented by means of default negation *not*. We provide a causal semantics that extends the *stable model*, the *answer set* and the *well-founded model semantics*. We also provide means for computing causal values under these three semantics.

v) Exploring how these semantics can be applied for obtaining causal information in dynamic domains. In particular, we explore some of the usual benchmark problems for the literature of Reasoning about Actions and Change.

vi) Incorporating the capability of, not only deriving causal information from a given program, but also using it inside the program, so that we can perform usual reasoning tasks with this information. For instance, solving the representation of the judge's statement in Example 1.3. For this purpose, causal explanations will be represented as a kind of syntactic expressions that can be manipulated as first class citizens by a new kind of causal literal.

vii) Exploring the computational cost of solving the main problems associated to the above objectives. As usual, this will be accomplished by making a complexity assessment of the associated decision problems.

viii) Comparing our approach to other closely related approaches in the literature. In particular, we will focus on literature about explanations and justifications in logic programming and on literature about actual causation.

ix) Providing a prototypical implementation that can handle the kind of scenarios and reasoning problems derived from the above goals.

The general methodology of this proposal will be the standard in research on Computer Science, a cyclic sequence including: review of the state-of-the-art, problem definition, posing hypotheses, and deriving their formal proof or rebuttal. In particular, this dissertation has generated the following publications:

Pedro Cabalar and Jorge Fandinno, *Enablers and Inhibitors in Causal Justifications of Logic Programs*, In Logic Programming and Nonmonotonic Reasoning, 13th International Conference, LPNMR 2015, Lexington, September 27-30, 2015. Proceedings. (to appear)

Jorge Fandinno, *Towards deriving conclusions from cause-effect relations*, In Answer Set Programming and Other Computing Paradigms, 8th Workshop, ASPOCP2015, Cork, August 31, 2015. Proceedings.

Pedro Cabalar and Jorge Fandinno, *Explaining Preferences and Preferring Explanations*. In Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation 2014, Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday. Thomas Eiter, Hannes Strass, Mirosław Truszczyński and Stefan Woltran (eds). Lecture Notes in Computer Science, Volume 9060, 2015.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *Causal graph justifications of logic programs.* In Theory and Practice of Logic Programming, TPLP 14, (4-5) 603-618, 30th International Conference on Logic Programming, July 2014.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *A complexity assessment for queries involving sufficient and necessary causes.* In Proc. of the 14th European Conf. on Logis in Artificial Intelligence, JELIA'14, Funchal, Madeira, Portugal, September 24th-26th, 2014. Lecture Notes in Artificial Intelligence (8761), pp. 300-310, Springer-Verlag, 2014.

Pedro Cabalar and Jorge Fandinno, *An algebra of causal chains*. In Proc. of the 6th International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP'13, Istambul, Turkey, August 2013.

Jorge Fandinno, *Algebraic Approach to Causal Logic Programs*, Theory and Practice of Logic Programming 13 (4-5), On-line Supplement (Doctoral Consortium), August 2013.

The rest of this dissertation is organised as follows. The next chapter is an overview of known results that will be required through the text. Chapter 3 is a first approximation to causal explanations as non-redundant logical proofs. As a result, we obtain a possible definition of causal explanations, based on graphs with an order relation, that allow identifying minimality with non-redundancy. Furthermore, causal explanations will be represented as a kind of algebraic expressions that can be handled inside the formalism. In particular, these expressions, called *causal values*, have the form of elements of a term algebra that can be manipulated, as usual, by rewriting rules that follows a set of equivalences. Chapter 4 uses the algebra proposed in the previous chapter to provide several multivalued semantics for logic programs which are a proper extension of the standard least model, well-founded model, stable model and answer set semantics.

Chapter 5 shows how the *causal answer sets semantics* can be used to capture the causes of any fluent when dynamic domains are represented as a logic program. Furthermore, it is shown that different representations, that do not differ when we are restricted to the fluent truth values, lead to different causal values, which, in its turn, correspond to different intuitions about the causal behaviour of those fluents.

Chapter 6 introduces causal literals. Causal literals are a new kind of literals that allow inspecting the causal values of standard literals, formalising, for instance, the statement of Example 1.3. Chapter 7 introduces different kinds of queries, besides entailment, that are interesting to ask about causal information. It also studies their computational cost.

Chapter 8 is a comparison to several approaches for obtaining justifications for the *answer set semantics* and to Pearl *actual causation*. Chapter 9 overviews a prototypical tool, based on the ASP tool Clingo script API [Gebser et al., 2014], that allows computing the *causal answer sets* of a program. Chapter 10 contains a final conclusion, summarising our results and outlining possible directions for future work. Appendices A and B respectively contain the formal proofs of the results shown along this dissertation and the examples in the language of the *cgraphs* tool.

2 BACKGROUND

2.1 ANSWER SET PROGRAMMING

ASP is a major logic programming paradigm rooted in knowledge representation and reasoning [Marek and Truszczyński, 1999, Niemelä, 1999], and it is an emerging approach to modelling and solving search and optimization problems arising in many applications areas of AI including planing, reasoning about actions, diagnosis, abduction and beyond [Baral, 2003, Brewka et al., 2011]. In ASP, knowledge is represented by a *logic program*, whose building block are *atoms*, *literals*, and *rules*. *Atoms* are elementary propositions (or factual statements) that may be *true* or *false*; *literals* are atoms or their strong negation. *NAF-literals* are literals A, *positive NAF-literals*, and their negations *not* A, *negative NAF-literals*. *Rules* are compounded expressions of the form:

$$A \leftarrow B_1, \ldots, B_m, \mathit{not}\ C_1, \ldots, \mathit{not}\ C_n \tag{23}$$

where A and all B_i's and C_j's are literals. The literal A is usually called the *head* of the rule, whereas literals B_i's and C_j's are called the *body*. It is also commonplace to distinguish between the *positive body*, literals B_i's, and the *negative body*, literals C_j's. The usual understanding of (23) it that "A is concluded or derived to be *true* when all the positive literals and none of the negative literals in its body are concluded." One interesting feature of ASP is its versatility: its semantics was defined in many different ways (in fact, Lifschitz [2010] listed thirteen of them), indicating an intrinsic richness of the concept. For our concerns, (23) can be understood as "all B_i's together cause the event A unless some C_j avoids it." For instance, the following variation of rule (14):

$$\mathit{open} \leftarrow \mathit{up}(a), \mathit{not}\ \mathit{down}(b) \tag{24}$$

may be read as "moving the first lock up, positive literal $\mathit{up}(a)$, causes the suitcase to *open*, *unless* the second lock is down, negative literal $\mathit{down}(b)$." Rules may have no body, that is $n = m = 0$, as for instance:

$$\mathit{up}(a) \leftarrow \tag{25}$$

in which case are called *facts*, in the sense that the head is unconditionally true, and usually the symbol $\leftarrow$ is removed. That is, fact (25) is usually written just as $up(a)$. A *program* is a set of rules. If a rule contains no negative literals, that is $m = 0$, then we say it is *positive*. *Positive programs* are those that only contain positive rules.

It is important to note that the *not* operator, called *default negation* is not *classical negation*. For instance, the negative literal *not* $down(b)$ should be read as "there is no way to derive $down(b)$" rather than $down(b)$ is false. In a causal sense, it should be read as "there is no cause of $down(b)$" rather than "there is a cause for $down(b)$ being false." If we consider the program formed by the two rules (24) and (25), we will conclude that $up(a)$ has been caused to be true because of fact (25), while we cannot conclude that $down(b)$ holds because there is no rule to cause it. This, in its turn, allows us to conclude that the suitcase is $open$, despite that there is no cause for $down(b)$ being false. Let us compare the understanding of (24) with the original rule:

$$open \leftarrow up(a),\, up(b) \tag{14}$$

If we consider a program formed by rule (14) plus (25) we cannot conclude that $open$ holds because there is no cause for $up(b)$ being true. As we commented in the introduction, the formalisation of the intuitions behind *default knowledge* and in particular *default negation* posed a challenge to the knowledge representation and logic programming communities for years. Eventually, the *stable model semantics* proposed by Gelfond and Lifschitz [1988] was one solution that has gained acceptance. In order to illustrate the key points of stable models, consider the following a simplification of the gear wheels scenario from McCain [1997]

Example 2.1 (The gear wheel (simplified)). *Consider a gear mechanism with a pair of wheels such that when one of them turns the other turns too. There is another switch to connect or disconnect the wheels.* □

We may represent an scenario where we press the switch to connect the wheels by the following logic program:

$$spinning(1) \leftarrow spinning(2),\, coupled \tag{26}$$
$$spinning(2) \leftarrow spinning(1),\, coupled \tag{27}$$
$$coupled \leftarrow switch \tag{28}$$
$$switch \tag{29}$$

In this program, we may conclude that we have pressed the switch, which is asserted by fact (29). In its turn, this fact together with rule (28) caused the wheels to be *coupled*. We may think now that *couple* is also a fact as *switch*. Then, in order to derive that the first wheel is spinning, represented by the atom $spinning(1)$, we must conclude that the second one is spinning in its turn, atom $spinning(2)$, and that they are *coupled*. We have that, indeed, they are *coupled*, but the fact that the second wheel is spinning we depends on the fact that the first one is spinning. This "vicious cycle" cannot be broken because there is no other rule that allows to cause neither the atom $spinning(1)$ nor $spinning(2)$. Then, we conclude that the set of facts $\{switch, coupled\}$ are caused to be true and we assume that atoms $spinning(1)$ and $spinning(2)$ do not hold because there is no cause for them. This bottom-up procedure was initially formalized, for positive programs, by van Emden and Kowalski [1976] .

Definition 2.1 (Direct consequences)**.** *Given a positive logic program P over a set of atoms At, the operator of* direct consequences *is a function $T_P : 2^{At} \longrightarrow 2^{At}$ such that:*

$$T_P(S) \stackrel{\text{def}}{=} \{\, A \mid (A \leftarrow B_1, \ldots, B_m) \in P \text{ and } B_i \in S \text{ for } 1 \le i \le m \,\}$$

for any set of atoms $S \subseteq At$. □

Then, for any set S, the iterative procedure we have seen is defined in the following way:

$$\begin{aligned} T_P^0(S) &\stackrel{\text{def}}{=} S \\ T_P^{i+1}(S) &\stackrel{\text{def}}{=} T_P(T_P^i(S)) \end{aligned}$$

We can compute the conclusions of a program by applying the iterative procedure starting with a set with no conclusions, that is the empty set $\emptyset$. For instance, in our running example, we have that

$$\begin{aligned} T_P^0(\emptyset) &= \emptyset \\ T_P^1(\emptyset) &= \{\, switch \,\} \\ T_P^2(\emptyset) &= \{\, switch, coupled \,\} \\ T_P^3(\emptyset) &= \{\, switch, coupled \,\} \\ &\ldots \end{aligned}$$

In fact, it is easy to see that $T_P^i(\emptyset) = \{\, switch, coupled \,\}$ for all integers $i \ge 2$. The set $T_P^2(\emptyset)$is the *least fixpoint* of the direct consequences operator. In fact, for every positive program there is always such a *least fixpoint* which, furthermore, coincides with the unique *classical minimal model* of the program when the operator ($\leftarrow$) is understood as the classical implication ($\subset$) and the (,) is understood as the conjunction ($\wedge$). This bottom-up procedure can be extended to programs with negation. For instance, if we apply this procedure to the program formed by the two rules (24) and (25) it follows that:

$$\begin{aligned} T_P^0(\emptyset) &= \emptyset \\ T_P^1(\emptyset) &= \{\ up(a)\ \} \\ T_P^2(\emptyset) &= \{\ up(a),\ open\ \} \\ T_P^3(\emptyset) &= \{\ up(a),\ open\ \} \\ &\dots \end{aligned}$$

However, this procedure does not work properly for every program with negation. If, for instance, we apply the same procedure to the program formed by the following pair of rules:

$$up(a) \leftarrow not\ down(a) \tag{30}$$

$$down(a) \leftarrow not\ up(a) \tag{31}$$

stating that the first lock must be up when it is not down and vice versa, it would follow:

$$\begin{aligned} T_P^0(\emptyset) &= \emptyset \\ T_P^1(\emptyset) &= \{\ up(a),\ down(a)\ \} \\ T_P^2(\emptyset) &= \emptyset \\ T_P^3(\emptyset) &= \{\ up(a),\ down(a)\ \} \\ &\dots \end{aligned}$$

and we never will reach a fixpoint, and thus we do not know which atoms we may conclude and which we may not.

2.1.1 The Stable Model Semantics

A way to overcome this problem was proposed by Gelfond and Lifschitz [1988], and it consists in starting with the assumption of which atoms would hold and which atoms would not. For instance we may assume that the conclusions of the above program is the set $\{up(a)\}$. Since we have assumed that $down(a)$ would not hold we can use rule (30) to derive that $up(a)$ must hold, and, since we have assumed that $up(a)$ would hold, we cannot use rule (31) to do the same with atom $down(a)$. Notice that, by assuming that $\{up(a)\}$ would be the set of conclusions, we have derived that $\{up(a)\}$ is, indeed, the set of conclusions. The set $\{up(a)\}$ is said to be *stable* in the sense that assuming it to be the set of conclusions of the program we confirm our assumption. In the same sense, the set $\{down(a)\}$ is also stable. On the other hand, the sets $\emptyset$ and $\{up(a), down(a)\}$ are not stable because, if we assume one of them to be the conclusions of the program, then we will derive that the other must be that set of conclusions.

Definition 2.2 (Reduct)**.** *Given a program P and a set of atoms S, the reduct of P with respect to S, in symbols P^S, is the program obtained from P by:*

1. *removing every rule of the form (23) such that some $C_i \in S$,*
2. *removing every negative literal from the remaining rules.* □

For instance, the reduct of the program formed by our pair of rules (30) and (31) with respect to the set $\{down(b)\}$ is obtained by firstly removing the rule (30):

$$\cancel{up(a) \leftarrow not\ down(b)} \tag{30}$$
$$down(b) \leftarrow not\ up(a) \tag{31}$$

and then removing the negative literal $not\ up(a)$ from rule (31):

$$down(b) \leftarrow \cancel{not\ up(a)} \tag{31}$$

The remaining is a positive program formed by the fact $down(b)$, from which we conclude that $\{down(b)\}$ is the set of conclusions of the program, that is, it is stable. On the other hand, the reduct of such program with respect to the empty set $\emptyset$ corresponds to the pair of facts $up(a)$ and $down(b)$, from which we conclude that $\{up(a), down(b)\}$ must be the set of concussion of the program. That is, $\emptyset$ is not stable.

Definition 2.3 (Stable Model)**.** *Given a program P, a set of atoms S is a* stable model *of P, in symbols $S \models P$, iff S is the least fixpoint of P^S.* □

The reduct of every program is positive and, thus, there always exists a least fixpoint of P^S. Furthermore, the reduct of a positive program is always itself.

Then, positive programs always have a *unique stable model* that coincides with its *unique classical minimal model*. In contrast, non-positive programs may have one, several or no *stable models*. The *stable models* of any program are always *minimal classical models*. But, perhaps, the most interesting feature is that the converse does not hold, that is, there may be minimal classical models that are not stable models. For instance, as we already have seen, the program formed by the rules (24) and (25):

$$open \leftarrow up(a),\ not\ down(b) \tag{24}$$

$$up(a) \tag{25}$$

has a unique stable model $\{up(a), open\}$, whereas if we see it as a classical theory, there is a second minimal classical model, $\{up(a), down(b)\}$, which is not stable. Notice the importance of this fact for representing causality. In the first case, we conclude that $up(a)$ has caused $open$ because there is not any cause for $down(b)$ which agrees with our reading of rules. On the contrary, in the second case, we would have concluded $down(b)$, but which is its cause? Neither (24) nor (25) has $down(b)$ in the head, so that, none of them could have caused it.

2.1.2 The Answer Set Semantics

In our running example we are treating the literal *down* as the opposite of *up* and, indeed, this is the case in English, but from a program point of view, *up* and *down* are just two different names with no relation between them. For instance, a program may conclude $up(b)$ and $down(b)$ without being considered inconsistent. We need a way to represent the complementary of a literal *up*. We recall that *not up* means that there is no cause for *up* being true, rather than there is a cause for *up* being false, or *down*. What we need is an explicit negation $\overline{up}$, similar to the classical logic negation, which represents that *up* is false[1]. We can now rewrite rule (24) as:

$$open \leftarrow up(a),\ not\ \overline{up}(b) \tag{32}$$

using the literal $\overline{up}(b)$ instead of $down(b)$. In order to further illustrate the difference between *default* and *strong* negation consider the following example[2] borrowed from Gelfond and Lifschitz [1991]:

1 This kind of negation is usually called *classical* or *strong* negation and represented either as $\neg up$ or $\sim up$. We will use instead $\overline{up}$ to represent it. Note that $\overline{\overline{up}} \equiv up$.

2 Gelfond and Lifschitz [1991] attributes this example to John McCarthy.

Example 2.2 (School bus)**.** *A school bus may cross a railway under the condition that there is not approaching train.* □

This fact can be expressed by the rule:

$$cross \leftarrow not\ train \tag{33}$$

However, this representation may be dangerous if the lack of cause for concluding the atom *train* is interpreted as the absence of an approaching train. Suppose that the train is coming but the driver's vision is blocked, so that, she cannot see it. Then, she would conclude that she may cross but, of course, we do not want the bus crossing the tracks. Instead, we should use the following representation to capture this scenario:

$$cross \leftarrow \overline{train} \tag{34}$$

In this case, the driver needs a cause to concluding that the train is not coming, for instance, that she saw that the tracks were empty.

We present here the basis of the *answer set semantics* that extends the *stable model semantics* [Gelfond and Lifschitz, 1991]. A set S of literals is an *answer set* of a positive program P (which does not contain the default negation operator *not*) iff

1. S is the least model of P when each pair of complementary atoms, a and $\overline{a}$ are treated as completely different atoms, and

2. if S contains a pair of complementary atoms, a and $\overline{a}$, then S must contain all atoms and all their complements.

Definition 2.4 (Answer Set)**.** *Given a program P, a set of literals S is an* answer set *of P, in symbols $S \models P$, iff S is the unique answer set of P^S.*

If a program does not contain strong negation its *answer sets* are exactly its *stable models*. It is also worth to mention, that under the *stable model semantics*, when we cannot conclude that some atom is true, then it is assumed to be false. This is what is called the *closed world assumption* [Reiter, 1987]. However, under the *answer set semantics*, we have an explicit way to say that something is false, and when we cannot conclude that an atom is neither true nor false we consider that it is just *unknown*. For instance, in Example 2.2, when the driver cannot see the tracks, she cannot conclude neither that the train is coming nor that it is not coming, but rather that *train* is just *unknown*. The *closed world assumption* for a given atom a can be restored by explicitly adding a rule of the form:

$$\overline{a} \leftarrow not\ a \tag{35}$$

Moreover, now truth and falsity are treated symmetrically, so we can assert that an atom a is *true by default* by adding a rule of the form:

$$a \leftarrow not\ \overline{a} \tag{36}$$

This can be generalised by more complex *default statements* that do not simply involve a default value, but rather a default behaviour. For instance, we may assert that "a is *true by default* when b holds" by adding the rule:

$$a \leftarrow b,\ not\ \overline{a} \tag{37}$$

2.1.3 Splitting a Logic Program

Consider now that we have a program formed by the following rules:

$$\begin{aligned} open &\leftarrow up(a),\ not\ \overline{up}(b) \\ up(a) &\leftarrow not\ up(b) \\ up(b) &\leftarrow not\ up(a) \end{aligned}$$

We know that program formed by rules:

$$\begin{aligned} up(a) &\leftarrow not\ \overline{up}(a) \\ \overline{up}(a) &\leftarrow not\ up(a) \end{aligned} \tag{38}$$

has two answer sets, $\{up(a)\}$ and $\{\overline{up}(a)\}$. From $up(a)$ and the rule:

$$open \leftarrow up(a),\ not\ \overline{up}(b) \tag{39}$$

we will conclude that *open* is true. Therefore, intuitively, it may seem that the sets $\{up(a), open\}$ and $\{\overline{up}(a)\}$ should be the stable models of the above program, and, indeed, they are. This intuitive idea was first formalized by Lifschitz and Turner [1994] relying on the so called *splitting sets*. We present here an alternative version due to Gelfond and Zhang [2014].

Theorem 2.1 (Splitting)**.** *Let P_1 and P_2 be a partition of a program P such that no atom occurring in P_1 is a head atom of any rule in P_2. Let S' be a set of atoms containing all head atoms of P_1 but no head atoms of P_2. A set S of atoms is an answer set of P if and only if $S \cap S'$ is an answer set of P_1 and S is an answer set of $(S \cap S') \cup P_2$.* □

In our running example P_1 would correspond to a program containing the rules in (38), while P_2 would just contain the rule (39). Then S' is the set $\{up(a), \overline{up}(a)\}$. Picking $S = \{up(a), open\}$, it follows that $S \cap S' = \{up(a)\}$ is an answer set of P_1, and as we have seen, S is also an answer set of the program $\{up(a)\} \cup P_2$. Similarly, $\{\overline{up}(a)\}$ is an answer set of P_1 and it is also an answer set of $\{\overline{up}(a)\} \cup P_2$, so it is an answer set of P.

2.1.4 Programs with Variables and Grounding

Continuing with the suitcase example (Example 1.1), we may represent the statement "lifting any of the locks causes it to be up" by adding the following pair of rules:

$$up(a) \leftarrow lift(a) \tag{40}$$
$$up(b) \leftarrow lift(b) \tag{41}$$

However, this has the inconvenient of forcing us to write a rule for each possible lock, and worse, if we later get knowledge of new locks, then we will need to add further rules for every statement involving locks. The solution is to allow the use of variables in rules. For instance, rules (40) and (41) can be replaced by the following single rule:

$$up(X) \leftarrow lift(X) \tag{42}$$

where X is a variable which can take the values a or b. The following revisits the given definitions to accommodate variables.

Definition 2.5 (ASP signature)**.** *The signature of a logic program consists of three sets of objects: variables, function symbols and predicate symbols.*

As usual in logic programming, we will write function and predicate symbols starting by a lowercase letter, while variables are written starting with uppercase. Function symbols of arity 0 are called *constants*. A *term* is then inductively defined as follows:

1. Variables are terms
2. If f is an n-ary function symbol and $t_1, \ldots, t_n$ are terms then $f(t_1, \ldots, t_n)$ is a term. Note that if f is 0-ary, that is, a constant, we omit the parentheses and write just f.

A term is said to be *ground* if no variable occurs in it. The *Herbrand Universe* denoted by *HU* is the set of all ground terms which can be formed with the functions (and constants) of the program signature.

An *atom* is a formula of the form $p(t_1,\ldots,t_n)$ where p is an n-ary predicate symbol and each t_i is a term. An atom is said to be *ground* iff every term occurring in it is ground. The *Herbrand Base*, denoted *HB*, is the set of all ground atoms that can be formed with predicates and functions from the program signature. A *literal* is either an atom $p(t_1,\ldots,t_n)$ or its complementary $\overline{p}(t_1,\ldots,t_n)$. A *NAF-literal* L is either a literal A (positive NAF-literal) or its default negation *not* A (negative NAF-literal). A *rule* is an expression of the form:

$$A \leftarrow B_1,\ldots, B_m, \textit{not } C_1,\ldots, \textit{not } C_n \tag{23}$$

where A, B_i's and C_j's are literals. The *grounding* of a rule with respect to a program is the set of rules that can be formed by all possible substitutions of elements of the Herbrand Universe for the variables in the rule. The *grounding* of a program is formed by the union of the grounding of all its rules with respect to that program.

For instance, if we consider the positive program consisting of the single rule (42), then its grounding would contain no rules, but if we add the facts $lift(a)$ and $lift(b)$, then we would obtain the program formed by rules (40) and (41) plus these two facts. It is important to notice that both the grounding and the answer sets of a finite program may be infinite. Consider, for instance, the program formed by the following pair of rules describing the natural numbers:

$$nat(s(X)) \leftarrow nat(X) \tag{43}$$

$$nat(0) \tag{44}$$

whose Herbrand Base and unique answer set is the infinite set

$$\{nat(0),\ nat(s(0)),\ nat(s(s(0))),\ldots\}$$

It is easy to see that, in the ground program, there is a rule of the form $nat(s^{i+1}(0)) \leftarrow nat(s^i(0))$ for every natural number i where we write $s^0(0)$ instead of 0 and $s^{i+1}(0)$ in place of $s(s^i(0))$.

2.1.5 Well-founded Semantics

The well-founded semantics is a closely related semantics for normal logic programs originally defined by Van Gelder et al. [1988, 1991] relying on the concept of *unfounded sets*. We will focus on an alternative definition based on *alternating fixpoints* given by Van Gelder [1989].

Given a program P over a set of atoms At, we denote by $\Gamma_P : 2^{At} \longrightarrow 2^{At}$ an operator that maps each interpretation I to the least model of the positive program P^I. Hence, by definition, the stable models of a program P are exactly the fixpoints of the Γ_P operator. The Γ_P operator is anti-monotonic, that is, for interpretations $I \leq J$ it holds that $\Gamma_P(I) \geq \Gamma_P(J)$. Hence the Γ_P^2 operator, given by $\Gamma_P^2(I) = \Gamma_P(\Gamma_P(I))$, is monotonic and, by Knaster and Tarski's theorem [Tarski, 1955], Γ^2 has a least and a greatest fixpoint, respectively denoted by $\mathtt{lfp}(\Gamma_P^2)$ and $\mathtt{gfp}(\Gamma_P^2)$. The well-founded model of a program P is given by a pair $\langle \mathtt{lfp}(\Gamma_P^2), \mathtt{gfp}(\Gamma_P^2) \rangle$.

We say that a program P satisfies an atom A under the well-founded semantics, in symbols $P \models_{wf} A$, when $A \in \mathtt{lfp}(\Gamma_P^2)$. On the contrary, P does not satisfy A, in symbols $P \not\models_{wf} A$, when $A \notin \mathtt{gfp}(\Gamma_P^2)$. Otherwise, A is said to be *undefined* with respect to P.

Proposition 2.1 (Baral [2003]). *Given a logic program P, then $P \models_{wf} A$ implies $P \models A$ and $P \not\models_{wf} A$ implies $P \not\models A$.* $\square$

Proposition 2.1 states that the well-founded semantics is sound with respect to the stable model semantics.

2.2 ABSTRACT ALGEBRA

In this section we address some basic notation of abstract algebra, which will be useful in Chapter 3. Note that, in this chapter, we will push aside the convention of using letters A, B and C for literals, and we will use them for representing sets.

2.2.1 Notation

A *family* $(a_s)_{s \in S}$ of elements of a set A is a function $f : S \longrightarrow A$ such that $f(s) = a_s$. We write $(a_s)_{s \in S} \subseteq A$ to mean that $(a_s)_{s \in S}$ is a family of elements of A, we write $(a_s)_{s \in S} = A$ to mean that A is the image of $(a_s)_{s \in S}$ and we write $(a_s)_{s=1}^{n}$ to mean that $(a_s)_{s \in \{ 1,\dots,n \}}$.

The *Cartesian product* $\times_{s \in S} A_s$ of the family $(A_s)_{s \in S}$ is the set of all functions $f : S \longrightarrow \bigcup_{s \in S} A_s$ such that $f(s) \in A_s$ for all $s \in S$. A *projection* is a function $p_t : \times_{s \in S} A_s \longrightarrow A_t$ such that $p_t(f) = f(t)$ where $t \in S$. When $A_s = A$ for all $s \in S$ we write A^S. As usual we denote $\times_{s \in S} A_s$ by $A_1 \times \ldots \times A_n$ when S is finite and $|S| = n$ and by A^n when furthermore $A_s = A$ for all $s \in S$.

The *restriction* of a function $f : A \longrightarrow B$ to a subset $A' \subseteq A$ is a function $f_{|A'} : A' \longrightarrow B$ such that $f_{|A'}(a) = f(a)$ for all $a \in A'$. Furthermore, for a superset $A'' \supseteq A$, a function $g : A'' \longrightarrow A$ such that the restriction of g to A is f, in symbols $g_{|A} = f$, is called an *extension* of f.

A binary relation $\leq$ on a set P is called a *partial order* if it is:

- reflexive, i.e. $a \leq a$,
- transitive, i.e. $a \leq b$ and $b \leq c$ implies $a \leq c$, and
- anti-symmetric, i.e. $a \leq b$ and $b \leq a$ implies $a = b$.

A *partial order set* $\langle P, \leq \rangle$ is a set P together with a partial order relation $\leq$. Given a subset $S \subseteq P$, an element $a \in P$ is an *upper bound* of S if $s \leq a$ for all $s \in S$, a is furthermore the *least upper bound* of S $a \leq b$ iff for any upper bound b. The least upper bound, when exists, of a set S is denoted by $\sum S$. Respectively, a *lower bound* of S is an element a s.t. $a \leq s$ for all $s \in S$ and the *greatest lower bound* of S is an element a s.t. $b \leq a$ for all lower bound b, and, when it exists, it is denoted by $\prod S$. If $S = \{a_1, a_2, \dots, a_n\}$ is finite we write $a_1 + a_2 + \ldots + a_n$ (resp. $a_1 * a_2 * \ldots * a_n$) instead of $\sum S$ (resp. $\prod S$).

For a set S, an S-ary *operation* (resp. *partial operation*) on a set A is a function (resp. partial function) of the form $\star : A^S \longrightarrow A$. When $S = \{1,2\}$, it is called a *binary operation* and we usually write $a_1 \star a_2$ instead of $\star(a_s)_{s \in \{1,2\}}$. An algebraic structure or *algebra* $\mathfrak{A} = \langle A, \star_1, \star_2, \dots, \star_m \rangle$ is a set A with one or more operations $\star_1, \star_2, \dots, \star_m$ defined in it. Two algebras $\mathfrak{A}$ and $\mathfrak{B}$ belong to the same *class* if they have the same operations. A function $\epsilon : A \longrightarrow B$ such that $\epsilon\big(\star_i \, (a_s)_{s \in S_i} \big) = \star_i \big(\epsilon(a_s) \big)_{s \in S_i}$ for all operations $\star_i$ with $i \in \{1, \dots, m\}$ is called an *homomorphism*. An injective homomorphism is an *embedding* or *monomorphism*, and a surjective embedding is an *isomorphism*.

Given two algebras $\mathfrak{A} = \langle A, \star_1, \star_2, \dots, \star_m \rangle$, and $\mathfrak{B} = \langle B, \star_1', \star_2', \dots, \star_m' \rangle$ such that $B \subseteq A$, we say that $\mathfrak{B}$ is a *subalgebra* of $\mathfrak{A}$ if $\star_i'$ is the restriction of $\star_i$ to B and

$\star_i\big((b_s)_{s\in S_i}\big) \in B$ for any family $(b_s)_{s\in S_i} \in B^{S_i}$ and any S_i-ary operation $\star_i$. When clear by the context we write $\star_i$ instead of $\star_{i|B}$ and $\mathfrak{B} = \langle B, \star_1, \star_2, \dots, \star_m\rangle$ instead of $\mathfrak{B} = \langle B, \star_{1|B}, \star_{2|B}, \dots, \star_{m|B}\rangle$. The smallest subalgebra, when it exists, of $\mathfrak{A}$ that contains a set B is called the *subalgebra of* $\mathfrak{A}$ *generated by* B and is denoted by $[B]$. An algebra $\mathfrak{A}$, with domain A, is *free* if there exists a set B such that

- $\mathfrak{A}$ is generated by B, i.e. $[B] = \mathfrak{A}$
- for any algebra $\mathfrak{C}$, with domain C, of the same class as $\mathfrak{A}$ and any function $f : B \longrightarrow C$ there exists an homomorphism $h : A \longrightarrow C$ such that $h_{|B} = f$.

2.2.2 Partial Lattices

In the following we introduce a generalization of the concept of lattice called *partial lattice* from Stumme [1997].

Definition 2.6 (Partial Lattice). *A* (weak) partial (complete) lattice $\langle P, \leq, +, *\rangle$ *is a partially ordered set* $\langle P, \leq\rangle$ *together with two families of operations* $+_n$ *and* $*_n$ *such that either* $+_n(a_1, \dots, a_n)$ *is the least upper bound (resp.* $*_n(a_1, \dots, a_n)$ *is the greatest lower bound) of* $\{a_1, \dots, a_n\}$ *or it is undefined for any positive integer* $n \geq 1$ *and all elements* $a_1, \dots, a_n$ *of* P. □

As usual, given a subset $A = \{a_1, \dots, a_n\} \subseteq P$ we respectively denote by $\sum A$ and $\prod A$ the least upper bound and the greatest lower bound of A, that is $\sum A = +_n(a_1, \dots, a_n)$ (resp. $\prod A = *_n(a_1, \dots, a_n)$). Partial lattices are a generalisation of lattices where some least upper bound or greatest lower bound may be undefined. For instance, every (possible incomplete) lattice or semilattice are partial lattices.

Definition 2.7 (Upper semilattice). *A partial lattice* $\langle P, \leq, +, *\rangle$ *s.t.* $+_2(a_1, a_2)$ *is defined for all* a_1 *and* a_2 *in* P *is an* upper *(or* join*)* semilattice. *An upper semilattice is said to be* bound *iff there exists an element* 1 *such that* $a \leq 1$ *for all* $a \in P$. □

Definition 2.8 (Down semilattice). *A partial lattice* $\langle P, \leq, +, *\rangle$ *s.t.* $*_2(a_1, a_2)$ *is defined for all* a_1 *and* a_2 *in* P *is a* down *(or* meet*)* semilattice. *A down semilattice is said to be* bound *iff there exists an element* 0 *such that* $0 \leq a$ *for all* $a \in P$. □

Definition 2.9 (Lattice). *A partial lattice* $\langle P, \leq, +, *\rangle$ *which is an upper and a down semilattice is a* lattice. *A lattice is said to be* bound *iff it is a bound upper and down semilattice. A lattice is said to be* complete *iff* $+_n(a_1, \dots, a_n)$ *and* $*_n(a_1, \dots, a_n)$ *are defined for every (possible infinite) subset* $\{a_1, \dots, a_n\} \subseteq P$. □

Definition 2.10 (Filter). *An* upperset *or* order filter *of a partially ordered set $\langle P, \leq \rangle$ is a subset $F \subseteq P$ with $a \in F$ and $a \leq b$ implying $b \in F$. The set of all order filters of $\langle P, \leq \rangle$ is denoted by $\mathcal{F}_{\langle P, \leq \rangle}$. A* filter *of a partial lattice $\langle P, \leq, \sum, \prod \rangle$ is an order filter F which is closed under the defined greatest upper bound, that is, if $A \subseteq F$ and $\prod A$ is defined, then $\prod A \in F$. The filter generated by A is denoted by $\Uparrow A$. When $A = \{a\}$, we write $\uparrow a$ instead of $\Uparrow A$ and $\uparrow a$ is called a principal filter. The set of all filters and all principal filters are respectively denoted by $\mathcal{F}_P$ and $\mathcal{PF}_P$.* □

The definition of ideals is dual:

Definition 2.11 (Ideal). *A* downset *or* order ideal *of a partially ordered set $\langle P, \leq \rangle$ is a subset $I \subseteq P$ with $b \in I$ and $a \leq b$ implying $a \in I$. The set of all order ideals of $\langle P, \leq \rangle$ is denoted by $\mathcal{I}_{\langle P, \leq \rangle}$. An* ideal *of a partial lattice $\langle P, \leq, \sum, \prod \rangle$ is an order ideal I which is closed under the defined least upper bound, that is, if $A \subseteq I$ and $\sum A$ is defined, then $\sum A \in I$. The ideal generated by A is denoted by $\Downarrow A$. When $A = \{a\}$, we write $\downarrow a$ instead of $\Downarrow A$ and $\downarrow a$ is called a principal filter. The set of all ideals and all principal ideals are respectively denoted by $\mathcal{I}_P$ and $\mathcal{PI}_P$.* □

Given a set of ideals $\mathbf{I} \subseteq \mathcal{I}_P$ and a set of filters $\mathbf{F} \subseteq \mathcal{F}_P$, $\mathbf{I}^F$ and $\mathbf{F}^I$ respectively denote the following sets of filters and ideals:

$$\mathbf{I}^F = \{\, F \in \mathcal{F}_P \mid \forall I \in \mathbf{I} : F \cap I \neq \emptyset \,\}$$
$$\mathbf{F}^I = \{\, I \in \mathcal{I}_P \mid \forall F \in \mathbf{F} : F \cap I \neq \emptyset \,\}$$

Definition 2.12 (Concept lattice of a partial lattice). *Let $\langle P, \leq, +, * \rangle$ be a partial lattice. Then, its* concept lattice *$\mathbf{B}_P$ is a complete lattice $\langle B_P, \leq, +, * \rangle$ where the elements of B_P are pairs of the form $\langle \mathbf{I}^F, \mathbf{I} \rangle$ with $\mathbf{I} \subseteq \mathcal{I}_P$ and $\mathbf{I}^{FI} = \mathbf{I}$ and meet and join are respectively defined as:*

$$\prod_{s \in S} (\mathbf{I}_s^F, \mathbf{I}_s) = \Big(\bigcap_{s \in S} \mathbf{I}_s^F, \Big(\bigcup_{s \in S} \mathbf{I}_s \Big)^{FI} \Big)$$
$$\sum_{s \in S} (\mathbf{I}_s^F, \mathbf{I}_s) = \Big(\Big(\bigcup_{s \in S} \mathbf{I}_s^I \Big)^{IF}, \bigcap_{s \in S} \mathbf{I}_s \Big)$$

Furthermore, $(\mathbf{I}_1^F, \mathbf{I}_1) \leq (\mathbf{I}_2^F, \mathbf{I}_2)$ *iff* $\mathbf{I}_1^F \subseteq \mathbf{I}_2^F$ *iff* $\mathbf{I}_1 \supseteq \mathbf{I}_2$. □

Theorem 2.2 (Stumme [1997, Theorem 9]). *Given a partial lattice $\langle P, \leq, +, * \rangle$, its concept lattice $\mathbf{B}_P = \langle B_P, \leq, +, * \rangle$ is the free completely distributive (complete) lattice generated by the image of ϵ_P where $\epsilon_P : P \longrightarrow B_P$ is given by:*

$$a \mapsto \langle \{\, F \in \mathcal{F}_P \mid a \in F \,\}, \{\, I \in \mathcal{I}_P \mid a \in I \,\} \rangle \tag{45}$$

That is, for each order-preserving map δ from $\langle P, \leq \rangle$ to a completely distributive lattice $\mathbf{S}$, there exists a homomorphism $h : B_P \longrightarrow \mathbf{S}$ such that $\delta = h \circ \epsilon_P$. □

2.3 GENERAL ANNOTATED LOGIC PROGRAMS

General Annotated Logic Programs (GAP) were introduced by Kifer and Subrahmanian [1992] as a formal semantics generalising various result and treatments of multivalued logic programming. In GAP, truth values $\mathbf{V}$ are assumed to form a bounded upper (but possible incomplete) semi-lattice with a least upper bound operator $(+)$ and top element 1.

For each positive integer $s \geq 1$, we assume there is a family F_s of total continuous (hence monotonic) functions of the type $\mathbf{V}^s \longrightarrow \mathbf{V}$ called *annotation functions*. Furthermore, F denotes the set of all functions, that is $F = \bigcup_{s \geq 1} F_s$, and all functions $f \in F$ are considered to be computable in the sense that there exists a uniform procedure $\tilde{f}$ such that if f is n-ary and $\mu_1, \ldots, \mu_s$ are given as input to $\tilde{f}$, then $f(\mu_1, \ldots, \mu_s)$ is computed by $\tilde{f}$ in a finite amount of time. Moreover, each F_s is assumed to contain an s-ary function $+_s$ derived from the semi-lattice operator $+$, which, given inputs $\mu_1, \ldots, \mu_s$ returns the least upper bound of $\{\mu_1, \ldots, \mu_s\}$. As done in Section 2.2.1 we will just denote by $+$ any of the $+_s$ functions and by $\sum\{\mu_1, \ldots, \mu_s\}$ the least upper bound of $\{\mu_1, \ldots, \mu_s\}$. Apart from the interpreted annotation functions, the language contains usual uninterpreted functions, constants and predicate symbols as those introduced in Section 2.1.4. Two disjoint sets of variables are considered — *object variables* and *annotation variables*.

Definition 2.13 (Annotation). *Given a family of annotation functions $F_1, F_2, \ldots$, a set of truth values $\mathbf{V}$ and a set of atoms At:*

- *An* annotation *is either an element of* $\mathbf{V}$, *a* annotation variable *or an* complex annotation term *of the form $f(\mu_1, \ldots, \mu_m)$ where f is an m-ary annotation function $f \in F_n$ and $\mu_1, \ldots, \mu_m$ are annotations.*
- *An* annotated atom *is a formula of the form $(A : \mu)$ where $A \in At$ is an atom and μ is an annotation. An annotated atom $(A : \mu)$ is said to be* c-annotated, v-annotated *and* t-annotated *if μ is respectively an element of* $\mathbf{V}$, *an annotation variable or a complex annotation term.* □

Definition 2.14 (Annotated clause). *Given a GAP signature, an* annotated clause *is a formula of the form:*

$$(A : \rho) \leftarrow (B_1 : \mu_1) \ \& \ \ldots \ \& \ (B_m : \mu_m)$$

where $(A : \rho)$ is an annotated atom and $(B_1 : \mu_1)$, ..., $(B_m : \mu_m)$ are c-annotated or v-annotated atoms. Any set of annotated clauses is also called a GAP. □

Intuitively, the above definition asserts that members of F may occur in the annotation in the head of a clause, but not in its body. A strictly ground instance of a clause is obtained by replacing all the object variables and annotated variables by object constants and c-annotations, respectively. Since all annotated functions $f \in F$ are evaluable, annotation terms of the form $f(\mu_1, \ldots, \mu_m)$ where $\mu_1, \ldots, \mu_m$ are values in $\mathbf{V}$ and $f \in F_m$ are also considered to be ground and are identified with the result of the computation of $\tilde{f}$ on $(\mu_1, \ldots, \mu_m)$.

An r-interpretation $I : At \longrightarrow \mathbf{V}$ is a mapping from ground atoms into the set of ideals of the values set $\mathbf{V}$. Since r-interpretations are functions into a partially order set $\mathbf{V}$, the order relation may be extended over interpretations such that $I \leq J$ iff $I(A) \leq J(A)$ for all $A \in At$.

Definition 2.15 (r-Satisfaction)**.** *Given an r-interpretation $I : At \longrightarrow \mathbf{V}$, a ground atom "a" and a c-annotation $\mu \in \mathbf{V}$, then*

1. $I \models_r (A : \mu)$ *iff* $\mu \leq I(A)$
2. $I \models_r \varphi_1 \,\&\, \varphi_2$ *iff* $I \models_r \varphi_1$ *and* $I \models_r \varphi_2$
3. $I \models_r \varphi_1 \leftarrow \varphi_2$ *iff* $I \models_r \varphi_1$ *or* $I \not\models_r \varphi_2$

A r-interpretation I is said to be an r-model *of a formula φ iff $I \models_r \varphi$. Furthermore, it is said to be a model of a GAP P iff it is a model of all the formulas in P.* □

Definition 2.16 (r-Direct consequences operator)**.** *Given a GAP program P, an r-interpretation I and an atom A, the r-direct consequence operator $R_P(I)(A)$ is given by:*

$$\sum \{\, f(\mu_1, \ldots, \mu_m) \mid (A : f(\mu_1, \ldots, \mu_m)) \leftarrow (B_1 : \mu_1) \,\&\, \ldots \,\&\, (B_m : \mu_m) \text{ in } P \,\}$$

for any r-interpretation I. □

The following results assert when the direct consequences operator R_P is well behaved.

Theorem 2.3 (Kifer and Subrahmanian [1992, Theorem 1])**.** *For any GAP program P and r-interpretation I_r, it holds that*

i) *I_r is a model of P iff $R_P(I_r) \leq I_r$, and*

ii) *R_P is monotonic.* □

Theorem 2.4 (Kifer and Li [1988])**.** *Any GAP program P with only v-annotation in rule bodies satisfies that*

i) *R_P is continuous,*

ii) *$R_P^\omega = \mathtt{lfp}(R_P)$ is the least model of P.* □

2.4 COMPLEXITY: THE POLYNOMIAL HIERARCHY

Complexity Theory is an approach to classify computational problems by their requirements on computer resources. In this section, we briefly summarise the basic notions that we will require in order to classify the problems we will study in this dissertation. For wider view of Complexity Theory, complexity classes and complete problems in Polynomial Hierarchy see Papadimitriou [2003], Johnson [1990] and Schaefer and Umans [2002].

A *decision problem* consists in deciding whether a given input w satisfies a certain property Q. That is, in set theoretic terms, whether it belongs to the set $S = \{w \mid Q(w)\}$ of the elements that satisfy the property Q. The *complementary* of a decision problem consists in deciding whether a given input w does not satisfy a certain property Q, that is, it belongs to the set $S = \{w \mid \neg Q(w)\}$.

Decision problems are classified into *complexity classes* by bounding some computer resources, particularly time and space are the most widely studied.

- DTIME($f(n)$) denotes the class of problems decidable by a deterministic Turing machine in almost $f(n)$ steps where n is the size of the input.
- DSPACE($f(n)$) denotes the class of problems decidable by a deterministic Turing machine using almost $f(n)$ space where n is the size of the input.

A problem is said to be decidable in polynomial time (resp. in polynomial size) iff it belong to the class DTIME(n^i) (resp. DSPACE(n^i)) for some non-negative integer i. The class of problems decidable in polynomial time (resp. polynomial size) is denoted by P (resp. by PSPACE). Similarly, a problem is said to be decidable in exponential time (resp. in exponential size) iff it belong to the class DTIME(2^{n^i}) (resp. DSPACE(2^{n^i})) for some non-negative integer i. The class of problems decidable in exponential time (resp. exponential size) is denoted by EXPTIME (resp. by EXPSPACE). A problem is said to be decidable in logarithmic time (resp. in logarithmic size) iff it belong to the class DTIME($i \cdot \log(n)$) (resp. DSPACE($i \cdot \log(n)$)) for some non-negative integer i. The class of problems decidable in logarithmic space is denoted by LOG.

Furthermore, NTIME($f(n)$) and NSPACE($f(n)$) respectively denote the classes of problems decidable by a non-deterministic Turning machine in almost $f(n)$ steps and space. NP and NEXPTIME are respectively the classes of problems decidable in polynomial and exponential time by a non-deterministic

Turing machine. It is worth to mention that the classes NPSPACE and NEXPSPACE respectively coincide with PSPAPCE and EXPSPACE, that is, adding non-determinism to the computational model does not make a difference in these complexity classes.

It is known that every problem decidable in logarithmic space is also decidable in polynomial time and so on:

$$\mathrm{LOG} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXPTIME} \subseteq \mathrm{EXPSPACE}$$

Furthermore it is known that some of these inclusions are strict, $\mathrm{P} \subset \mathrm{EXPTIME}$ and $\mathrm{LOG} \subset \mathrm{PSPACE}$. Other are believed to be different, like $\mathrm{P} \subset \mathrm{NP}$ but remain unproven.

Given any complexity classes C and O, the class C^{O} is the class of problems decidable in the class C with the possibility of calling in each step a subroutine, called *oracle*, capable of solving problems of class O "for free." The Polynomial Hierarchy (PH) is recursively defined from the classes P and NP in the following way: $\Sigma_0^{\mathrm{P}} = \Delta_0^{\mathrm{P}} = \Pi_0^{\mathrm{P}} = \mathrm{P}$ and

$$\begin{aligned}
\Delta_{i+1}^{\mathrm{P}} &\stackrel{\text{def}}{=} \mathrm{P}^{\Sigma_i^{\mathrm{P}}} \\
\Sigma_{i+1}^{\mathrm{P}} &\stackrel{\text{def}}{=} \mathrm{NP}^{\Sigma_i^{\mathrm{P}}} \\
\Pi_{i+1}^{\mathrm{P}} &\stackrel{\text{def}}{=} \mathrm{coNP}^{\Sigma_i^{\mathrm{P}}}
\end{aligned}$$

where coNP is the complementary of the class NP. The PH class is defined as the class of problems in Σ_i^{P} for some non-negative integer i. The inclusion in the direction of the arrow holds between the classes depicted in Figure 2 Furthermore, $\mathrm{P} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$ and, although it is believed that these inclusions are strict, it is still unknown whether this is the case or not.

Given a pair of problems A and B, it is said that A *reduces* to B when there exists a transformation g that for every input w of A, produces an input $g(w)$ of B which has the same answer than w. A transformation g that reduces a problem A to B is a C-*reduction* if reduces A to B and can be computed within the bounds of the complexity class C.

Reductions play a central role in the definition of *complete problems*. A problem A in a complexity class C is said to be *complete* under C′-reductions iff for every problem B in C there exists a C′-reduction that maps problem instances

$$\begin{tikzcd}
 & \mathrm{PH} & \\
 & \Delta^{\mathrm{P}}_{i+1} \arrow[u, dotted] & \\
\Sigma^{\mathrm{P}}_{i} \arrow[ur] & & \Pi^{\mathrm{P}}_{i} \arrow[ul] \\
 & \Delta^{\mathrm{P}}_{i} \arrow[ul] \arrow[ur] & \\
 & \Delta^{\mathrm{P}}_{2} \arrow[u, dotted] & \\
\mathrm{NP} \arrow[ur] & & \mathrm{coNP} \arrow[ul] \\
 & \mathrm{P} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Figure 2: Relations inside the Polynomial Hierarchy.

of B into A. In particular, we will be interested in complete problems under *logspace* and *polynomial time* reductions.

Since we will be dealing with problems in the PH, in the following we enumerate some problems that are known to be complete for some of the classes in the PH.

- Given a positive logic program P, deciding whether $P \models A$ for some literal A is P-complete [Dantsin et al., 2001].
- Deciding whether a propositional theory T is satisfiable in classical logic, that is, whether there exists a model M of T, $M \models T$, is NP-complete [Cook, 1971, Levin, 1973].
- Deciding whether a propositional theory T is unsatisfiable in classical logic, that is, whether there does not exist a model M of T, $M \models T$, is coNP-complete.

- Deciding whether a quantified Boolean formula of the form

  $$\exists x_{11} \dots x_{1n_1} \forall x_{21} \dots x_{2n_2} Q x_{m1} \dots x_{mn_m} \dots \varphi(x_{11}, \dots, x_{1n_1}, \dots, x_{mn_m})$$

  where "Q" stands for "$\exists$" if m is odd and for "$\forall$" if m is even and φ is a Boolean formula with variables $x_{11}, \dots, x_{1n_1}, \dots, x_{mn_m}$, is Σ_m^P-complete. It is complete even if φ is a 3-CNF formula, that is φ is in disjunction normal form with only 3 literals per conjunction and m is odd or if φ is a 3-DNF formula, that is φ is in disjunctive normal form with only 3 literals per conjunction and m is even [Wrathall, 1976].

Furthermore, the following problems in LP are known to be complete for some class in the PH:

- Given a ground, logic program P, deciding whether $P \models_{wf} A$ under the well-founded semantics is P-complete [Van Gelder, 1989, Van Gelder et al., 1991, Eiter and Gottlob, 1995].

- Given a ground logic program P, deciding whether P has a stable model is NP-complete [Marek and Truszczynski, 1991].

- Given a ground logic program P, deciding whether $P \models A$ under the stable model semantics for some literal A is coNP-complete [Marek and Truszczynski, 1991, Schlipf, 1995, Kolaitis and Papadimitriou, 1991].

- Given a ground disjunctive logic program P, deciding whether P has a stable model is Σ_2^P-complete [Eiter and Gottlob, 1995].

- Given a ground disjunctive logic program P, deciding whether $P \models A$ under the stable model semantics for some literal A is Π_2^P-complete [Eiter and Gottlob, 1995].

3 SUFFICIENT CAUSES

As commented in the introduction, this dissertation will be focused on the study of *sufficient causation*. Therefore, we will use *cause* as a shorthand for *sufficient cause* and "A has caused B" as a shorthand for "A has been sufficient to cause B." We have also commented that a central goal of this dissertation is representing statements like (6) without falling down in problems of *elaboration tolerance*. For instance, we have seen that by formulas (7) and (9) we respectively conclude that $hascaused(up(a), open)$ and $hascaused(open, bomb)$ hold. For concluding that $hascaused(up(a), bomb)$, we may rely on Lewis' idea of causal chain: if $up(a)$ has caused $open$ and $open$ has caused $bomb$ then $up(a) \cdot open \cdot bomb$ is a causal chain that explains $bomb$. Similarly, $up(b) \cdot open \cdot bomb$ is also a causal chain of $bomb$. It is also worth to recall that the idea of causal chains presents some troubles in distinguish between alternative and joint causes. It is clear that rule (14) has a completely different meaning than rules (19) and (20), but they lead to the same causal chains. With the aim of differentiating between both, alternative an joint causes, we combine Lewis' idea of *causal chains* with Mackie's *minimal disjunctive normal form*, so that alternative causes will be represented as:

$$up(a) \cdot open \cdot bomb \ + \ up(b) \cdot open \cdot bomb \tag{46}$$

while joint causes will be represented as:

$$up(a) \cdot open \cdot bomb \ * \ up(b) \cdot open \cdot bomb \tag{47}$$

Now it is clear that both $up(a)$ and $up(b)$ form sufficient and alternative causes of $bomb$ with respect to (46), while they are joint causes of it with respect to (47).

Cabalar [2012] has proposed labelling program rules, representing causes as graphs whose vertices are those rule labels, and then, using these graphs as the basis for the "truth values" of a multivalued semantics. We will follow here this approach. In Chapter 6, we will use these "truth values" for defining a precise semantics for the predicate $hascaused(A, B)$ as a particular case of a kind of new literal.

Definition 3.1 (Labelled program). *Given a signature $\langle At, Lb \rangle$ where At and Lb respectively represent a set of atoms and a set of labels, a* labelled rule *R is an expression of the form*

$$l_R : \; A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_n \tag{48}$$

where l_R is either a label, that is $l_R \in Lb$, or the special symbol $l_R = 1$, and where A and all B_i's and C_j's' are atoms. A labelled program *P is a set of labelled rules.* □

An unlabelled rule stands for an abbreviation of a rule with $l_R = 1$. We will see later that 1 corresponds, in fact, to the empty graph. We say that a program is *uniquely labelled* when every rule has a different label, and that it is *completely labelled* if furthermore it has not unlabelled rules. A program is *head labelled* if rules with different heads have different labels, and it is *completely head labelled* if furthermore it has not unlabelled rules. We will focus on normal programs, that is, programs without disjunction in the head. To illustrate these ideas, consider the scenario of Example 1.4 represented by the following labelled program.

Program 3.1.

$$\begin{array}{lllll}
b & : & bomb & \leftarrow & open \\
o & : & open & \leftarrow & up(a), up(b) \\
k & : & open & \leftarrow & key \\
u(L) & : & up(L) & \leftarrow & lift(L)
\end{array}
\qquad
\begin{array}{lll}
lift(a) & : & lift(a) \\
lift(b) & : & lift(b) \\
key & : & key
\end{array}$$

We will usually label facts with a label with the same name as the fact. For the readability sake, we will use label '\$:' for indicating that the label of a rule has the same name than its head. That is, the following facts are equivalent to the ones in Program 3.1.

$$\begin{array}{ll}
\$: & lift(a) \\
\$: & lift(b) \\
\$: & key
\end{array}$$

Figure 3 depicts two graphs capturing the two causal chains that have caused the *bomb* explosion.

$$\begin{tikzcd}
lift(a) \arrow[d] & & lift(b) \arrow[d] & & key \arrow[dd] \\
u(a) \arrow[dr, bend right] & & u(b) \arrow[dl, bend left] & & \\
& o \arrow[d] & & & k \arrow[d] \\
& b & & & b \\
& G_1 & & & G_2
\end{tikzcd}$$

Figure 3: Graphs G_1 and G_2 associated to the proofs of *bomb* in Example 1.2.

Graph G_1 corresponds to the first cause initiated by the action of lifting both locks. Graph G_1 contains two causal chains, one initiated by the lift of the first lock $lift(a) \cdot u(a) \cdot o \cdot b$, and the second one initiated, in its turn, by the second lifting $lift(b) \cdot u(b) \cdot o \cdot b$. On the other hand, graph G_2 corresponds to the second cause initiated by the action of turning the *key*, which represents itself a causal chain of the form $key \cdot k \cdot b$. The causes of *bomb* may be then captured by the following formula in minimal disjunctive normal form.

$$lift(a) \cdot u(a) \cdot o \cdot b \; * \; lift(b) \cdot u(b) \cdot o \cdot b \; + \; key \cdot k \cdot b \tag{49}$$

Each subterm without sums $(+)$ represents a cause for the fact *bomb*[1].

3.1 CAUSES AS GRAPHS

In this section, we study sufficient causes as true graphs, rather than just products of causal chains. This possess the issue of how to define the minimality criterion to guarantee that causes do not contain unnecessary events for being sufficient. Notice that we differentiate between a *sufficient explanation* (an explanation that contains enough events to get the effect) and being a *sufficient cause* which we assume that, in addition to contain enough events to get the effect, it does not contain superfluous ones. When causes are represented as a product of causal chains, that is a set of chains, the minimality criterion was basically subset inclusion. Hence, we may think now that the minimality criterion

1 We have preliminary explored this approach in Cabalar and Fandinno [2013]. However, a subsequent complexity analysis revealed that identifying sufficient causes with this representation was surprisingly harder than intuitively expected. This fact has led us to the refinement presented in this dissertation.

should coincide with subgraph minimality, and this will be, in fact, the basis of our definition. However, taking this idea in a naive manner produces some troubles, which we illustrate with the following example taken from Cabalar et al. [2014b].

Example 3.1 (Alarm circuit). *An alarm is connected to three switches as depicted in Figure 4. Each switch is operated by a different person and, at a given moment, they all accidentally close the switches. We want to analyse the responsibility for firing a false alarm.* □

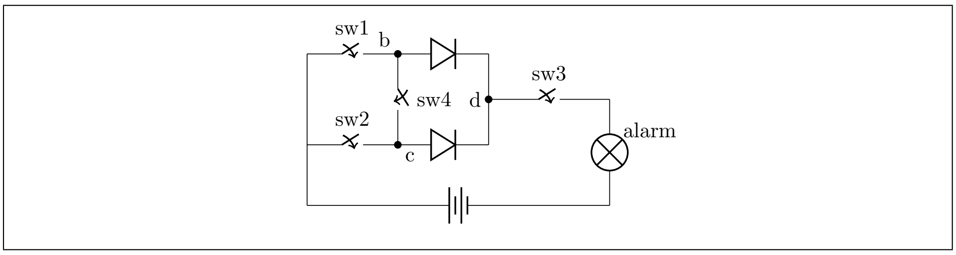

Figure 4: A circuit connecting four switches and an alarm.

A possible representation of this scenario could be as follows:

Program 3.2.

$$
\begin{array}{llllll}
a: & alarm & \leftarrow sw_3, current(d) & \quad & \$: & sw_1 \\
b: & current(b) & \leftarrow sw_1 & & \$: & sw_2 \\
b: & current(b) & \leftarrow sw_4, current(c) & & \$: & sw_3 \\
c: & current(c) & \leftarrow sw_2 & & \$: & sw_4 \\
c: & current(c) & \leftarrow sw_4, current(b) & & & \\
d: & current(d) & \leftarrow current(b) & & & \\
d: & current(d) & \leftarrow current(c) & & &
\end{array}
$$

In this program, we have used a same label for all rules defining the current at some circuit point. This reflects the idea that they are part of a same component: points in the circuit where two wires join represent OR gates. For instance, rules with label d may be read as a logical formula of the form:

$$d: \quad current(d) \leftarrow current(b) \ \vee \ current(c)$$

Figure 5 depicts three graphs that one could cite among the explanations of *alarm*. The sufficient causes of *alarm* will be those explanations which do not

$$\begin{tikzcd}
sw_1 \arrow[d] & & sw_3 \arrow[dddl, bend left] \\
b \arrow[dd] & & \\
 & & \\
d \arrow[r, bend right] & a & \\
 & G_1 &
\end{tikzcd}
\qquad
\begin{tikzcd}
sw_2 \arrow[d] & & sw_3 \arrow[dddl, bend left] \\
c \arrow[dd] & & \\
 & & \\
d \arrow[r, bend right] & a & \\
 & G_2 &
\end{tikzcd}
\qquad
\begin{tikzcd}
sw_2 \arrow[d] & sw_4 \arrow[dl, bend left] & sw_3 \arrow[dddl, bend left] \\
c \arrow[d] & & \\
b \arrow[d] & & \\
d \arrow[r, bend right] & a & \\
 & G_3 &
\end{tikzcd}$$

Figure 5: Sufficient explanations for the alarm firing in Example 3.1.

contain redundant events (those unnecessary for being sufficient). Notice that, neither switch 4 nor the current going through b are actually necessary for G_2 to be sufficient for *alarm*. Furthermore, graph G_3 is like G_2 but with vertex b placed between vertices c and d. The edges $c \to b \to d$ just show the existence of a redundant path between c and d which depends on closing switch 4. Hence, it seems clear that G_3 is redundant with respect to G_2, and thus it should not be a sufficient cause of *alarm*. Despite of that, neither graph G_1 nor G_2 are subgraphs of G_3, and thus G_3 is subgraph minimal. Indeed, G_2 is not a subgraph of G_3 because edge $c \to d$ does not belong to graph G_3, although, it belongs to its transitive closure. This fact points out that, in order to compare causes in programs where some rules share the same label, we should appeal to the edges of their transitive closures.

Furthermore, it is also worth to represent causes as reflexively closed graphs. This will allow ignoring the number of steps that a recursive definition has been applied. For instance, the natural numbers can be defined with the set of rules:

$$z: \; nat(0) \qquad\qquad s: \; nat(X+1) \leftarrow nat(X)$$

where the last rule is an abbreviation for all rules obtained by replacing X for all natural number n. The graph obtained from the proof of atom $nat(n)$ will be of the form:

$$z \overbrace{\to s \to \ldots \to s}^{n}$$

Note that this graph is just the same as $z \to s \to s$ since actually s always corresponds to the same vertex. By assuming that causal graphs are reflexively closed, graph $z \rightarrow s$ and graph $z \rightarrow s \rightarrow s$ actually stand for the same causal graph depicted in Figure 6. If we want to keep track of each of the applica-

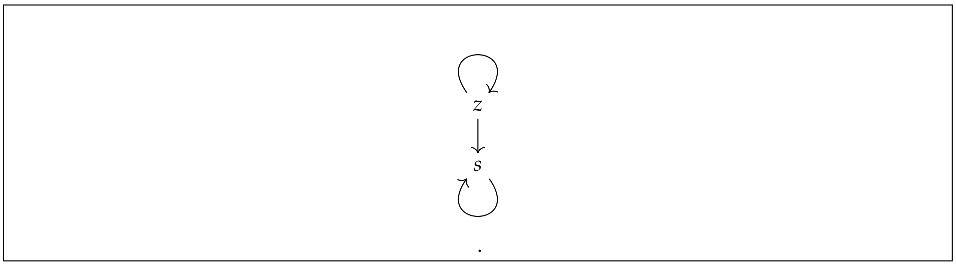

Figure 6: Causal graph corresponding to graphs $z \rightarrow s$ and $z \rightarrow s \rightarrow s$.

tions of the recursive definition, it is only necessary to change label s to include variable X, writing $s(X)$. In this case, we will obtain $z \rightarrow s(0) \rightarrow \ldots \rightarrow s(n)$. A fortunate side effect of reflexivity is that every vertex has at least one edge (the reflexive one) which allows treating graphs just as sets of edges, and the subgraph relation to be just the subset $\subseteq$ relation.

Definition 3.2 (Causal graph). *Given a set of labels Lb, a* causal graph (c-graph) *$G \subseteq Lb \times Lb$ is set of edges transitively and reflexively closed. The set of causal graphs is denoted by $\mathbf{C}_{Lb}$. A c-graph is said to be* cyclic *if contains a (non-reflexive) cycle,* acyclic *otherwise.* □

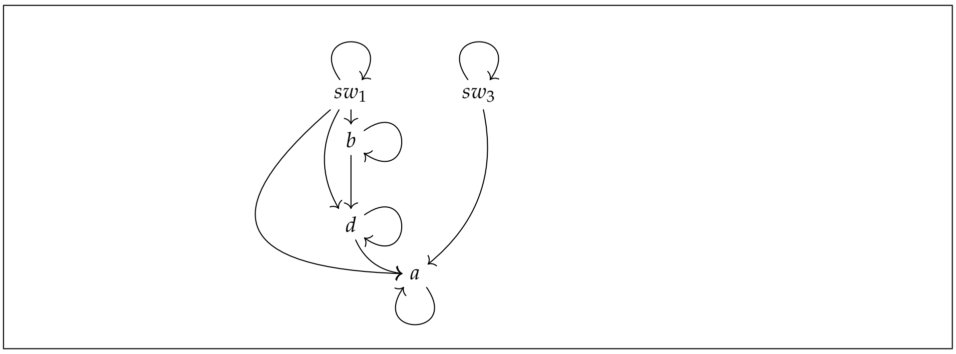

Figure 7: Causal graph corresponding to graph G_1 in Figure 5.

Although, for mathematical treatment, it is worth to define causal graphs as transitively and reflexively closed structures, for readability sake, it is clearer to depict them as their *transitive and reflexive reductions*[2]. In this way, for instance graph G_1 in Figure 5 will implicitly stand for that in Figure 7.

3.2 CAUSAL GRAPHS AND LOGICAL PROOFS

We proceed next to formalize the idea we had sketched in the introduction, where we considered building explanations by going from the causes to their final events passing through the causal laws that connect them. This type of explanations can be captured by proofs in their usual logical meaning.

Definition 3.3 (Proof). *Given a positive program P, a* proof $\pi(A)$ *of an atom A is recursively defined as a derivation:*

$$\pi(A) \quad \stackrel{\text{def}}{=} \quad \frac{\pi(B_1)\ \ldots\ \pi(B_m)}{A}\ R, \tag{50}$$

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

For each proof of the form (50), we say that $\pi(B_1)$, $\ldots$, $\pi(B_m)$ are the direct subproofs of $\pi(A)$. The subproofs of $\pi(A)$ are recursively defined to be itself plus the subproofs of all its direct subproofs. Figure 8 depicts one of the proofs for *bomb* in Program 3.1. It is easy to see that, by joining with an edge the label of the rule of each direct subproof with the label of its parent we will obtain graph G_1 in Figure 3.

This intuitive relation between graphs and logical proofs provides us with a new tool for formalising the notion of *minimality* or *non-redundant explanation*. Consider the following example:

Example 3.2 (Ex. 1.2 continued). *Our suitcase has now a wireless mechanism that flip both locks remotely. The wireless mechanism is activate by two different remote controls (s and y) simultaneously.* □

2 Recall from graph theory that the *transitive and reflexive reduction* of a graph G is a minimal graph whose transitive and reflexive closure the same as the closure of G.

$$\cfrac{\cfrac{\cfrac{\cfrac{\top}{lift(a)}\,lift(a)}{up(a)}\,u(a) \qquad \cfrac{\cfrac{\top}{lift(b)}\,lift(b)}{up(b)}\,u(a)}{open}\,o}{bomb}\,b$$

Figure 8: Proof for atom *bomb* corresponding to Program 3.1.

Program 3.3.

$$\begin{array}{lllll} b : & bomb & \leftarrow open & \qquad s : & wireless \\ u : & open & \leftarrow up(a), up(b) & \qquad y : & wireless \\ l(L) : & up(L) & \leftarrow wireless & & \end{array}$$

where $L \in \{a,b\}$. The two labelled facts for *wireless* stand for the two different controls that activate this mechanism. Figure 9 depicts the four $\subseteq$-minimal causal graphs corresponding to the proofs of the literal *bomb*. Notice that

$$\begin{tikzcd}
& s \arrow[dl] \arrow[dr] & \\
l(a) \arrow[dr, bend right] & & l(b) \arrow[dl, bend left] \\
& u \arrow[d] & \\
& b & \\
& G_1 &
\end{tikzcd}
\quad
\begin{tikzcd}
& y \arrow[dl] \arrow[dr] & \\
l(a) \arrow[dr, bend right] & & l(b) \arrow[dl, bend left] \\
& u \arrow[d] & \\
& b & \\
& G_2 &
\end{tikzcd}
\quad
\begin{tikzcd}
s \arrow[d] & & y \arrow[d] \\
l(a) \arrow[dr, bend right] & & l(b) \arrow[dl, bend left] \\
& u \arrow[d] & \\
& b & \\
& G_3 &
\end{tikzcd}
\quad
\begin{tikzcd}
y \arrow[d] & & s \arrow[d] \\
l(a) \arrow[dr, bend right] & & l(b) \arrow[dl, bend left] \\
& u \arrow[d] & \\
& b & \\
& G_4 &
\end{tikzcd}$$

Figure 9: Causal graphs associated to the proofs of the literal *bomb* in Example 3.2.

graph G_3 represents that the first lock has been moved up by the first wireless mechanism s whereas the second one has been lifted by the second one y. On the other hand, in graph G_1, both locks have been lifted by the same mechanism s. In this sense, the causal relations represented by graphs G_1 and G_3 are not comparable. However, the second mechanism y is not necessary to explain why the bomb has exploded and, thus, G_3 should be considered redundant with respect to G_1, i.e. G_3 should not be considered a cause of *bomb*. The same reasoning can be applied to pairs G_2-G_3, G_1-G_4 and G_2-G_4. Furthermore, note that we can represent G_1 and G_3 respectively as the proofs depicted in Figures 10 and 11. It is easy to see that subproofs of the proof depicted in

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{wireless}\,s}{up(a)}\,u(a) \qquad \dfrac{\dfrac{\top}{wireless}\,s}{up(b)}\,u(a)}{open}\,o}{bomb}\,b$$

Figure 10: Proof for atom *bomb* corresponding to graph G_1 w.r.t. Program 3.3

Figure 10 are a strictly subset of the proofs of the one depicted in Figure 11. Therefore, we may state the following definition of non-redundant proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{wireless}\,s}{up(a)}\,u(a) \qquad \dfrac{\dfrac{\top}{wireless}\,y}{up(b)}\,u(a)}{open}\,o}{bomb}\,b$$

Figure 11: Proof for atom *bomb* corresponding to graph G_3 w.r.t. Program 3.3

Definition 3.4 (Redundant proof). *Given a positive program P, a proof $\pi = \pi(A)$ is said to be* redundant *iff there exists a another proof $\pi' = \pi(A)$ of the same literal A with a strict subset of subproofs, that is, $subproofs(\pi') \subset subproofs(\pi)$. Proofs are* non-redundant *otherwise.* □

We may incorporate this notion of non-redundant proof into the graph representation by means of a function $\delta : Lit \longrightarrow Lb \cup \{1\}$. Using function δ we can assign a graph to every proof π in the following way:

Definition 3.5 (Proof graph). *Given a proof π and a function $\delta : Lit \longrightarrow Lb \cup \{1\}$, by $graph(\pi)$ we denote the graph which, for each direct subproof π' of π of the form (50), contains edges $(l_{B_1}, l_R), \ldots, (l_{B_m}, l_R), (l_R, l_A)$ where l_R is the label of rule R in subproof π' and $l_{B_1}, \ldots, l_{B_m}$ and l_A are the labels assigned by δ to the atoms $B_1, \ldots, B_m$ and A respectively. In addition, $graph(\pi)$ recursively contains $graph(\pi')$ for all direct subproof π' of π. By $cgraph(\pi)$ we denote the causal graph generated by $graph(\pi)$, that is, its transitive and reflexive closure.* □

In other words, $cgraph(\pi)$ is the transitive and reflexive closure of a graph obtained by adding an edge from the consequent of each subproof (which corresponds to an atom in the body of the rule) to the label of the proof and another edge from its label to the label of its consequent. For instance, if we assume that δ maps each atom to a different label, with the same name as the atom, and which, furthermore, is different from all the labels in the program rules, the graphs corresponding to the proofs of *bomb* become those depicted in Figure 12. It is easy to see that graphs in Figure 12 correspond somehow to

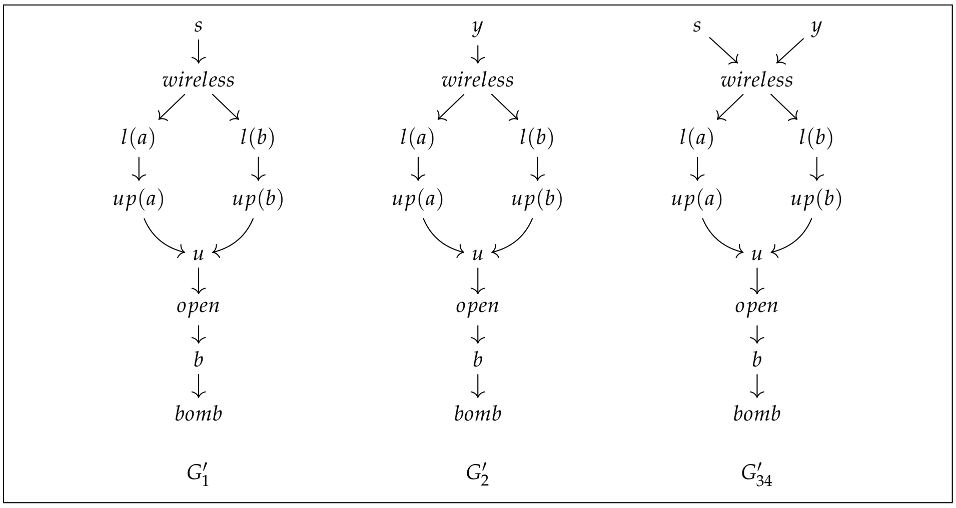

Figure 12: Causal graphs associated to the proofs of the literal *bomb* in Example 3.2 with $\delta(A)$ a different label for each atom A.

those depicted in Figure 9. In fact, graphs G_1 and G_2 in Figure 9 are the result of removing the vertices associated to atoms by the function δ, and contracting their adjacent edges, in the corresponding graphs G'_1 and G'_2 depicted in Figure 12. Graph G'_{34} corresponds to both G_3 and G_4. Note that, the result from adding the edges $(s,wireless)$, $(y,wireless)$, $(wireless,l(a))$ and $(wireless,l(b))$ to graphs G_3 and G_4 leads to the same graph. More importantly, note that G'_{34} is a strict superset of G'_1 and G'_2, and so, it is not $\subseteq$-minimal. Hence, G'_{34} will not be considered to be a cause of the fact *bomb*, whereas G_3 and G_4, which are $\subseteq$-minimal, will be. We will see later, in Chapter 4, that causal graphs in Figure 9 can be obtained in the same way as those in Figure 12, by just selecting a function δ that maps each literal A to 1.

Definition 3.6 (Atom labelling)**.** *Given a program P, we say that a function δ which assigns a different label l_A to each literal A and different from any label in P is a* unique atom labelling. *A function δ assigning* 1 *to each literal in P is an* empty atom labelling. $\square$

For most of the examples in this dissertation there will be a one-to-one correspondence between the causes obtained under a unique atom labelling and an empty one. For the sake of simplicity, in all cases that both atom labelling agree, we will choose an empty one. However, as we have seen with Example 3.2, some examples are sensitive to the atom labelling chosen. As a consequence, some formal results will depend on the assumption that δ assigns a different label to each literal, and different from any label in the program. Results that depend on this assumption will be explicitly indicated.

Definition 3.7 (Graph redundant proof)**.** *Given a positive program P, a proof $\pi = \pi(A)$ is said to be* graph redundant *iff there exists a another proof $\pi' = \pi(A)$ of the same literal A such that $cgraph(\pi) \subset cgraph(\pi')$. Proofs are* graph non-redundant *otherwise.* □

Proposition 3.1. *Given a positive completely labelled program P and a unique atom mapping $\delta : Lit \longrightarrow Lb$, a proof $\pi(A)$ of an atom A is* non-redundant *if and only if it is* graph non-redundant. □

Proof. The proof can be found in Appendix A on page 240. □

Proposition 3.1 shows that, in the particular case that the program is labelled with a different label for each rule and each literal, the concept of non-redundant proof coincides with the concept of subgraph minimality among causal graphs. In Chapter 4 we will show that, in this particular case, the causes of a literal, obtained from our semantics, are exactly the causal graphs corresponding to the non-redundant proofs of that literal. This result does not necessary hold if δ is chosen to be an empty atom labelling as can be seen in Example 1.4.

3.3 CAUSAL VALUES: REPRESENTING ALTERNATIVE CAUSES

As commented in the introduction, one of the goals of this dissertation is representing causes as manipulable expressions. In particular, we have seen that causal graphs G_1 and G_2 depicted in the Figure 3 will be respectively represented by the following expressions:

$$lift(a) \cdot u(a) \cdot o \cdot b \; * \; lift(b) \cdot u(b) \cdot o \cdot b \tag{51}$$
$$key \cdot k \cdot b \tag{52}$$

Recall that products $(*)$ capture the idea of causes that need to work together to produce an effect, while application $(\cdot)$ captures the idea of causal chain. Furthermore, in general, we will be interested not only in expressions that represent single causes, but also expressions that allows representing several alternative causes. We will use the sum $(+)$ for separating alternative causes:

$$lift(a) \cdot u(a) \cdot o \cdot b \; * \; lift(b) \cdot u(b) \cdot o \cdot b \; + \; key \cdot k \cdot b \tag{49}$$

Furthermore, we will see that (51) is equivalent to the following expression:

$$\big(lift(a) \cdot u(a) * lift(b) \cdot u(b) \big) \cdot o \cdot b \tag{53}$$

which is more compact and closer to the graph representation of graph G_1. For this reason, we will usually prefer writing (53) rather than (51). Consequently, we will rephrase (49) as:

$$\big(lift(a) \cdot u(a) * lift(b) \cdot u(b) \big) \cdot o \cdot b \; + \; key \cdot k \cdot b \tag{54}$$

In the following we formally define causal values as the means to capture alternative causes.

Definition 3.8 (Causal term)**.** *A* (causal) term, *t, over a set of labels Lb, is recursively defined as one of the following expressions:*

$$t \;\; ::= \;\; l \;\mid\; \prod S \;\mid\; \sum S \;\mid\; t_1 \cdot t_2$$

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

When the set S is finite and non-empty, $S = \{t_1, \ldots, t_n\}$ we write $\prod S$ simply as $t_1 * \cdots * t_n$ and $\sum S$ as $t_1 + \cdots + t_n$. When $S = \emptyset$, as usual, $\prod \emptyset$ is the product identity which we denote by 1. Similarly, $\sum \emptyset$ is the sum identity which we denote by 0. We also assume that $(\cdot)$ has higher priority than $(*)$, and, in its turn, the last has higher priority than $(+)$.

Definition 3.9 (Causal value)**.** Causal value *are the classes of equivalence of causal terms under the equations of Figures 13, 14 and 15. We denote by $\mathbf{V}_{Lb}$ the set of causal values.* □

Associativity	*Addition distributivity*
$t \cdot (u{\cdot}w) \;=\; (t{\cdot}u) \cdot w$	$t \cdot (u{+}w) \;=\; (t{\cdot}u) + (t{\cdot}w)$
	$(t + u) \cdot w \;=\; (t{\cdot}w) + (u{\cdot}w)$

Absorption	*Identity*	*Annihilator*
$t \;=\; t + u \cdot t \cdot w$	$t \;=\; 1 \cdot t$	$0 \;=\; t \cdot 0$
$u \cdot t \cdot w \;=\; t * u \cdot t \cdot w$	$t \;=\; t \cdot 1$	$0 \;=\; 0 \cdot t$

Figure 13: Equations of the $(\cdot)$ operator (t, u, w are arbitrary causal terms).

Idempotence	*Product distributivity*	*Transitivity*
$l \cdot l = l$	$c \cdot (d * e) \;=\; (c \cdot d) * (c \cdot e)$	$c \cdot d \cdot e \;=\; (c \cdot d) * (d \cdot e)$ with $d \neq 1$
	$(c * d) \cdot e \;=\; (c \cdot e) * (d \cdot e)$	

Figure 14: Equations of the $(\cdot)$ operator (c,d,e terms without '+' and l is a label).

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*
$t + (u * w) \;=\; (t{+}u) * (t{+}w)$	$t \;=\; t + 0$	$t \;=\; t + t$	$1 \;=\; 1 + t$
$t * (u{+}w) \;=\; (t * u) + (t * w)$	$t \;=\; t * 1$	$t \;=\; t * t$	$0 \;=\; 0 * t$

Figure 15: Equations of a completely distributive lattice.

Besides these equations, it is also worth to introduce some extra derived from them, as those shown in Figure 16.

$$\begin{array}{ccc} \multicolumn{3}{c}{\underline{\textit{Absorption (der)}}} \\ d * c{\cdot}d & = & c{\cdot}d \\ d * d{\cdot}e & = & d{\cdot}e \end{array} \qquad \begin{array}{ccc} \multicolumn{3}{c}{\underline{\textit{Transitivity (der)}}} \\ c{\cdot}d * d{\cdot}e & = & c{\cdot}d * d{\cdot}e * c{\cdot}e \end{array}$$

Figure 16: Derived properties of the $(\cdot)$ and $(*)$ operations over causal graphs.

Proposition 3.2. *Equivalences in Figure 16 follows from those in Figures 13, 14 and 15.* □

Proof. The proof can be found in Appendix A on page 240. □

Definition 3.10 (Term associated to graphs)**.** *The function* $term : \mathbf{C}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ *given by:*

$$term(G) \stackrel{\text{def}}{=} \prod \{ \, v_1 \cdot v_2 \mid (v_1, v_2) \in G \, \} \tag{55}$$

associates a causal term (without sums) to every causal graph $G \in \mathbf{C}_{Lb}$. □

For instance, causal graph G_1 in Figure 3 can be represented as a term $term(G_1) = \alpha * \beta * \gamma$ where:

$$\begin{aligned} \alpha &= lift(a) \cdot u(a) * lift(b) \cdot u(b) * u(a) \cdot o * u(b) \cdot o * o \cdot b \\ \beta &= lift(a) \cdot lift(a) * lift(b) \cdot lift(b) * u(a) \cdot u(a) * u(b) \cdot u(b) * o \cdot o * b \cdot b \\ \gamma &= lift(a) \cdot o * lift(a) \cdot b * lift(b) \cdot o * lift(b) \cdot b * u(a) \cdot b * u(b) \cdot b \end{aligned}$$

Note that α corresponds to the product of the depicted edges of G_1. In their turn, β and γ respectively correspond to the edges in the reflexive and transitive closure of G_1. We will see that, by successive application of the equivalences shown in Figures 13, 14 and 15, it holds that $G_1 = \alpha * \beta * \gamma$ can be simplified first to $G_1 = \alpha$, then to the expression in (51), and finally to (53) First, note that by applying the *label idempotence* equation we can rewrite β just as:

$$\beta = lift(a) * lift(b) * u(a) * u(b) * o * b$$

Furthermore, applying the associative and commutative equations of the product, $\alpha * \beta$ can be rearranged as follows:

$$\begin{aligned}
\alpha * \beta = {} & lift(a) \cdot u(a) * lift(a) * \\
& lift(b) \cdot u(b) * lift(a) * \\
& \quad u(a) \cdot o \quad * \quad u(a) \quad * \\
& \quad u(b) \cdot o \quad * \quad u(b) \quad * \\
& \quad\; o \cdot b \quad\;\; * \quad\; o \quad\; * \\
& \quad\;\; b
\end{aligned}$$

and then, by applying the *(derived) absorption* equation, it follows that:

$$\begin{aligned}
\alpha * \beta = {} & lift(a) \cdot u(a) * \\
& lift(b) \cdot u(b) * \\
& \quad u(a) \cdot o \quad * \\
& \quad u(b) \cdot o \quad * \\
& \quad\; o \cdot b \quad\;\; * \\
& \quad\;\; b
\end{aligned}$$

Applying the same procedure to $o \cdot b * b$ it follows that $\alpha * \beta = \alpha$. We can follow a similar procedure in order to show that $\alpha * \gamma = \alpha$. Furthermore, applying *transitivity* equation, it follows that:

$$\begin{aligned}
\alpha &= \big(lift(a) \cdot u(a) * u(a) \cdot o * o \cdot b \big) * \big(lift(b) \cdot u(b) * u(b) \cdot o * o \cdot b \big) \\
&= \qquad lift(a) \cdot u(a) \cdot o \cdot b \qquad * \qquad lift(b) \cdot u(b) \cdot o \cdot b
\end{aligned}$$

which is the expression in (51). Applying the *distributivity of application over products*:

$$\begin{aligned}
\alpha &= lift(a) \cdot u(a) \cdot o \cdot b \; * lift(b) \cdot u(b) \cdot o \cdot b \\
&= \big(lift(a) \cdot u(a) * lift(b) \cdot u(b) \big) \cdot o \cdot b
\end{aligned}$$

Notice that $G_1 = \alpha * \beta * \gamma = \alpha = \big(lift(a) \cdot u(a) * lift(b) \cdot u(b) \big) \cdot o \cdot b$ which is exactly the same expression we have seen in (53). Although the form of this last expression is more readable, it has the drawback of not being a normal form, and hence, we will be forced, sometimes, to rely on the more verbose expression of the form $\alpha * \beta * \gamma$.

By abuse of notation, we usually will omit *term*() and just write G instead of $term(G)$. Hence, we may just rewrite (54) as $G_1 + G_2$. Furthermore note that, at first sight, the equations of Figures 13, 14 and 15 may seem arbitrarily chosen. However, these equations derive from a semantic structure (ideals of causal graphs) which captures the intuitive idea that causal values are the set $\subseteq$-minimal causal graphs. We will address the details of the structure behind causal values in the next section. In particular, the following result (whose formal proof is also addressed in the next section) reflects the relation between causal values and causal graphs.

Theorem 3.1. *Given a set of labels Lb, the structure $\langle \mathbf{V}_{Lb}, +, * \rangle$ is isomorphic to the free completely distributive (complete) lattice, with meet ($*$) and join ($+$), generated by the set of causal graphs $\mathbf{C}_{Lb}$. The function $term : \mathbf{C}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ is an embedding.* $\square$

As usual in lattices, we say that a causal value t is smaller than other u:

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

An important observation that should be made is that, for any pair of causal graphs G_1 and G_2 it holds that $G_1 \subseteq G_2$ iff $term(G_1) \geq term(G_2)$. Note that, by our convention of omitting $term()$, this may be rewriting as $G_1 \subseteq G_2$ iff $G_1 \geq G_2$. Saying that G_1 is *stronger* than G_2, in symbols $G_1 \geq G_2$, intuitively means that G_2 contains enough information to yield the same effect than G_1, but perhaps more than needed (this explains $G_1 \subseteq G_2$). This implies that causes, which are $\subseteq$-minimal causal graphs, will be $\leq$-maximal causal values. An interesting side effect of switching $\leq$ with respect to the subgraph relation $\subseteq$ is that, as usual, the semantics of logic programs will coincide with selecting $\leq$-minimal models.

For a better characterization of the intuition that causal values intuitively represent sets of $\subseteq$-minimal causal graphs, we introduce the following normal forms.

Definition 3.11 (Disjunctive normal form)**.** *A causal term t is in* (minimal) disjunctive normal form *iff sums are not in the scope of applications and products, and all addends in a sum are pairwise incomparable. It is in* (minimal) chain normal form *iff furthermore application is not in the scope of products, and all chains of applications are pairwise incomparable. Moreover, it is in* graph normal form *iff application is only applied to labels and all terms of products are pairwise incomparable.* $\square$

For instance, in Example 3.1, we may associate with *alarm* the following causal term in non-minimal disjunctive normal form:

$$\begin{aligned}(sw_1 \cdot b \cdot d * sw_3) \cdot a \;+\; (sw_2 \cdot c \cdot d * sw_3) \cdot a \\ +\; ((sw_2 \cdot c * sw_4) \cdot b \cdot d * sw_3) \cdot a\end{aligned} \tag{56}$$

Each addend respectively correspond to the causal graphs G_1, G_2 and G_3 depicted in Figure 5. Since $G_2 \subseteq G_3$, it follows that $G_2 \geq G_3$, that is:

$$(sw_2 \cdot c \cdot d * sw_3) \cdot a \quad \geq \quad ((sw_2 \cdot c * sw_4) \cdot b \cdot d * sw_3) \cdot a$$

Therefore the causal term (56) is not in minimal disjunctive normal form and may be rewritten as the equivalent causal term:

$$(sw_1 \cdot b \cdot d * sw_3) \cdot a \;+\; (sw_2 \cdot c \cdot d * sw_3) \cdot a \tag{57}$$

in which each addend respectively correspond to the causal graphs G_1 and G_2. That is, each addend corresponds to a $\subseteq$-minimal causal graph, and thus to a cause. We may also apply the distributivity of application over products, so that we obtain a causal term in chain normal form:

$$sw_1 \cdot b \cdot d \cdot a \;*\; sw_3 \cdot a \;\;+\;\; sw_2 \cdot c \cdot d \cdot a \;*\; sw_3 \cdot a \tag{58}$$

Finally, we may apply transitivity to obtain a causal term in graph normal form:

$$sw_1 \cdot b \;*\; b \cdot d \;*\; d \cdot a \;*\; sw_3 \cdot a \;\;+\;\; sw_2 \cdot c \;*\; c \cdot d \;*\; d \cdot a \;*\; sw_3 \cdot a \tag{59}$$

Note that, in each addend, each causal chain of the form $v_1 \cdot v_2$ respectively correspond to an edge (v_1,v_2) in the causal graphs G_1 and G_2.

3.4 ALGEBRAIC PROPERTIES OF CAUSAL VALUES

In the previous section, we have seen that we will represent alternative causes by means of causal values. In this section, we will derive the axiomatization given by equations of Figures 13, 14 and 15 from basic operations on causal graphs. We use standard algebraic notation that has been collected in the Background (Section 2.2). Furthermore, we represent the reflexive and transitive closure of a graph G as G^*, and its reflexive and transitive reduction as G^r. Recall that causal graphs are transitively and reflexively closed set of edges. We write $(v,v') \in G$ to state that the (v,v') is an edge of G and $v \in G$ as a shorthand of $(v,v) \in G$, that is, v is a vertex of G. By abuse of notation we will also use v to represent a causal graph with a unique reflexive edge corresponding to the vertex v, that is the set $\{\ (v,v)\ \}$. We also introduce a *concatenation* operation $G \odot G'$ on causal graphs corresponding to a graph with edges

$G \cup G' \cup \{(v,v') \mid v \in G, v' \in G'\}$. Notice that $G \cup G' \subseteq G \odot G'$, that is, the concatenation extends the union of causal graphs by adding all possible edges that go from some node in G to some node in G'. By using this operations we can split any graph as an algebraic expression formed by *atomic graphs* which just contain a single vertex. For instance, the graph G_1 in Figure 3 corresponds to the expression:

$$\Big(\big(lift(a) \odot u(a)\big) \cup \big(lift(b) \odot u(b)\big) \Big) \odot o \odot b \tag{60}$$

The graph resulting from (60) is transitively closed but not reflexively. Hence, the causal graph corresponding to G_1 in Figure 3 will be the reflexive closure of (60). However, in general, operations ($\cup$) and ($\odot$), on causal graphs, do not guarantee that the result will be even transitively closed. In order to deal with internal operations inside the set of causal graphs we define two of the three operations mentioned in the previous section: product ($*$) and application ($\cdot$) are respectively the transitive and reflexive closure of operations ($\cup$) and ($\odot$). That is, $G_1 * G_2 \overset{\text{def}}{=} (G_1 \cup G_2)^*$ and $G_1 \cdot G_2 \overset{\text{def}}{=} (G_1 \odot G_2)^*$. In this sense, the causal graph G_1 and G_2 in Figure 3 respectively correspond to the expressions:

$$\big(lift(a) \cdot u(a) * lift(b) \cdot u(b)\big) \cdot o \cdot b \tag{53}$$

$$key \cdot k \cdot b \tag{52}$$

Proposition 3.3. *Product and application are associative and they have the empty graph as their identity. Furthermore, product is also commutative and idempotent. Application is idempotent but only with respect to atomic causal graphs. Both are also monotonic with respect to the subgraph relation.* □

Proof. The proof can be found in Appendix A on page 241. □

Since product is associative and commutative, we can extend their application, not only two a pair of causal graphs, but also to any set of them. Thus, for any (possibly infinite) set of causal graphs S, by $\prod_{G \in S} G$ or just $\prod S$ we denote the product of all causal graphs in S.

Proposition 3.4. *Application distributes over (possible infinity) products, and they hold the absorption and transitivity laws as shown in Figures 13 and 14.* □

Proof. The proof can be found in Appendix A on page 242. □

Notice the importance of the distributive law for connecting causal graphs with the concept of *causal chain*: its application to (53) leads to (51).

Definition 3.12 (Causal graph term)**.** *A (causal) graph term or cg-term, t is a causal term without sums. By $\mathbf{C}^t_{Lb} \subseteq \mathbf{V}_{Lb}$ we denote the set of causal values which have some cg-term as representative.* □

Theorem 3.2 (Causal graphs isomorphism)**.** *The algebraic structures $\langle \mathbf{C}_{Lb}, *, \cdot \rangle$ and $\langle \mathbf{C}^t_{Lb}, *, \cdot \rangle$ are isomorphic, and the function $term : \mathbf{C}_{Lb} \longrightarrow \mathbf{C}^t_{Lb}$ (Definition 3.10) is an isomorphism.* □

Proof. The proof can be found in Appendix A on page 249. □

Theorem 3.2 formally settle the intuitive correspondence between causal graphs and causal terms without sums that we have introduce in the previous section, and justify our convention of omit $term()$ and write just G instead of $term(G)$.

Proposition 3.5. *Given a labelled program P, any proof $\pi(A)$ satisfies that:*

$$cgraph(\pi) \quad = \quad \big(cgraph(\pi_1) * \ldots * cgraph(\pi_m)\big) \cdot l_R \cdot l_A$$

where $\pi_1, \ldots, \pi_m$ are the direct sub-proofs of π, l_R is the label of its rule R and l_A is a label associated to the atom in the consequent A. □

Proof. The proof can be found in Appendix A on page 244. □

Proposition 3.5 formalises the intuition that product captures causes that need to work together whereas, in its turn, application captures the idea of a sequence of events that connect a cause to its final effect.

Definition 3.13 (Stronger Cause)**.** *A causal graph G is said to be* stronger *than another causal graph G', written $G \leq G'$, when $G \supseteq G'$.* □

The following result, which establishes the relation between product $(*)$ and application $(\cdot)$ operations and the $\leq$ order relations, will be useful through the dissertation.

Proposition 3.6. *The following statements hold:*

i) product is the greatest lower bound of the sufficient $\leq$ relation,

ii) for every pair of causal graphs G and G' it holds that $G \leq G'$ iff $G * G' = G$.

iii) both product and application are monotonic operations with respect to sufficient $\leq$ relation,

iv) for every pair of causal graphs G and G' it holds that $G \cdot G' \leq G * G'$. □

Proof. The proof can be found in Appendix A on page 245. □

Definition 3.14 (Ideal)**.** *Given any poset $\langle A, \leq \rangle$, an* ideal[3] *S is any set $S \subseteq A$ satisfying that: if $x \in S$ and $y \leq x$ then $y \in S$.* $\square$

When any subset $S \subseteq A$ has a set of maximal elements (that is $\leq$ is an upwards well-founded relation), a compact way of representing an ideal S is by using its set of maximals elements U, since the rest of S contains *all elements below U*. The *principal ideal* of an individual element $x \in A$ is denoted as $\downarrow x \overset{\text{def}}{=} \{y \in A \mid y \leq x\}$, and we extend this notion for any set of elements U so that:

$$\Downarrow U \overset{\text{def}}{=} \bigcup \{ \downarrow x \mid x \in U \} \ = \ \{ y \in A \mid y \leq x, \text{for some } x \in U \} \tag{61}$$

For the sake of compactness, when an ideal S has a set U of maximal elements, we will represent it by $\Downarrow U$. For instance, going back to the firing alarm of Example 3.1 we may represent the causal value corresponding to the atom $current(d)$ by the following expression:

$$sw_1 \cdot b \cdot d \ + \ sw_2 \cdot c \cdot d \ + \ (sw_1 \cdot b * sw_4) \cdot c \cdot d \ + \ (sw_2 \cdot c * sw_4) \cdot b \cdot d \tag{62}$$

Let G'_1, G'_2, G'_3 and G'_4 be respectively the causal graphs corresponding to each of the addends of (62). Then, if we interpret each addend in (62) as a principal ideal and sums (+) as set unions, we may rephrase (62) as:

$$\downarrow G'_1 \ \cup \ \downarrow G'_2 \ \cup \ \downarrow G'_3 \ \cup \ \downarrow G'_4 \tag{63}$$

Furthermore, since $G'_3 \leq G'_1$ and $G'_4 \leq G'_2$ it follows $\downarrow G'_3 \subseteq \downarrow G'_1$ and $\downarrow G'_4 \subseteq \downarrow G'_2$. Hence:

$$\begin{aligned} \downarrow G'_1 \ \cup \ \downarrow G'_3 \ &= \ \downarrow G'_1 \\ \downarrow G'_2 \ \cup \ \downarrow G'_4 \ &= \ \downarrow G'_2 \end{aligned}$$

Consequently (63) is simply equivalent to:

$$\downarrow G'_1 \ \cup \downarrow G'_2 \ = \ \Downarrow \{ \ G'_1, \ G'_2 \ \} \tag{64}$$

That is, we just obtain two expressions corresponding to the two causes of the presence of current at point d. In addition, from the rules:

$$a: \quad alarm \leftarrow down(sw_3), \ current(c) \qquad\qquad \$: \quad sw_3$$

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

it follows that:

$$I(alarm) \quad = \quad \Downarrow\{ \, G_1' \cdot a, \; G_2' \cdot a \, \} \tag{65}$$

which is the $\subseteq$-minimal ideal containing the two causes, G_1 and G_2 depicted in Figure 5.

Notice that, as we have seen in the above example, the relevant information of the ideal corresponds to its set of maximal elements, while keeping all other elements is convenient for simplicity of algebraic treatment (but we do not assign a particular meaning to them). The next result shows that every ideal in the partial order set of causal graphs has maximal elements.

Proposition 3.7. *Given any (possible infinite) ideal (down set) $S \subseteq \mathbf{C}_{Lb}$ from the partial order set $\langle \mathbf{C}_{Lb}, \leq \rangle$ there exists a subset $U \subseteq S$ such that $\Downarrow U = S$ and no pair of elements of U are $\leq$-comparable.* □

Proof. If such subset U does not exist, then there must be an infinite increasing chain of causal graphs $G_1 \leq G_2 \leq \ldots$ in S. However, just recall that, by definition, $G_1 \leq G_2$ holds if and only if $G_1 \supseteq G_2$ where $\supseteq$ stands for the superset relation (causal graphs are represented by the set of their edges). Then there exists a infinite decreasing chain of sets $G_1 \supseteq G_2 \supseteq \ldots$ which is a contradiction with the fact that the subset relation is well-founded. □

Definition 3.15 (Causal Ideals). *Given a set of labels Lb, by $\mathbf{I}_{Lb}$ we denote the set of ideals of the partially ordered set $\langle \mathbf{C}_{Lb}, \leq \rangle$.* □

It is worth to mention that the set of causal graphs $\mathbf{C}_{Lb}$ and the set of principal ideals, that we denote by $\Downarrow\mathbf{C}_{Lb}$, are isomorphic. Furthermore, the function $\downarrow : \mathbf{C}_{Lb} \longrightarrow \Downarrow\mathbf{C}_{Lb}$ is an isomorphism that maps products to set intersections and the $\leq$ relation to the subset $\subseteq$ relation.

Proposition 3.8. *The function $\downarrow : \mathbf{C}_{Lb} \longrightarrow \Downarrow\mathbf{C}_{Lb}$ is an isomorphism between structures $\langle \mathbf{C}_{Lb}, \leq, *, 1 \rangle$ and $\langle \Downarrow\mathbf{C}_{Lb}, \subseteq, \cap, \mathbf{C}_{Lb} \rangle$.* □

Proof. The proof can be found in Appendix A on page 245. □

In order to keep notation, we will still use the product to represent set intersection and $\leq$ to represent the subset relation. In addition, we use the sum operator to represent set unions. As usual, we also denote by 0 the bottom element of the $\leq$ relation, that is the empty set $\emptyset$, and by 1 its top element which is the principal ideal formed by the empty graph, also represented by 1, and is equal to the set $\mathbf{C}_{Lb}$. We also define the application over ideals in the following preserving way:

$$U \cdot U' \stackrel{\text{def}}{=} \Downarrow\{\, G \cdot G' \mid G \in U \text{ and } G' \in U' \,\} \tag{66}$$

Proposition 3.9. *Application is associative, distributes over sums and holds the absorption, identity and annihilator properties reflected in Figure 13.* □

Proof. The proof can be found in Appendix A on page 88. □

Proposition 3.10. *The function* $\downarrow \colon \mathbf{C}_{Lb} \longrightarrow \Downarrow\mathbf{C}_{Lb}$ *is an isomorphism between structures* $\langle \mathbf{C}_{Lb}, \leq, *, \cdot, 1\rangle$ *and* $\langle \Downarrow\mathbf{C}_{Lb}, \leq, *, \cdot, 1\rangle$. □

Proof. The proof can be found in Appendix A on page 247. □

Proposition 3.10 means that all equivalences that hold for causal graphs also hold for their principal ideals. Figure 14 shows those equations that hold for causal graphs (and so for principal ideals or causal terms without sums) but do not apply to all causal values in general.

It is also worth to recall that, since sum and product are defined respectively as set union and intersection, they hold the usual properties of these operations, that is, the equivalences of *completely distributive (complete) lattices* shown in Figure 15. It is important to mention that the result of adding the sum operator does not force any new equivalence between principal ideals that does not hold for causal graphs. The following results asserts this fact.

Theorem 3.3 (Cabalar et al. [2014a, Theorem 1])**.** *Given a set of labels Lb, the structure* $\langle \mathbf{I}_{Lb}, +, * \rangle$ *is isomorphic to the free completely distributive (complete) lattice generated by the principal ideals of causal graphs* $\Downarrow\mathbf{C}_{Lb}$. □

Proof. The proof can be found in Appendix A on page 256. □

Theorem 3.4 (Causal values isomorphism)**.** *The algebras* $\langle \mathbf{V}_{Lb}, +, *, \cdot, 1, 0\rangle$ *and* $\langle \mathbf{I}_{Lb}, +, *, \cdot, 1, 0\rangle$ *are isomorphic, and*

$$term(U) \mapsto \sum\{\, term(G) \mid G \in U \,\} \tag{67}$$

where $term(G)$ *refers to the mapping Definition 3.10, is an isomorphism.* □

Proof. The proof can be found in Appendix A on page 257. □

Theorem 3.4 shows that in order to represent the idea of alternative sufficient causes, we can use causal values or causal terms interchangeably by means of the isomorphism (67). Notice also that Theorem 3.4 also means that every causal value (or causal term) can be rewritten in a causal graph normal form (Definition 3.11). Now the proof of Theorem 3.1 follows straightforwardly from Theorems 3.3 and 3.4.

Proof of Theorem 3.1. Note that, from Theorem 3.3, $\langle \mathbf{I}_{Lb}, +, * \rangle$ is isomorphic to the free completely distributive lattice generated by the principal ideals of causal graphs $\Downarrow\mathbf{C}_{Lb}$. Furthermore, the sets of causal graphs $\mathbf{C}_{Lb}$ and the set of ideals of causal graphs $\Downarrow\mathbf{C}_{Lb}$ are also isomorphic. Hence, $\langle \mathbf{I}_{Lb}, +, * \rangle$ is isomorphic to the free completely distributive lattice generated by the set of causal graphs $\mathbf{C}_{Lb}$. Moreover, from Theorem 3.4, the algebras $\langle \mathbf{V}_{Lb}, +, *, \cdot, 1, 0 \rangle$ and $\langle \mathbf{I}_{Lb}, +, *, \cdot, 1, 0 \rangle$ are isomorphic, and so $\langle \mathbf{V}_{Lb}, +, * \rangle$ and $\langle \mathbf{I}_{Lb}, +, * \rangle$ are isomorphic too. Consequently $\langle \mathbf{V}_{Lb}, +, * \rangle$ is isomorphic to the free completely distributive lattice generated by the set of causal graphs $\mathbf{C}_{Lb}$. □

The following result establishes the relation between operations and the sufficient $\leq$ order relation extending the results of Proposition 3.6 from causal graphs to causal values.

Proposition 3.11. *The following statements hold.*

i) product ($*$) and sum ($+$) are respectively the greatest lower bound and the least upper bound of the sufficient $\leq$ relation,

ii) for every pair of causal values u and u' it holds that $u \leq u'$ iff $u * u' = u$ iff $u + u' = u$.

iii) product ($*$), application ($\cdot$) and sums ($+$) are monotonic and continuous operations with respect to the sufficient $\leq$ relation,

iv) $u \cdot u' \leq u * u' \leq u + u'$ for every pair of causal values u and u'. □

Proof. The proof can be found in Appendix A on page 258. □

4 CAUSAL SEMANTICS FOR LABELLED LOGIC PROGRAMS

In this section we define several semantics for labelled logic programs which are conservative extensions of the *stable model*, the *well-founded model* and the *answer set semantics*. As usual, we give the semantics for ground programs; programs with variables are understood as abbreviations of all their ground instances.

4.1 CAUSAL SEMANTICS FOR POSITIVE PROGRAMS

Definition 4.1 (Interpretation). *A* (causal) interpretation *$I : At \longrightarrow \mathbf{V}_{Lb}$ is a mapping assigning a causal value to each literal. We denote by $\mathbf{I}_\sigma$ the set of all possible interpretations over a signature σ. We say that an interpretation I* satisfies *an atom A, in symbols $I \models A$ iff $I(A) \neq 0$.* □

We will see later that satisfaction, that is $I(A) \neq 0$, is equivalent to say that there is some cause for A. Note also that the order relation $\leq$ among causal values is extend over interpretations that, for interpretations I and J, we say that $I \leq J$ when $I(A) \leq J(A)$ for all atom $A \in At$. Hence, there is a $\leq$-bottom (resp. $\leq$-top) interpretation $\mathbf{0}$ (resp. $\mathbf{1}$) that stands for the interpretation mapping each literal A to 0 (resp. 1). An interpretation is *two-valued* if it maps all atoms into $\{0,1\}$. Furthermore, for any causal interpretation, its corresponding two-valued interpretation, written I^{cl}, is defined so that for any literal A: $I^{cl}(A) \overset{\text{def}}{=} 0$ if $I(A) = 0$; and $I^{cl}(A) \overset{\text{def}}{=} 1$ otherwise.

Definition 4.2 (Least causal model). *Given a positive labelled rule R of the form* (48) *(with $m = 0$) and mapping $\delta : At \longrightarrow Lb \cup \{1\}$, we say that a causal interpretation I* satisfies *R iff the following condition holds:*

$$\big(\, I(B_1) * \ldots * I(B_m) \,\big) \cdot l_R \cdot \delta(A) \;\leq\; I(A) \tag{68}$$

Given a positive program P, we say that an interpretation I is a causal model *of P, in symbols $I \models P$, iff I is a $\leq$-minimal causal model of all rules R in P.* □

Note that if all rules are unlabelled, that is $l_R = 1$ for all rule R and δ is an empty atom labelling (maps each literal into 1), then, since 1 is the identity of application, condition (68) becomes:

$$I(B_1) * \ldots * I(B_m) \;\leq\; I(A) \tag{69}$$

Note that, if we further have two-valued interpretations, this is just the least model definition for standard logic programs. In this sense, a *standard (logic) program* is just a labelled program with an empty atom labelling which contains only unlabelled rules. As it is the case for standard logic programs, positive programs have a $\leq$-least model that can be computed by iterating an extension of the well-known *direct consequences operator* van Emden and Kowalski [1976].

Definition 4.3 (Direct consequences)**.** *Given a positive program P, the operator of* direct consequences $T_P : \mathbf{I}_\sigma \longrightarrow \mathbf{I}_\sigma$ *is a function such that* $T_P(I)(A)$ *is given by:*

$$\sum \{ \; (\; I(B_1) * \ldots * I(B_m) \;) \cdot l_R \cdot \delta(A) \;\big|\; R \in P \textit{ and } head(R) = A \; \}$$

for any interpretation I and any atom $A \in At$. □

The iterative application of the direct consequences operator can be used analogously as in standard logic programming (see page 49) to get the least model from its least fixpoint $\mathtt{lfp}(T_P)$.

Theorem 4.1 (Direct consequences)**.** *Let P be a (possibly infinite) positive labelled program. Then,*

i) $\mathtt{lfp}(T_P)$ *is the least model of P, and*

ii) $\mathtt{lfp}(T_P) = T_P^\omega(\mathbf{0})$.

Furthermore, if P is finite and has n rules, then

iii) $\mathtt{lfp}(T_P) = T_P^n(\mathbf{0})$. □

The proof of Theorem 4.1 will rely on an encoding of causal logic programs into *Generalized Annotated Logic Programming* (GAP) [Kifer and Subrahmanian, 1992] and applying existing results for that general multivalued LP framework. This encoding will be discussed in Chapter 6 and, in particular, we will see that Theorem 4.1 is a particular case of Theorem 6.2 on that chapter.

To illustrate how the iterative procedure of the T_P operator works we appeal to the circuit in Example 3.1. The following tables depict each iteration until we reach the least fixpoint. In particular, the next table shows the iterative computation of the T_P operator for literals $current(b)$ and $current(c)$.

i	$current(b)$	$current(c)$
1	0	0
2	$sw_1 \cdot b$	$sw_2 \cdot c$
3	$sw_1 \cdot b + (sw_2 \cdot c * sw_4) \cdot b$	$sw_2 \cdot c + (sw_1 \cdot b * sw_4) \cdot c$
4	$sw_1 \cdot b + (sw_2 \cdot c * sw_4) \cdot b$	$sw_2 \cdot c + (sw_1 \cdot b * sw_4) \cdot c$
5	$sw_1 \cdot b + (sw_2 \cdot c * sw_4) \cdot b$	$sw_2 \cdot c + (sw_1 \cdot b * sw_4) \cdot c$

Figure 17: Iterative application $T_P^i(current(b))$ and $T_P^i(current(c))$.

Note that $T_P^0(\mathbf{0})(A) = 0$ for any literal A, and furthermore $T_P^i(\mathbf{0})(sw_j) = sw_j$ for all $i \geq 1$ and $j \in \{1,2,3,4\}$. Hence we omit the values for these cases. Note also that, for the sake of simplicity, we have assumed that δ is an empty atom labelling (that is $\delta(A) = 1$ for a literal A). Otherwise, a label corresponding to the head of each rule should be added. For instance, if $\delta(A) = l_A$ for any atom A, then $T_P^5(\ current(b))$ would be:

$$sw_1 \cdot l_{sw_1} \cdot b \cdot l_{current(b)} + (sw_2 \cdot l_{sw_2} \cdot c \cdot l_{current(c)} * sw_4 \cdot l_{sw_4}) \cdot b \cdot l_{current(b)}$$

The following table shows the corresponding iterations for $current(d)$.

i	$current(d)$
1	0
2	0
3	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d$
4	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d + (sw_2 \cdot c * sw_4) \cdot b \cdot d + (sw_1 \cdot b * sw_4) \cdot c \cdot d$
5	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d + (sw_2 \cdot c * sw_4) \cdot b \cdot d + (sw_1 \cdot b * sw_4) \cdot c \cdot d$

Figure 18: Iterative application $T_P^i(d)$. No simplification has been performed.

It is worth to mention that, according to the algebraic properties of causal values, it holds that:

$$sw_1 \cdot b \cdot d \ < \ sw_1 \cdot b * sw_4 \cdot c \cdot d$$
$$sw_2 \cdot c \cdot d \ < \ sw_2 \cdot c * sw_4 \cdot b \cdot d$$

and, since $\alpha \leq \beta$ iff $\alpha + \beta = \beta$, it follows that $T_P^2(\mathbf{0})(current(d))$ is equal to $T_P^i(\mathbf{0})(current(d))$ for all $i \geq 2$. The following table shows the value of $current(d)$, after being simplified, together with the value for $alarm$:

i	$current(d)$	$alarm$
1	0	0
2	0	0
3	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d$	0
4	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d$	$(sw_1 \cdot b \cdot d * sw_3) \cdot a + (sw_2 \cdot c \cdot d * sw_3) \cdot a$
5	$sw_1 \cdot b \cdot d + sw_2 \cdot c \cdot d$	$(sw_1 \cdot b \cdot d * sw_3) \cdot a + (sw_2 \cdot c \cdot d * sw_3) \cdot a$

Figure 19: Iterative application $T_P^i(d)$ and $T_P^i(alarm)$. Simplification has been performed.

Terms $(sw_1 \cdot b \cdot d * sw_3) \cdot a$ and $(sw_2 \cdot c \cdot d * sw_3) \cdot a$ respectively correspond to graphs G_1 and G_2 in Figure 5 representing the causes of *alarm* in Program 3.2. Furthermore, that the term $(sw_2 \cdot c * sw_4) \cdot b \cdot d$ appearing in Figure 18 would produce the term $((sw_2 \cdot c * sw_4) \cdot b \cdot d * sw_3) \cdot a$ for *alarm*, which corresponds to the graph G_3 in Figure 5. As we discussed in Chapter 3, we reasonably obtain that G_3 was not a cause of *alarm*, while G_1 and G_2 were. As we have just seen, algebraic equivalences allow selecting causes, those $\subseteq$-minimal ($\leq$-maximal) causal graphs, by computing the minimal disjunctive normal form of the causal term associated to each literal. It is also worth to recall that 0 is the bottom element of the $\leq$-relation, thus, the value that every literal will take *by default* when there is not cause for it. That is, 0 corresponds to be *false*.

Notice that Theorem 4.1 ensures that this iterative procedure ends after a finite number of steps for finite programs. This may seem surprising for programs with positive cycles given that the T_P operator is accumulating labels in each step. However, every explanation derived from a cycle will be not minimal because there is another explanation that does not involve the cycle. To illustrate this we appeal to the simplified gear wheel scenario of Example 2.1 represented by the following labelled program:

Program 4.1.

$$
\begin{aligned}
t_1 &: spinning(1) \leftarrow spinning(2),\, coupled \\
t_2 &: spinning(2) \leftarrow spinning(1),\, coupled \\
c \ &: coupled \\
s \ &: spinning(1)
\end{aligned}
$$

Then, the iteration $T_P^i(\mathbf{0})$ yields the following steps:

i	$coupled$	$spinning(1)$	$spinning(2)$
1	c	s	0
2	c	s	$(s * c) \cdot t_2$
3	c	$s + ((s * c) \cdot t_2 * c) \cdot t_1$	$(s * c) \cdot t_2$
4	c	$s + ((s * c) \cdot t_2 * c) \cdot t_1$	$(s * c) \cdot t_2(((s * c) \cdot t_2 * c) \cdot t_1 * c) \cdot t_2$
	$\dots$	$\dots$	$\dots$

Figure 20: Iterative application $T_P^i(spinning(1))$ and $T_P^i(spinning(2))$.

By the equations of causal terms, it holds that $s + ((s * c) \cdot t_2 * c) \cdot t_1 = s$. Similarly, $(s * c) \cdot t_2 + (((s * c) \cdot t_2 * c) \cdot t_1 * c) \cdot t_2 = (s * c) \cdot t_2$. Consequently, it follows that $T_P^2(\mathbf{0}) = T_P^3(\mathbf{0}) = T_P^4(\mathbf{0}) = \mathtt{lfp}(T_P)$. Following the ideas shown in these examples, we may now formally define the concept of sufficient cause by relying on the causal value associated to a literal.

Definition 4.4 (Sufficient Cause)**.** *Given an interpretation I, a causal graph G is* (sufficient) cause *of an atom A, in symbols $G \leq_{\max} I(A)$, iff it is a $\subseteq$-minimal ($\leq$-maximal) causal graph G such that $G \leq I(A)$.* □

For instance, according to Definition 4.4, causal graphs G_1 and G_2 in Figure 5 are the only sufficient causes of the literal *alarm*. Similarly, s and $s \cdot t_2$ are respectively the only sufficient causes of the literals $spinning(1)$ and $spinning(2)$ in the gear wheels scenario above.

Observation 1. *Let S_A be the set of sufficient causes of some literal A with respect to some interpretation I. Then $\sum_{G \in S_A} G = I(A)$.* □

We address now the relation between the syntactic notion of *non-redundant proofs* and the sufficient causes of a literal.

Theorem 4.2 (Proof-cause correspondence (positive programs))**.** *Given a positive, completely labelled program P with a unique atom labelling δ, a proof π is a non-redundant proof of a literal A iff $cgraph(\pi)$ is a sufficient cause of A w.r.t. its least model.* □

Theorem 4.2 shows that causes, obtained by purely semantic methods, exactly correspond to the syntactic notion of non-redundant proofs in completely labelled programs. The proof of this result is actually postponed to Chapter 6 where Theorem 6.1 generalizes it. The following result states the correspondence between our causal semantics and the standard least model semantics for positive programs.

Theorem 4.3 (Correspondence to non-causal least model semantics)**.** *Let P be a positive labelled program and P' its standard unlabelled version. Furthermore, let I be the least causal model of P and I' the standard least two-valued model of P'. Then, they satisfy $I' = I^{cl}$.* □

Proof. The proof can be found in Appendix A on page 260. □

In the above theorem, I' denotes the standard least model of P. That is, I' assigns 1 to a literal A if A is a consequence of P and 0 otherwise. Recall that I^{cl} is a two-valued interpretation which replaces by 1 each value different from 0. Then, Theorem 4.3 shows that every literal has a cause — that is, a value different from 0 — if and only if it holds in the standard least model of the program. We may also define the entailment of a program in the usual way:

Definition 4.5 (Entailment)**.** *Given a positive program P, we say that P satisfy A, in symbols $P \models A$, when the least model I of P satisfies A, that is $I \models A$.* □

which leads to the following more friendly rewriting of Theorem 4.3.

Corollary 4.1. *Let P be a positive program and P' its standard unlabelled version. Then $P \models A$ if and only if $P' \models A$.* □

Notice that, in previous work, Cabalar et al. [2014a,b] defined a causal model in the following way:

Definition 4.6 (Least causal model Cabalar et al. [2014a]). *Given a positive rule R of the form* (48) *(with $m = 0$), and an interpretation I, I is said to satisfy R iff:*

$$\big(\ I(B_1) * \ldots * I(B_m)\ \big) \cdot l_R \ \leq\ I(A) \tag{70}$$

Given a positive program P, I is the least causal model of P if and only if I is the $\leq$-minimal causal model such that $R \models I$ for all rules R in P. □

It is easy to see that this definition corresponds to our current definition (Definition 4.2) when δ is an empty atom labelling ($\delta(A) = 1$ for every literal A). Note that, since 1 is the application identity, it follows that $\alpha \cdot l_R \cdot \delta(A) = \alpha \cdot l_R \cdot 1 = \alpha \cdot l_R$ for all literal A. As introduced in Chapter 3 and we have just showed above in Theorem 4.2, a unique atom labelling δ allows to better capture the notion of *non-redundant proof*.

4.2 CAUSAL ANSWER SET SEMANTICS

A usual way of informally reading a NAF-literal of the form *not A* is as "there is no way to derive A." As we commented in the introduction, we will give a causal meaning to it, so that, *not A* may be read as "there is no cause for A." Recall that the causal value 0 stands, according to our interpretation, for the absence of cause. Hence, we may define the value assigned by an interpretation I to a negative NAF-literal *not A*, in symbols $I(not\ A)$, as:

$$I(not\ A) \ \overset{\text{def}}{=}\ \begin{cases} 1 & \text{if } I(A) = 0 \\ 0 & \text{otherwise} \end{cases}$$

According to this evaluation, default negation does not propagate any causal value. In order to illustrate how this definition may be useful for representing default knowledge reconsider the match-oxygen asymmetry of Example 1.7.

Program 4.2.

$$\begin{array}{llll} f: & fire \leftarrow match, oxygen & \$: & match \\ & & \$: & oxygen \end{array}$$

It is easy to see that the least model I of this program must satisfy:

$$I(fire) \quad = \quad (match * oxygen) \cdot f$$

That is, both, the *match* and the *oxygen*, form part of the cause of the *fire* as equal partners. We may now add a rule stating that *oxygen* is present by default as follows.

Program 4.3.

$$\begin{array}{llll} f: & fire \leftarrow match, oxygen & \$: & match \\ & oxygen \leftarrow not\ \overline{oxygen} & \$: & oxygen \end{array}$$

If we consider now the Program 4.3 it is easy to see that there is no rule with $\overline{oxygen}$ in the head, and therefore $I(\overline{oxygen}) = 0$. Hence, the valuation of the literal $not\ \overline{oxygen}$ by I will correspond to $I(not\ \overline{oxygen}) = 1$. Notice furthermore that, since unlabelled rules are syntactic sugar for rules labelled with the application identity value 1, it follows that:

$$I(oxygen) \quad = \quad oxygen + 1 \cdot 1 \quad = \quad oxygen + 1 \quad = \quad 1$$

Then, I must also satisfy that:

$$I(fire) \quad = \quad (match * 1) \cdot f \quad = \quad match \cdot f$$

That is, including the unlabelled rule, which states that *oxygen* is present by default, makes the effect of removing it from every cause that would require it otherwise. It is easy to see that this behaviour agrees with the principle stated in the Introduction that causes do not need to incorporate those conditions that hold by default. Another approach to represent this example is by considering the absence of oxygen to be an exception to the rule f in the following way:

Program 4.4.

$$\begin{array}{llll} f: & fire \leftarrow match, not\ ab & \$: & match \\ ab: & ab \leftarrow \overline{oxygen} & \$: & oxygen \end{array}$$

Note that in Program 4.4 there is not any rule with $\overline{oxygen}$ in the head either. Therefore $I(\overline{oxygen}) = 0$. Hence, $I(ab) = 0 \cdot ab = 0$ and, thus, $I(fire) = match \cdot f$ as in the previous case.

In the rest of the section we formalise causal semantics for normal programs that follows the principles of the *stable model semantics*. We may define a causal semantics for normal programs by appealing to a straightforward extension of the traditional program reduct defined by [Gelfond and Lifschitz, 1988].

Definition 4.7 (Causal stable model). *The* reduct *of program P with respect to an interpretation I, in symbols P^I, is the result of:*

i) removing from P all rules R, such that $I(C_i) \neq 0$ for some negative literal not C_i *in the body of R;*

ii) removing all negative literals from the remaining rules of P.

An interpretation I is a causal stable model *of a program P iff I is the least causal model (Definition 4.2) of the positive program P^I.* $\square$

It is easy to see that Program 4.5 below is the reduct of Program 4.4 with respect to any causal interpretation I holding $I(ab) = 0$. It is also easy to see that it least model I holds $I(fire) = match \cdot f$.

Program 4.5.

$$\begin{array}{llll} f: & fire \leftarrow match & \$: & match \\ ab: & ab \leftarrow \overline{oxygen} & \$: & oxygen \end{array}$$

Theorem 4.4 (Correspondence to non-causal stable models). *Let P be a labelled logic program and P' its standard unlabelled version. Then:*

i) If I is a causal stable model of P, then I^{cl} is a stable model of P'.

ii) If I' is a stable model of P' then there is a unique causal stable model I of P such that $I' = I^{cl}$. $\square$

Proof. The proof can be found in Appendix A on page 261. $\square$

Theorem 4.4 shows a one-to-one correspondence between the causal stable models of a program and the standard stable models when labels are ignored. In other words, the causal stable models of a program are exactly the standard stable models of such program but containing causal information.

Furthermore, since I^{cl} maps any non-zero causal value to 1 (*true*), and so, $I^{cl}(A) = 0$ iff $I(A) = 0$, the reduct of a program with respect to I and I^{cl} is the same. Combining this observation with the result of Theorem 4.4 we immediately get a method for obtaining the causal stable models of a labelled program P:

1. obtain a standard stable model I' of P (ignoring labels),
2. compute the reduct of $P^{I'}$, and
3. compute the least model I of the positive program $P^{I'}$.

Hence, from Theorem 4.4, it holds that $I' = I^{cl}$. Then $P^{I'} = P^{I}$ and, consequently, interpretation I is a causal stable model of P. As an example, consider a program formed by the cycle:

Program 4.6.

$$\begin{array}{lcl} a: & p & \leftarrow not\ q \\ b: & q & \leftarrow not\ p \end{array}$$

This program has two standard stable models $I'_1 = \{p\}$ and $I'_2 = \{q\}$. Furthermore, it is easy to see that the reduct of the above program with respect to I'_1 corresponds to the positive program:

$$a: \quad p \leftarrow$$

whose least causal model is $I_1(p) = a$ and $I_1(q) = 0$. Note that after removing the causal information from I_1 it follows that $I_1^{cl}(p) = 1$ and $I_1^{cl}(q) = 0$. Note also that $I'_1 = \{p\}$ is just the set representation of I_1^{cl} where the elements of the set correspond to those literals with value 1. By a similar reasoning, we may obtain the second causal stable model corresponding to $I_2(p) = 0$ and $I_2(q) = b$.

We may also replicate the standard answer set semantics of Gelfond and Lifschitz [1991]. A literal A is either an atom $A \in At$ or its strong negation$\overline{A}$ with $A \in At$. Note that $\overline{\overline{A}} = A$.

Definition 4.8 (Causal Answer Set). *Given a positive program P, an interpretation I is a* (consistent) answer set *of P iff it is the $\leq$-minimum interpretation such that I satisfies all rules in P and*[1]

i) $I(A) = 0$ or $I(\overline{A}) = 0$ for all literal $A \in Lit$.

Given a normal program P, an interpretation I is an answer set *of P iff I is an answer set of the positive program P^I.*

1 We restrict the definition to consistent answer sets. Inconsistent answer sets can be considered by allowing I to satisfy either i) or ii) $I(A) = 1$ and $I(\overline{A}) = 1$ for all literal $a \in Lit$. A third alternative, that we left for future work, is allowing paraconsistent models in which the causal values will allow us to distinguish which literals are involved in an inconsistency and which do not.

Theorem 4.5 (Correspondence to non-causal answer sets)**.** *Let P be a causal logic program and P' its unlabelled version. Then:*

i) If I is causal answer set of P, then I^{cl} is a answer set of P'.

ii) If I' is a answer set of P' then there is a unique causal answer set I of P such that $I' = I^{cl}$. □

Proof. The proof can be found in Appendix A on page 261. □

Theorem 4.5 shows a one-to-one correspondence between the causal answer sets of a program and the answer sets of its unlabelled version in a similar manner as Theorem 4.4 did for stable models. We may as well extend the notion of proof for normal programs in the following way:

Definition 4.9 (Proof (stable model))**.** *Given a normal program P and a stable model (resp. answer set) I, a* proof *$\pi(A)$ of an atom A is recursively defined as a derivation:*

$$\pi(A) \overset{\text{def}}{=} \frac{\pi(B_1) \ \dots \ \pi(B_m)}{A} \ (R), \tag{50}$$

where $R \in P$ is a rule with head A, $body^+(R) = \{B_1, \dots, B_m\}$ and such that $I \models body^-(R)$. When $m = 0$, the derivation antecedent $\pi(B_1) \ \dots \ \pi(B_m)$ is replaced by $\top$ (corresponding to the empty body). □

Theorem 4.6 (Proof-cause correspondence (stable models))**.** *Given a completely labelled program P and a stable model (resp. answer set) I, π is a non-redundant proof of A iff $cgraph(\pi)$ is a cause of A w.r.t. I.* □

Proof. The proof can be found in Appendix A on page 262. □

Theorem 4.6 extends the result of Theorem 4.2 to normal programs under the stable model and the answer set semantics, showing that causes exactly correspond to the notion of non-redundant proofs in completely labelled programs. Note however that Theorem 4.6 says nothing about the causal values obtained from programs that are not completely labelled. In order to show the behaviour of unlabelled rules consider the following completely labelled program obtained by labelling the unlabelled rule in Program 4.3.

Program 4.7.

$$\begin{array}{llllll} f: & fire & \leftarrow match,\ oxygen & \quad \$: & match \\ d: & oxygen & \leftarrow not\ \overline{oxygen} & \quad \$: & oxygen \end{array}$$

The causal stable model I of Program 4.7 satisfies that:

$$I(fire) \;=\; (match * oxygen) \cdot f \;+\; (match * d) \cdot f$$

As unlabelled rules are just syntactic sugar for rules labelled with the special label '1 :' one would expects that replacing a label by another in a program rule should produce the same effect in the causes that previously contained such label. The following definition formalises this intuition.

Definition 4.10 (Label replacement)**.** *Given a program P, the* replacement *of a label $l \in Lb$ by $t \in Lb \cup \{1\}$, in symbols $P[l \mapsto t]$ is the program obtained by replacing each occurrence of the label l by t.* □

Following with our running example, the result of replacing the label d by 1 in the above program corresponds to Program 4.3 whose unique causal model J satisfies:

$$\begin{aligned} J(fire) \;=\; I(fire)[d \mapsto 1] \;&=\; (match * oxygen) \cdot f + (match * 1) \cdot f \\ &=\; (match * oxygen) \cdot f \;+\; match \cdot f \end{aligned}$$

Note that $(match * oxygen) \cdot f < match \cdot f$ and hence:

$$I(fire)[d \mapsto 1] \;=\; match \cdot f$$

That is, the only cause of the fire corresponds to the value $match \cdot f$ that we had in the beginning of this chapter. We may extend the notion of replacement to causal values and interpretations in the following way:

Definition 4.11 (Graph replacement)**.** *Given a causal graph G, the* replacing *of a label $l \in Lb$ by a label l', in symbols $G[l \mapsto l']$, is the result of replacing l by l' in every edge of G. The removing of a label $l \in Lb$, in symbols $G[l \mapsto 1]$, is the result of removing every edge of the form (v, l) or (l, v) from G.* □

Definition 4.12 (Value and interpretation replacement)**.** *Given a causal value u, a label l and a term $t \in Lb \cup \{1\}$, by $u[l \mapsto t]$ we denote the causal value*

$$\sum \{G[l \mapsto t] \mid G \in U\}$$

Similarly, given an interpretation I, by $I[l \mapsto t]$ we denote an interpretation such that $I[l \mapsto t](A) = I(A)[l \mapsto t]$ for every literal A. □

Theorem 4.7 (Replacement)**.** *Let P be a head labelled program with a unique atom labelling δ. Let $l \in Lb$ be a label and $t \in Lb \cup \{1\}$ not in the image of δ.*

i) If I is a stable model (resp. answer set) of P, then $I[l \mapsto t]$ is a stable model (resp. answer set) of the program $P[l \mapsto t]$.

ii) If I' is a stable model (resp. answer set) of $P[l \mapsto t']$, then there is a unique stable model (resp. answer set) I of P such that $I' = I[l \mapsto t]$.

Proof. The proof can be found in Appendix A on page 267. □

It is worth to mention that unlabelled rules are not only useful for representing defaults, but also for representing parts of a program that we do not want to assign a causal meaning. Suppose, for instance, we want to represent a program with a fact for each natural number and a causal rule stating that each natural number causes p. That is:

$$\begin{aligned} l(0) &: nat(0) \\ l(1) &: nat(1) \\ &\dots \\ a \quad &: p \leftarrow nat(X) \end{aligned}$$

In the least model of this infinite program there are infinitely many causes of p, each one corresponding to one of the addends of the infinite sum $I(p) = l(0) \cdot a + l(1) \cdot a + \dots$.

In standard LP, a finite representation of this program can be achieved by defining the predicate *nat* with a Peano-like representation. Unfortunately, if we assign labels to the rules for *nat* as follows:

$$\begin{aligned} l(0) \qquad &: nat(0) \\ l(X+1) &: nat(X+1) \leftarrow nat(X) \end{aligned}$$

we will obtain for every natural number n that:

$$I(nat(n) = l(0) \cdot l(1) \cdot \dots \cdot l(n)$$

instead of $l(n)$ as we have in the original infinite representation. That is, every natural number n causally depends on $nat(0)$. Hence $l(0) \geq l(0) \cdot l(1) \cdot \dots \cdot l(n)$ and it follows now that $I(p) = l(0)$. That is, $l(0)$ is the only cause of p because any other explanation depends on it.

A way to overcome this issue is by an unlabelled recursive definition using an auxiliary atom q as follows:

$$\begin{aligned} & q(0) \\ & q(X+1) \leftarrow q(X) \\ l(X) : nat(X) \quad & \leftarrow q(X) \end{aligned}$$

Note that now the rules with q in the head do not represent any causal dependence. That is $I(\, q(n)) = 1$ for all natural number n. Therefore it holds that $I(\, nat(n)) = l(n)$ as we intend with in the original infinite representation.

Note furthermore that Theorem 4.7 does not hold in general if δ is not a unique atom labelling. Consider, for instance, the following program.

Program 4.8.

$$\begin{array}{lllll} a : p & \quad c : q & \leftarrow p & \quad e : s & \leftarrow q \\ b : p & \quad d : r & \leftarrow q & \quad f : t & \leftarrow t, s \end{array}$$

whose least model assign the term $u = ((a+b)\cdot c\cdot d * (a+b)\cdot c\cdot e)\cdot f$ to the atom t. We may rewrite u in disjunctive normal form as

$$u \ = \ a\cdot c\cdot d\cdot f * a\cdot c\cdot e\cdot f \ + \ b\cdot c\cdot d\cdot f * b\cdot c\cdot e\cdot f$$

If we remove the label c by u, it follows that

$$u[l \mapsto 1] \ = \ a\cdot d\cdot f * a\cdot e\cdot f \ + \ b\cdot d\cdot f * b\cdot e\cdot f$$

On the other hand, if we unlabelled rule c in Program 4.8, we obtain:

Program 4.9.

$$\begin{array}{lllll} a : p & \quad q & \leftarrow p & \quad e : s & \leftarrow q \\ b : p & \quad d : r & \leftarrow q & \quad f : t & \leftarrow t, s \end{array}$$

whose least model assigns the term $u' = ((a+b)\cdot d * (a+b)\cdot e)\cdot f$ to the literal t. We may as well rewrite u' in disjunctive normal form as:

$$\begin{aligned} u' \ = \ & a\cdot d\cdot f * a\cdot e\cdot f \ + \ b\cdot d\cdot f * b\cdot e\cdot f \ + \\ & a\cdot d\cdot f * b\cdot e\cdot f \ + \ b\cdot d\cdot f * a\cdot e\cdot f \end{aligned}$$

Note that $u[c \mapsto 1] \neq u'$. In fact, u' corresponds to the sum of the four causal graphs depicted in Figure 21, while $u[c \mapsto 1]$ corresponds to the sum of the

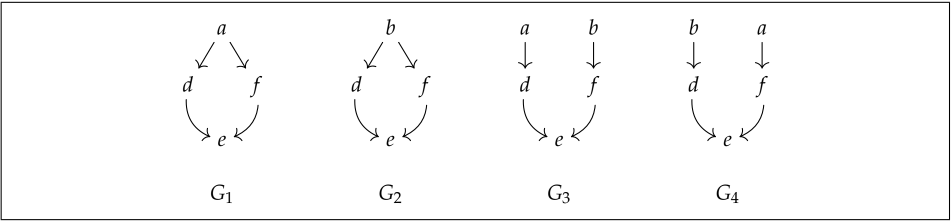

Figure 21: Sufficient causes of t in Program 4.9.

two leftmost graphs. Note that these graphs share the same basic structure than those depicted in Figure 9, with which we illustrate the importance of mapping each literal to a different label.

It is also worth to mention that, according to Theorem 4.7, replacing a label by another label has the effect of collapsing both vertices in all causes. This last effect is useful for representing component that requires several rules to capture its behaviour. For instance, in Program 3.2, representing the circuit of Example 3.1 we have the following pair of rules sharing the same label:

$$d: \quad current(d) \leftarrow current(b)$$
$$d: \quad current(d) \leftarrow current(c)$$

As we commented in Chapter 3, these rules are used to represent a disjunction of the form:

$$d: \quad current(d) \leftarrow current(b) \vee current(c)$$

However, it is important to notice that an arbitrary use of shared labels can be problematic. For instance, in the least model of the following program:

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

it holds that $I(r) = a \cdot b \cdot c$. However, if label c is replaced by a, then $I(r) = a \cdot b \cdot a$. That is, the obtained cause contains a cycle between vertices a and b. At the moment, we do not have any intuition for causes with cycles. They apparently point out an ill causal representation. For instance, in the above program, p and r are considered to be part of the same component a, but q is not, despite of being a causal event in between. Acyclic causes are somehow intuitively expected. Theorem 4.8 below asserts that this intuition holds for any program where rules with different heads have different labels. Of course, this includes programs where each rule has a different label.

Theorem 4.8 (Acyclic causes)**.** *Let P be a head labelled program with a unique atom labelling δ. Then every cause G of any literal A is acyclic, that is, it does not contain (non-reflexive) cycles.* □

Proof. The proof can be found in Appendix A on page 268. □

4.3 CAUSAL WELL-FOUNDED SEMANTICS

We may define the causal counterpart of the *well-founded semantics* [Van Gelder et al., 1991] for normal programs by mimicking the alternating fixpoint definition given by Van Gelder [1989]. Given a program P, we denote by $\Gamma_P : \mathbf{I}_\sigma \longrightarrow \mathbf{I}_\sigma$ an operator that maps each interpretation I to the least model of the positive program P^I. It is easy to see that, as it happened with the standard stable models, the causal stable models of a program P are exactly the fixpoints of the Γ_P operator. It is interesting to note that the Γ_P operator is anti-monotonic, that is, for interpretations I and J such that $I \leq J$ we have that $\Gamma_P(I) \geq \Gamma_P(J)$. Hence the Γ^2_P operator, given by $\Gamma^2_P(I) = \Gamma_P(\Gamma_P(I))$, is monotonic and, from Knaster and Tarski's theorem [Tarski, 1955], it has a least and a greatest fixpoint, respectively denoted by $\mathtt{lfp}(\Gamma^2_P)$ and $\mathtt{gfp}(\Gamma^2_P)$.

Definition 4.13 (Causal well-founded model)**.** *Given a labelled program P, its causal well-founded model is given by the pair $\langle \mathtt{lfp}(\Gamma^2_P), \mathtt{gfp}(\Gamma^2_P)\rangle$.* □

The following result states the relation between the causal well-founded model and the causal stable model in a similar manner as it has happen with the standard well-founded semantics and the stable model semantics.

Proposition 4.1. *Given a labelled program P and a causal stable model I of P, it holds that $\mathtt{lfp}(\Gamma^2_P) \leq I \leq \mathtt{gfp}(\Gamma^2_P)$.* □

Proof. The proof can be found in Appendix A on page 268. □

Theorem 4.9 (Correspondence to non-causal well-founded)**.** *Let P be a labelled program and P' its standard unlabelled version. Then $\mathtt{lfp}(\Gamma^2_{P'}) = \mathtt{lfp}(\Gamma^2_P)^{cl}$ and $\mathtt{gfp}(\Gamma^2_{P'}) = \mathtt{gfp}(\Gamma^2_P)^{cl}$. Moreover, $\langle \Gamma_P(\mathtt{gfp}(\Gamma^2_{P'})), \Gamma_P(\mathtt{lfp}(\Gamma^2_{P'}))\rangle$ is the causal well-founded model of P.* □

Proof. The proof can be found in Appendix A on page 269. □

Theorem 4.9 shows that, as it happened with the stable model semantics, the causal well-founded semantics agrees with the standard well-founded semantics when the causal information is ignored. Note, furthermore, that the causal well-founded model can be obtained just by a single application of the Γ_P operator over its standard well-founded model. Next, we define the entailment of a program in the following way:

Definition 4.14 (Well-founded satisfaction)**.** *A program P* satisfies *an atom A under the well-founded semantics, in symbols $P \models_{wf} A$, when $\texttt{lfp}(\Gamma_P^2) \models A$. On the contrary, P does not satisfy A, in symbols $P \models_{wf} \text{not } A$, when $\texttt{gfp}(\Gamma_P^2) \not\models A$. Otherwise, $\texttt{gfp}(\Gamma_P^2) \models A$ but $\texttt{lfp}(\Gamma_P^2) \not\models A$, the atom A is said to be* undefined *with respect to P.* □

Definition 4.14 allows rewriting the statement of Theorem 4.9 in the following more friendly way.

Corollary 4.2. *Let P be a normal program, P' its standard unlabelled version. Then, $P \models_{wf} L$ iff $P' \models_{wf} L$ for all $L \in \{A, \text{not } A\}$ and any atom A.*

We may also define the sufficient causes under the well-founded semantics in the following way:

Definition 4.15 (Well-founded sufficient cause)**.** *Given a normal program P, a causal graph G is a* (sufficient) cause *of a literal A under the well-founded semantics iff it is a sufficient cause w.r.t. $\texttt{lfp}(\Gamma_P^2)$, in symbols $G \leq_{\max} \texttt{lfp}(\Gamma_P^2)(A)$.* □

Theorem 4.10 (Proof-cause correspondence (well-founded))**.** *Given a completely labelled program P with a unique atom labelling δ, a proof π is a non-redundant proof of A with respect to $\texttt{gfp}(\Gamma_P^2)$ and P (Definition 4.9) iff $cgraph(\pi)$ is a cause of A under the well-founded semantics.* □

Proof. The proof can be found in Appendix A on page 269. □

Theorem 4.10 shows the correspondence between causes and syntactic proofs under the well-founded semantics.

Note that from Proposition 4.1 and Definition 4.15 it follows that if G is not a cause of some literal A under the well-founded semantics, then G is not a cause of A w.r.t. to any stable model of the program. However, if G is a cause of some literal A under the well-founded semantics, this does not imply that G is a cause of A w.r.t. to any stable model of the program. Consider, for instance, the following program:

Program 4.10.

$$\begin{array}{ll} a: & p \\ & q \leftarrow \mathit{not}\ r \\ & r \leftarrow \mathit{not}\ q \\ & p \leftarrow q \\ & p \leftarrow r \end{array}$$

whose well-founded causal model corresponds to a pair $\langle \mathtt{lfp}(\Gamma_P^2), \mathtt{gfp}(\Gamma_P^2) \rangle$ such that $\mathtt{lfp}(\Gamma^2)(p) = a$ and $\mathtt{gfp}(\Gamma^2)(p) = 1$ and whose two causal stable models hold $I(p) = 1$. Then a is a cause of p under the well-founded semantics, but it is not with respect any of the stable models of the program because a is not $\leq$-maximal with respect to any of them. This example shows that we can use the causal well-founded semantics to rule out some possible causes, but we may only assert that a causal graph is a cause w.r.t to any/some causal stable model if it is a cause w.r.t. both $\mathtt{lfp}(\Gamma_P^2)$ and $\mathtt{gfp}(\Gamma_P^2)$ components of the well-founded causal model.

5 ACTIONS DOMAINS EXAMPLES

Previous chapters have formalised causal semantics for normal programs that are conservative extensions of the *stable model*, the *well-founded model* and the *answer set semantics*. These semantics allow us to capture, not only the truth of each literal, but also the causes for those truth values. The (standard) *answer set semantics* has arisen as suitable tool for representing and reasoning about actions in dynamic domains which, in its turn, has been one of the central areas in knowledge representation. Research in this area has been focused on two interesting topics from the perspective of this thesis. First, the use of inertia, a commonsense law that provides an example of dynamic default, i.e., a default whose behaviour is not just fixed by a single value, but depends on time. Representing inertia has been a crucial feature for solving the *frame problem*. Second, the interplay between inertia and direct or indirect effects of actions, which gave rise to different causal approaches used in AI and mentioned in the introduction. This chapter will show that the *causal answer set semantics* can be used for representing some of the traditional examples of reasoning about actions. For that purpose the common sense law of inertia needs to propagate not only the truth value of the fluents but also the causes explaining it.

As usual for action domains, we will distinguish three sorts of logic object: *fluents*, *actions* and *situations*. We will represent situations as non-negative integers 0,1,2,... Situation 0 represents the *initial situation*. Every fluent F has an associated set of values called the domain of F, in symbols $Dom(F)$. A *fluent literal* is an expression of the form $F = v$ where F is a fluent and v is a value in the domain of F. When $Dom(F) = \{\mathbf{t}, \mathbf{f}\}$ we say that F is *Boolean* and we usually write F and $\overline{F}$ instead of $F = \mathbf{t}$ and $F = \mathbf{f}$ respectively. We will write $F_s = v$ to represent that a fluent F holds the value v in the situation s and A_s to represent that the action A occurs at s. When a fluent is Boolean we will write F_s and $\overline{F}_s$. Note also that the fact that the action A does not occur at s will be represented by the absence of A_s instead of a literal $\overline{A}_s$. Furthermore, for a simpler reading we will use a high level representation of causal laws similar to the one used in action language $\mathcal{A}$ [Gelfond and Lifschitz, 1993]. In our case, causal laws are written as formulas of the form:

$$l_R: \ B_1,\ldots,B_m \ \textbf{causes} \ A \ \textbf{unless} \ C_1,\ldots,C_n \tag{71}$$

where A, B_i's and C_j's are literals. We will assume that A and all C_j's are fluent literals, while B_i's may be either fluent literals or actions. A causal law like (71) in which all B_i's are fluent literals is called an *state constraint*. Otherwise, it is called a *effect axiom*. A causal law of the form (71) may be read as "$B_1,\ldots,B_m$ together causes A to occur unless some abnormality among $C_1,\ldots,C_n$ occur." Thus, a causal law like (71) has the following natural translation into a logic program rule:

$$l_{R,s}: \ A_{s+1} \leftarrow B_{1,s}, \ldots, B_{m,s}, \ not \ C_{1,s}, \ldots, \ not \ C_{n,s} \tag{48}$$

for each situation s. However, it is interesting to keep this high level representation for observing how slightly different translations of the same causal laws (and different implementations of the commonsense law of inertia) lead to different derived causal information, in spite of sharing the same behaviour with respect to the standard truth values. In addition, we will state that a fluent F is respectively true or false in the initial situation by formulas of the form:

$$\textbf{initially} \ F \tag{72}$$

$$\textbf{initially} \ \overline{F} \tag{73}$$

In the rest of the chapter, we revisit some traditional examples in the literature of reasoning about actions in dynamic domains.

5.1 YALE SHOOTING SCENARIO: THREE POSSIBLE REPRESENTATIONS

Consider, as a first example, the Yale Shooting scenario introduced by Hanks and McDermott [1987]:

Example 5.1 (Yale Shooting Scenario)**.** *There is a turkey called Fred that we try to kill. Shooting a loaded gun will kill it. We load the gun, wait a situation and then shoot.* □

We may capture this scenario by a pair of causal laws of the form:

$$l: \quad \mathit{load}\ \textbf{causes}\ \mathit{loaded} \tag{74}$$
$$d: \quad \mathit{loaded}, \mathit{shoot}\ \textbf{causes}\ \mathit{dead} \tag{75}$$

together with the sequence of actions: $\mathit{load}_1, \mathit{wait}_2, \mathit{shoot}_3$ stating that, first, the gun is loaded and, after waiting a transition, it is shot. We also use laws of the form:

$$\textbf{initially}\ \overline{\mathit{loaded}} \tag{76}$$
$$\textbf{initially}\ \overline{\mathit{dead}} \tag{77}$$

to represent that, initially, the gun is unloaded and Fred is alive.

In order to represent these causal laws as a logic program, on the one hand, causal laws (74) and (75) are respectively translated as:

$$l_{s+1} : \mathit{loaded}_{s+1} \leftarrow \mathit{load}_s$$
$$d_{s+1} : \mathit{dead}_{s+1} \leftarrow \mathit{loaded}_s, \mathit{shoot}_s$$

On the other hand, initial conditions are represented by labelled facts at situation 0. For instance, (76) and (77) are represented by:

$$\$: \quad \overline{\mathit{loaded}}_0$$
$$\$: \quad \overline{\mathit{dead}}_0$$

Finally, inertia is represented as the pattern:

$$F_{s+1} \leftarrow F_s, \mathit{not}\ \overline{F}_{s+1} \tag{78}$$

for each fluent literal $F \in \{\mathit{loaded}, \mathit{dead}, \overline{\mathit{loaded}}, \overline{\mathit{dead}}\}$. Recall that $\overline{F}$ denotes the strong negation of F and that $\overline{\overline{F}} = F$. The scenario in the example consists of the above program plus the following labelled facts representing the actions that have been performed:

$$\$: \quad \mathit{load}_1$$
$$\$: \quad \mathit{wait}_2$$
$$\$: \quad \mathit{shoot}_3$$

With respect to Lifschitz [2002], we have just added labels to all facts describing the initial situation and the actions occurrences, and a label per each causal law.

Furthermore, it is easy to see that, when we ignore the labels, this program has a unique standard answer set. As seen in Chapter 4, this implies that it also has a unique causal answer set I. It is also easy to see that the causal values associated to literals representing the initial situation and actions just correspond to the labels of their corresponding facts. For instance, for atom $load_1$, we have $I(load_1) = load_1$. Then, the application of rule l_2 implies that:

$$I(loaded_2) = load_1 \cdot l_2$$

That is, loading the gun in the first situation has caused the gun to be loaded at the second situation by means of the causal law l applied at step 2.

As in standard answer set semantics, the inertia axiom:

$$loaded_{s+1} \leftarrow loaded_s, \, not \; \overline{loaded}_{s+1} \tag{79}$$

propagates the fact that the gun is loaded from situation 2 to situation 3. In a similar manner, in the causal answer set semantics, (79) will also propagate the causes explaining why the gun is loaded. By the application of (79) with $s = 2$, it follows that:

$$I(loaded_3) \;=\; load_1 \cdot l_2 \cdot 1$$

Recall that unlabelled rules are syntactic sugar for rules with label '1 :' and that causal value 1 is the application identity. Therefore, for the atom $loaded_3$, it follows that $I(loaded_3) = load_1 \cdot l_2$. Finally, the application of rule d_3 similarly implies:

$$I(shoot_4) \;=\; (load_1 \cdot l_2 * shoot_3) \cdot d_4$$

That is, shooting the gun together with loading it has been the cause of Fred's death by means of causal laws l and d respectively applied at steps 2 and 4. The causal value $I(shoot_4) = (load_1 \cdot l_2 * shoot_3) \cdot d_4$ corresponds to the graph depicted in Figure 22.

Consider now the following variation of the Yale Shooting scenario which incorporates a second shooter that acts after the first one.

Example 5.2 (Two shooters)**.** *There is a turkey called Fred that we try to kill. Shooting a loaded gun will kill it. There are also two shooters, Suzy and Billy, who load the gun, wait a situation and then shoot. However, Billy performs all his actions one situation later than Suzy.* □

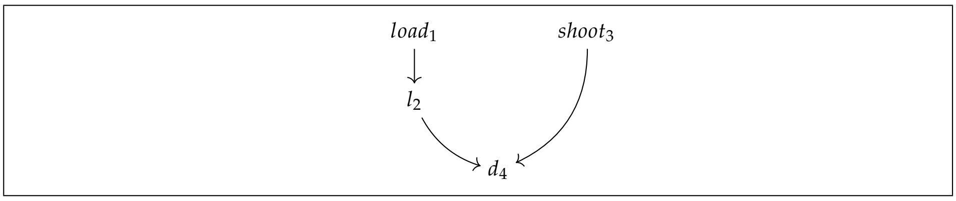
$$\begin{tikzcd} load_1 \arrow[d] & & shoot_3 \arrow[dl, bend left] \\ l_2 \arrow[r, bend right] & d_4 & \end{tikzcd}$$

Figure 22: Sufficient cause of *dead* in the Yale Shooting scenario.

In order to incorporate a second shooter we need a slight modification of the causal laws (74) and (75) by incorporating a variable A representing the agent — the shooter — that performs the actions:

$$l(A): \quad load(A) \ \textbf{causes} \ loaded(A) \tag{80}$$
$$d \ : \quad loaded(A), shoot(A) \ \textbf{causes} \ dead \tag{81}$$

Furthermore, we will have the following narrative of actions:

$$load(suzy)_1 \qquad load(billy)_2 \qquad shoot(suzy)_3 \qquad shoot(billy)_4$$

Note that we just omit the *wait* actions because they are irrelevant in the formalisations that we are analysing. The translation of these causal laws is the following one::

$$\begin{aligned} l(A)_{s+1}: &\quad loaded(A)_{s+1} \leftarrow load(A)_s \\ d_{s+1} \ : &\quad dead_{s+1} \qquad\ \leftarrow loaded(A)_s, shoot(A)_s \end{aligned}$$

plus the corresponding inertia axioms. As happened in the example with a single shooter, the unique causal model of this program satisfies that:

$$I(dead_4) \ = \ (load(suzy)_1 \cdot l(suzy)_2 * s(suzy)_3) \cdot d_4 \tag{82}$$

We have just added the agent constant *suzy* to the corresponding labels, that is, we write $load(suzy)_1$, $l(suzy)_1$ and $s(suzy)_3$ instead of $load_1$, l_1 and s_3. Furthermore, by the actions of Billy and the causal law d_s, a model I must also satisfy:

$$\begin{aligned} I(dead_5) \ = \ & (load(suzy)_1 \cdot l(suzy)_2 * s(suzy)_3) \cdot d_4 \\ & \qquad + (load(billy)_2 \cdot o(billy)_3 * s(billy)_4) \cdot d_5 \end{aligned}$$

As we have seen in Chapter 3, causal values can be represented as sums (or sets) of graphs. In our running example, $I(dead_5)$ corresponds to the causal

$$\begin{tikzcd}
load(suzy)_1 \arrow[d] & & shoot(suzy)_3 \arrow[ddl, bend left] & & load(billy)_2 \arrow[d] & & shoot(billy)_4 \arrow[ddl, bend left] \\
l(suzy)_2 \arrow[dr, bend right] & & & & o(billy)_3 \arrow[dr, bend right] & & \\
& d_4 & & & & d_5 & \\
& G_1 & & & & G_2 &
\end{tikzcd}$$

Figure 23: Sufficient causes G_1 and G_2 of the dead in the Yale Shooting scenario with two shooters.

graphs G_1 and G_2 depicted in Figure 23. Hence, we may rewrite the value corresponding to $dead_5$ as $G_1 + G_2$. That is, both shooters are considered equally responsible of killing Fred.

Examples involving death, like many others in the causal literature, share a dependence with respect to time: once the effect has been caused — in this case the death of Fred — it cannot be caused again. In our running example, this idea can be captured by replacing rule d_s by:

$$d_s : \quad dead_{s+1} \leftarrow loaded(A)_s, shoot(A)_s, not\ dead_s \tag{83}$$

Since $dead_4$ holds, this rule is not applicable to the actions performed by any second shooter. Hence, the only cause of $dead$, in this variation, is G_1, that is $I(dead) = G_1$. Furthermore, (83) is the direct translation of a causal law of the form:

$$loaded, shoot\ \textbf{causes}\ dead\ \textbf{unless}\ dead \tag{84}$$

However, we prefer retain the same high lever representation (that is, causal law (84)) and assign different behaviours to fluents. In this sense, we may classify the possible ways in which a previous cause can be propagated by into three main categories characterised as which, the causal laws or the inertia, takes preference:

1. *Symmetric*: explanations coming from inertia and from causal laws at the current situation are equally "valid" causes.
2. *Inertial preference*: Explanations coming from inertia are preferred to those that could potentially come from causal laws at that situation.
3. *Causal preference*: Explanation coming from causal laws are preferred to those that could potentially come from inertia.

In order to represent the third behaviour, causal preference, we introduce a new predicate $noninertial(F)$ that means that fluent F does not follow the inertia law[1]. Hence, for representing the causal preference behaviour we will add the following to rules two our representation:

$$\begin{aligned} noninertial(loaded)_{s+1} &\leftarrow load_s \\ noninertial(dead)_{s+1} &\leftarrow loaded_s, shoot_s \end{aligned}$$

and replace the previous inertia axioms in the following way:

$$F_{s+1} \leftarrow F_s, not\ noninertial(F)_{s+1} \tag{85}$$

In the same way as has happen in the symmetric and inertial preference behaviour representations, the unique causal stable model I must hold (82) and

$$I(dead_5) \ \geq \ (load(billy)_2 \cdot l(billy)_3 * s(billy)_4) \cdot d_5 \ = \ G_2$$

However, since $I(noninertial(dead_5)) \neq 0$ the inertia axiom with $dead_5$ in the head is not in the reduct of the program. Therefore $I(dead_5) = G_2$, which coincides with the idea that explanations coming from causal laws are preferred over inertial ones.

5.2 THE BLOCK'S WORLD SCENARIO

Consider the block world scenario from Lifschitz [2002]. This scenario illustrates the importance of using the notation $on(B) = L$, instead of $on(B, L)$, to clearly distinguish that the location L is a function of the block B and not the other way around.

Example 5.3 (Block's World). *Imagine that blocks are moved by a robot with several grippers, so that a few blocks can be moved simultaneously. However, the robot is unable to move a block onto a block that is being moved at the same time. As usual in blocks world planning, we assume that a block can be moved only if there are not blocks on top of it.*

1 Technically, this predicate is not different from Lin's *caused* or Shanahan's *occluded* predicate [Lin, 1995, Shanahan, 1999]. For the sake of clarity, we have preferred not to overload the word "caused" in the current contest and use the word "noninertial" as it better reflects the technical purpose of this predicate: prevent the application of inertia for that fluent at that situation.

We may represent Example 5.3 by the following causal law:

$$m(B,L): \quad move(B,L) \ \textbf{causes} \ on(B) = L \tag{86}$$

We will also assume that fluent $on(B)$ follows the *inertial preference behaviour*, so that a block cannot newly be caused to be in the same position in which it already is. Then (86) will be translated as the following pair of rules:

$$\begin{array}{lll} m(B,L)_{s+1}: & on(B,L)_{s+1} & \leftarrow move(B,L)_s, not\ on(B,L)_s \\ & noninertial(on(B))_{s+1} & \leftarrow move(B,L)_s, not\ on(B,L)_s \end{array}$$

plus the following rules to represent the inertia axiom:

$$on(B,L)_{s+1} \leftarrow on(B,L)_s, not\ noninertial(on(B))_{s+1}$$

The notation $on(B) = L$ is important to differentiate the fluent value L from the rest of arguments (in this case B) so that inertia can be properly formalised. In this way, $noninertial(on((B))_{s+1}$ is used to disable the application of all inertial laws for each possible value of fluent $on(B)$.

Domain constraints can be stated as usual in ASP:

$$\leftarrow on(B_1,B)_s, on(B_2,B)_s, B_1 \neq B_2, block(B)$$

states that two blocks cannot be on the same block,

$$\leftarrow move(B_1,L)_s, on(B_2,B_1)_s$$

states that a block B_1 cannot be moved when another block is on it, and

$$\leftarrow move(B_1,B_2)_s, move(B_2,L)_s$$

states that a block cannot be moved onto a block that is being moved also.

Then, the causal information can be obtained in a similar manner as in the Yale Shooting Scenario. If we suppose that there are three blocks a, b and c which are initially on the table, that is, we add the following facts

$$\begin{array}{l} on(a,table)_0 \\ on(b,table)_0 \\ on(c,table)_0 \end{array}$$

and then we move the block b on top of a, wait and move the block c on top of b, that is we add the actions

$$\begin{aligned} \$:& \quad move(b,a)_0 \\ \$:& \quad move(c,b)_2 \end{aligned}$$

we will obtain that block b is on top of a because of:

$$I(on(b,a)_1) \;=\; move(b,a)_0 \cdot m(b,a)_1$$

and that block c is on top of b because of:

$$I(on(c,b)_3) \;=\; move(c,b)_2 \cdot m(c,b)_3$$

We may also define a predicate $over(B_1,B_2)_s$ as follows:

$$\begin{array}{lll} o(B_1,B_2) : on(B_1,B_2) & \textbf{causes} & over(B_1,B_2) \\ p(B_1,B_3) : over(B_1,B_2), over(B_2,B_3) & \textbf{causes} & over(B_1,B_3) \end{array}$$

Hence, we will conduce that:

$$\begin{aligned} I(over(c,a)_3) \;=\; & \big(move(b,a)_0 \cdot m(b,a)_1 \cdot o(b,a)_1 * \\ & \qquad move(c,b)_2 \cdot m(c,b)_3 \cdot o(c,b)_3\big) \cdot p(a,c)_3 \end{aligned}$$

That is, the block c is over the block a because we have move the block b on top of a and the block c on top of b.

5.3 LIN'S SUITCASE SCENARIO AND THE INDIRECT EFFECTS OF ACTIONS

Another interesting example is the suitcase scenario of Example 1.1 in the introduction. We may represent this scenario by the following causal laws:

$$o \quad : \quad up(a), up(b) \;\textbf{causes}\; open \tag{87}$$

$$u(L) : \quad lift(L) \;\textbf{causes}\; up(L) \tag{88}$$

The interest of this example relies on the presence of the indirect effect expressed by causal law (87). Representing the indirect effects of actions is the core of the so called *ramification problem* [Kautz, 1986]. We may represent the indirect effects of actions in a similar manner as we did with direct effects. Suppose that all fluents follow the *symmetric behaviour*. In such case, causal laws (87) and (88) will be simply written as:

$$\begin{aligned} o_s \quad &: \quad open_s &&\leftarrow up(a)_s, up(b)_s \\ u(L)_{s+1} &: \quad up(L)_{s+1} &&\leftarrow lift(L)_s, not\ up(L)_s \end{aligned}$$

It is perhaps worth to mention that, as in most action domains representations, direct effects are located in the *next state* after the action execution, whereas indirect effects are located in the *same state* than their preconditions. Suppose that both locks are lifted with a waiting situation in between. That is, we add the following labelled facts to the above rules:

$$\begin{aligned} \$ &: \quad lift(a)_1 \\ \$ &: \quad lift(b)_3 \end{aligned}$$

Similarly to the Yale Shooting Scenario, it can be checked that the following equations holds: $I(up(a)_4) = lift(a)_1 \cdot u(a)_2$ and $I(up(b)_4) = lift(b)_3 \cdot u(b)_4$. Consequently, we obtain that:

$$I(open_4) \quad = \quad \big(lift(a)_1 \cdot u(a)_2 * lift(b)_3 \cdot u(b)_4\big) \cdot o_4 \quad = \quad G_1$$

which corresponds to the graph G_1 depicted in Figure 24.

$$\begin{tikzcd}
lift(a)_1 \arrow[d] & & lift(b)_3 \arrow[d] \\
u(a)_2 \arrow[dr, bend right] & & u(b)_4 \arrow[dl, bend left] \\
& o_4 &
\end{tikzcd}$$

Figure 24: Sufficient cause G_1 of $open$ in the suitcase scenario.

Consider now the variation of this example stated by Example 1.4. This variation introduces a second opening mechanism activated by a key. That is, we add a causal law of the form:

$$k: \quad key\ \textbf{causes}\ open \tag{89}$$

to (87) and (88). We will use this example to illustrate how the inertial and causal preference behaviours work with indirect effects. Suppose that we try to open the suitcase with the key after the second lock is lifted, that is, we add the labelled fact $(\$: key_4)$. Under the inertial preference behaviour, the causal law (89) is translated as:

$$k_{s+1} : \quad open_{s+1} \leftarrow key_s, not\ open_s$$

As we have seen above $I(open_4) = G_1 \neq 0$, therefore rule k_4 is not in the reduct of the program and, consequently, $open_5$ only inherits its causal value from $open_4$ by the inertia axiom. That is $I(open_5) = I(open_4) = G_1$. In other words, using the key does not yield a new cause for open, because the suitcase was *already open*.

On the other hand, under the causal preference behaviour, the causal law (89) is translated as:

$$\begin{aligned} k_{s+1} : \quad & open_{s+1} && \leftarrow key_s \\ & noninertial(open)_{s+1} && \leftarrow key_s \end{aligned}$$

While inertia for *open* has now the form:

$$open_{s+1} \leftarrow open_s, not\ noninertial(open)_{s+1}$$

Since $I(noninertial(open)_5) \neq 0$, the inertia axiom is disabled, and consequently $I(open_5) = key_4 \cdot k_5$. In other words, we have preferred the explanation provided by the *last* action, even though the suitcase was already open.

Finally, in the case where indirect effects (like *open* in this scenario) follow the symmetrical behaviour we will require a further elaboration which is left for future work and will be briefly commented in Chapter 6.

5.4 THE GEAR WHEELS SCENARIO

The gear wheels scenario introduced by McCain [1997] is one of the benchmark problems for analysing how causal cycles behave in different action representations.

Example 5.4 (The gear wheel). *Consider a gear mechanism with a pair of wheels, each one powered by a separate motor. Each motor has a switch to start and stop it. There is another switch to connect or disconnect the wheels.* □

A static version of this scenario was studied in Chapter 4. We will represent now the dynamic counterpart by the following set of direct effect causal laws:

$$\begin{array}{lll} m(W): & start(W) & \textbf{causes } motor(W) \\ \overline{m}(W): & stop(W) & \textbf{causes } \overline{motor}(W) \\ p \quad : & couple & \textbf{causes } coupled \\ \overline{p} \quad : & uncouple & \textbf{causes } \overline{coupled} \end{array}$$

plus the following set of indirect effect causal laws:

$$\begin{array}{lllll} r(W): & motor(W) & \textbf{causes } spinning(W) & & \\ t(a) \ : & spinning(b) & \textbf{causes } spinning(a) & \textbf{unless} & \overline{coupled} \\ t(b) \ : & spinning(a) & \textbf{causes } spinning(b) & \textbf{unless} & \overline{coupled} \\ \overline{t}(a) \ : & \overline{spinning}(b) & \textbf{causes } \overline{spinning}(a) & \textbf{unless} & \overline{coupled} \\ \overline{t}(b) \ : & \overline{spinning}(a) & \textbf{causes } \overline{spinning}(b) & \textbf{unless} & \overline{coupled} \end{array}$$

Suppose that, initially, both motors are off, the wheels are not spinning and uncoupled. Assume also that all fluents follow the inertial preference behaviour. Initial conditions will be represented by the following labelled facts:

$$\begin{array}{llllll} \$: & \overline{motor}(a)_0 & \$: & \overline{spinning}(a)_0 & \$: & \overline{coupled}_0 \\ \$: & \overline{motor}(b)_0 & \$: & \overline{spinning}(b)_0 & & \end{array}$$

We also assume the following narrative of actions:

$$\begin{array}{ll} \$: & start(a)_1 \\ \$: & couple_3 \\ \$: & uncouple_5 \end{array}$$

That is, $motor(a)$ is started at situation 1 leading the first wheel to spin at situation 2, that is $I(spinning(a)_2) = start(a)_1 \cdot r(a)_2$. Since the wheels are uncoupled, the second one is still by inertia, that is, it keeps its initial value $I(\overline{spinning}(b)_2) = \overline{spinning}(b)_0$. At situation 3, both wheels are coupled, leading to:

$$I(spinning(b)_4) \;=\; start(a)_1 \cdot r(a)_2 \cdot t(b)_4 \;=\; G_1$$

where G_1 is the graph depicted in Figure 25. Then, at situation 5 the wheels are uncoupled again. The second wheel will still spin forever by inertia and, therefore, its causal value does not change. That is, I must satisfy $I(spinning(2)_s) =$

$$\begin{tikzcd}
start(a)_1 \arrow[d] & & start(a)_1 \arrow[d] & & couple_3 \arrow[d] \\
r(a)_2 \arrow[d] & & r(a)_2 \arrow[dr, bend right] & & p_4 \arrow[dl, bend left] \\
t(b)_4 & & & t(b)_4 & \\
G_1 & & & G_2 &
\end{tikzcd}$$

Figure 25: Sufficient causes of $spinning(b)$ in the two variation of the gear wheel scenario.

$I(spinning(2)_4) = G_1$ for all $s \geq 4$. Notice that we have considered $\overline{coupled}$ to be an exception to rules $t(X)$ and $\bar{t}(X)$ with $X \in \{a,b\}$ and, then, $coupled$ is not reflected in the cause of $spinning(2)$. If we considered that $coupled$ is a precondition of those rules, that is, we replace them by:

$$\begin{array}{lll}
r(W): & motor(W) & \textbf{causes } spinning(W) \\
t(a) \;\; : & spinning(b) & \textbf{causes } spinning(a), coupled \\
t(b) \;\; : & spinning(a) & \textbf{causes } spinning(b), coupled \\
\bar{t}(a) \;\; : & \overline{spinning}(b) & \textbf{causes } \overline{spinning}(a), coupled \\
\bar{t}(b) \;\; : & \overline{spinning}(a) & \textbf{causes } \overline{spinning}(b), coupled
\end{array}$$

we will obtain instead that:

$$I(spinning(b)_4) \quad = \quad \big(start(a)_1 \cdot r(a)_2 * couple_3 \cdot p_4\big) \cdot t(b)_4 \quad = \quad G_2$$

where G_2 is the graph depicted in Figure 25. That is, $coupled$ is now part of the cause of $spinning(2)_4$.

An interesting variation of this example is incorporating some mechanic device that allows stopping the wheels [Van Belleghem et al., 1998, Lin and Soutchanski, 2011]. This can be achieved by adding the following causal laws:

$$\begin{array}{lll}
b(W): & brake(W) & \textbf{causes } braked(W) \\
\bar{b}(W): & unbrake(W) & \textbf{causes } \overline{braked}(W) \\
\bar{r}(W): & braked(W) & \textbf{causes } \overline{spinning}(W)
\end{array}$$

If the second wheel is braked at s_5, we will get instead that $spinning(b)_6$ is false. The cause of $\overline{spinning}(b)_s$ in any future situation $s \geq 6$ is:

$$brake(b)_5 \cdot b(b)_6 \cdot \bar{r}(b)_6$$

Since both wheels are uncoupled, the first wheel will go on spinning.

6 CAUSAL LITERALS

The previous chapter has shown how to derive causal information from action scenarios. A natural question is whether this information can be used, in its turn, to derive new conclusions. To put an example, take a statement like "whoever causes a bomb explosion will be punished with imprisonment" (Example 1.3). As commented in the introduction, representing this kind of knowledge requires a new kind of literal of the form $hascaused(A,B)$ that captures the idea that "event A has been sufficient to cause event B." We may define the precise semantics of this new kind of literal by inspecting the causal value of B to check whether A is one of its sufficient causes or not. If this kind of literal is available, we may represent the above statement by a causal law of the form:

$$p: \quad \textbf{hascaused}(A,bomb) \ \ \textbf{causes} \ prison(A)$$

For a proper representation, we add an argument A to action lift to represent the agent performing the action:

$$u: \quad lift(A,L) \ \textbf{causes} \ up(L) \tag{90}$$

and actions are now represented by the labelled facts:

$$\$: \quad lift(billy,a)_1$$
$$\$: \quad lift(billy,b)_3$$

and we also add the law:

$$b: \quad open \ \textbf{causes} \ bomb$$

If we assume that all fluents follow the inertia preference behaviour, these causal laws will be translated into the following program:

Program 6.1.

$$
\begin{array}{lcll}
p_s & : & prison(A) & \leftarrow hascaused(A,bomb) \\
b_s & : & bomb_s & \leftarrow open_s, not\ bomb_{s-1} \\
o_s & : & open_s & \leftarrow up(a)_s, up(b)_s not\ open_{s-1} \\
u(L)_{s+1} & : & up(L)_{s+1} & \leftarrow lift(L)_s, not\ up(L)_s \\
\$ & : & lift(billy,a)_1 & \\
\$ & : & lift(billy,b)_3 &
\end{array}
$$

Then, the causal value of $bomb_5$ corresponds to:

$$I(bomb_5) \quad = \quad \big(lift(billy,a)_1 \cdot u(a)_2 * lift(billy,b)_3 \cdot u(b)_4\big) \cdot o_4 \cdot b_4$$

which, in its turn, corresponds to the graph G_1 depicted in Figure 26.

$$\begin{tikzcd}
lift(billy,a)_1 \arrow[d] & & lift(billy,b)_3 \arrow[d] \\
u(a)_2 \arrow[dr, bend right] & & u(b)_4 \arrow[dl, bend left] \\
& o_4 \arrow[d] & \\
& b_4 &
\end{tikzcd}$$

Figure 26: Sufficient cause G_1 of *bomb* in the suitcase scenario.

Intuitively $\mathbf{hascaused}(billy,bomb)$ will hold when all actions which are the source of every causal chain in the graph have been performed by Billy. In fact, this is the case in our running example.

6.1 FORMALIZATION OF CAUSAL LITERALS

Definition 6.1 (Causal literal). *A* (causal) literal *is a formula of the form:*

$$(\psi :: A) \tag{91}$$

where $A \in At$ is a standard atom and $\psi : \mathbf{C}_{Lb} \longrightarrow \{0,1\}$ is a monotonic function. □

Intuitively ψ is a function that acts as a query over the causes of the standard atom A: it is evaluated to 1 if a cause G is an answer of ψ and to 0 otherwise. In other words, a causal literal of the form $(\psi :: A)$ would be satisfied iff there was some cause G of A which is an answer of ψ. By selecting an appropriate query function ψ we may use causal literals in order to define particular kinds of literals. For instance, let $\mathcal{A} \subseteq Lb$ be a set of actions labels and $\psi_{\mathcal{A}} : \mathbf{C}_{Lb} \longrightarrow \mathbf{C}_{Lb}$ be a function such that $\psi_{\mathcal{A}}(G)$ is the result of removing all labels non corresponding to an action in $\mathcal{A}$. Formally:

$$\psi_{\mathcal{A}}(G) \stackrel{\text{def}}{=} G[l_1 \mapsto 1, \ldots, l_n \mapsto 1]$$

where $Lb \backslash \mathcal{A} = \{l_1, \ldots, l_n\}$ and $G[l_1 \mapsto 1, \ldots, l_n \mapsto 1]$ is label replacement (Definition 4.10).

Definition 6.2 (Hascaused predicate). *Given a pair of sets of action labels $\mathcal{A}$ and $\mathcal{A}_a$ such that $\mathcal{A}_a \subseteq \mathcal{A} \subseteq Lb$, and a standard atom A, by "**hascaused**$(\mathcal{A}_a, A)$" we denote a causal literal of the form $(\psi_{\mathcal{A}_a} :: A)$ such that $\psi_{\mathcal{A}_a}(G) \stackrel{\text{def}}{=} 1$ iff*

$$\prod \mathcal{A}_a \ \leq \ \psi_{\mathcal{A}}(G)$$

and $\psi_{\mathcal{A}_a}(G) \stackrel{\text{def}}{=} 0$ otherwise. □

Intuitively, the set $\mathcal{A}_a$ represents the actions performed by an agent a. For instance, in the suitcase-bomb scenario, the literal **hascaused**$(billy, bomb)$ can be captured by a set of label actions:

$$billy \ \stackrel{\text{def}}{=} \ \{\ lift(billy,a)_1,\ lift(billy,b)_3\ \}$$

capturing all actions performed by Billy. Then, it is easy to see now that $\prod billy \ = \ lift(billy,a)_1 * lift(billy,b)_3$. Furthermore, it can be shown that:

$$\begin{aligned}\psi_{\mathcal{A}}(G_1) \ &= \ \big(lift(billy,a)_1 \cdot 1 * lift(billy,b)_3 \cdot 1\big) \cdot 1 \cdot 1 \\ &= \ lift(billy,a)_1 \ * \ lift(billy,b)_3\end{aligned}$$

and, thus:

$$\prod billy \ \leq \ \psi_{\mathcal{A}}(G_1)$$

That is, $\psi_{billy}(G_1) = 1$ and the unique causal stable model I satisfies the causal literal **hascaused**$(billy, bomb)$. If we are now told that Suzy has lifted both locks in the first situation, that is, the following pair of labelled facts are added to the program:

$$\begin{aligned} \$: &\quad \mathit{lift}(\mathit{suzy},a)_1 \\ \$: &\quad \mathit{lift}(\mathit{suzy},b)_1 \end{aligned}$$

and we assume that the fluent *bomb* follows the *inertial preference behaviour*, then the cause of bomb_5 corresponds now to the graph G_2 in Figure 27. In

$$\begin{tikzcd}
\mathit{lift}(\mathit{suzy},a)_1 \arrow[d] & & \mathit{lift}(\mathit{suzy},b)_1 \arrow[d] \\
u(a)_2 \arrow[dr, bend right] & & u(b)_2 \arrow[dl, bend left] \\
& o_2 \arrow[d] & \\
& b_2 &
\end{tikzcd}$$

Figure 27: Sufficient cause G_1 of *bomb* in the suitcase scenario.

this modified scenario, $\psi_{\mathcal{A}_{billy}}(G_2) = 0$, and so, in the presence of the new evidence, we obtain that Billy was not who has caused the bomb explosion. If fact, $\psi_{suzy}(G_2) = 1$ points out that it was Suzy who has caused it.

Definition 6.3 (Causal literal valuation)**.** *The* valuation of a causal literal *of the form* $(\psi :: A)$ *is given by:*

$$I(\psi :: A) \;\overset{\text{def}}{=}\; \sum \{\, G \in \mathbf{C}_{Lb} \mid G \leq I(A) \text{ and } \psi(G) = 1 \,\}$$

for any interpretation $I : At \longrightarrow \mathbf{V}_{Lb}$*. We say that* I *satisfy a causal literal* $(\psi :: A)$*, in symbols* $I \models (\psi :: A)$*, iff* $I(\psi :: A) \neq 0$. □

A causal literal, as a standard one, is evaluated to a causal term that captures the causes explaining why it holds. In the same way, a causal literal holds when there is some cause that justifies it. For instance, going on with our former example in which Suzy did nothing and, thus, atom bomb_5 was caused by G_1 in Figure 27. The unique causal stable model I of the program satisfies that $I \models \mathbf{hascaused}_{\mathcal{A}}(\mathit{billy},\mathit{bomb})$. It is interesting to note that $I(\mathbf{hascaused}_{\mathcal{A}}(\mathit{billy},\mathit{bomb})) = G_1$, and hence $I(\mathit{prison}(\mathit{billy})) = G_1 \cdot p$. In words, Billy is in *prison* because it has performed the actions that have lead to the bomb explosion and besides the law p has been applied to him. In general, we may state that a causal literal is satisfied when there is some sufficient cause that is an answer to the query ψ.

Proposition 6.1. *An interpretation I satisfies a causal literal of the form $(\psi :: A)$, in symbols $I \models (\psi :: A)$, if and only if there is some sufficient cause G of A w.r.t. I (Definition 4.4) that is an answer to ψ, that is $\psi(G) = 1$.* □

Proof. The proof can be found in Appendix A on page 270. □

Although this result shows that the evaluation of causal literals matches the above intuition, it does not give us much information about the precise form of the obtained causes for causal literals. To cover this aspect, we extend the notion of *proof* (Definition 3.3) and its correspondence to sufficient causes (Theorem 4.2) to programs with causal literals.

Definition 6.4 (Causal program)**.** *Given a signature $\langle At, Lb, \Psi \rangle$ where At, Lb and Ψ respectively represent sets of atoms, labels and functions $\psi : \mathbf{C}_{Lb} \longrightarrow \{0,1\}$, a* (causal) rule *$R$ is an expression of the form:*

$$l_R : A \leftarrow B_1, \ldots, B_m, \text{ not } C_1, \ldots, \text{ not } C_n \tag{92}$$

where l_R is either a label, $l_R \in Lb$, or the special symbol $l_R = 1$, and where A is a standard atom and all B_i's and C_i's are causal literals or terms. A (causal) program *P is a set of rules.* □

We assume that the signature of every program contains a query function $\psi^1 \in \Psi$ mapping every causal graph to 1. It is important to notice that any standard atom A satisfies $I(\psi^1 :: A) = I(A)$. Hence, we just use standard atoms of the form A as a shorthand for causal literals whose query function is ψ^1. With this observation in mind it is easy to see that a labelled program (Definition 3.1) is just a particular case of a causal program where $\Psi = \{\psi^1\}$. We may also extend the notion of *proof* in the following way:

Definition 6.5 (Proof (causal program))**.** *Given a positive program P and a stable model (resp. answer set) I, a* proof *$\pi(A)$ of a standard atom (resp. literal) A is recursively defined as a derivation:*

$$\pi(A) \ \stackrel{\text{def}}{=} \ \frac{\pi(\psi_1 :: B_1) \ \ldots \ \pi(\psi_m :: B_m)}{A} \ (R), \tag{93}$$

where $R \in P$ is a rule with head A, $body^+(R) = \{(\psi_1 :: B_1), \ldots, (\psi_1 :: B_m)\}$ and $I \models body^-(R)$ and each $\pi(\psi_i :: B_i)$ is, in its turn, a proof $\pi(B_i)$ of B_i such that $\psi_i(graph(\pi(B_i))) = 1$. When the positive body is empty, that is $m = 0$, the derivation antecedent $\pi(\psi_1 :: B_1) \ \ldots \ \pi(\psi_m :: B_m)$ is replaced by $\top$. □

Theorem 6.1 (Proof-cause correspondence (causal program)). *Given any completely labelled program P with a unique atom labelling δ and a stable model (resp. answer set) I of P, a proof $\pi = \pi(A)$ is a non-redundant proof of A iff $cgraph(\pi)$ is a cause of A w.r.t. I.* □

Proof. The proof can be found in Appendix A on page 252. □

We recall that in Chapters 4 we had not provided the proofs of Theorems 4.2 and 4.6. By the relation $I(\psi^1 :: A) = I(A)$, these theorems are just particular cases of Theorem 6.1.

6.2 ENCODING IN GENERAL ANNOTATED LOGIC PROGRAMS

In this section, we will show that positive causal programs may be encoded into the *General Annotated Logic Program* (GAP) framework [Kifer and Subrahmanian, 1992]. GAP provides a general semantics for multivalued logic programs where the set of truth values $\mathbf{V}$ is assumed to be a bounded upper (but possibly incomplete) semi-lattice. In our case, we have in fact a complete lattice and, thus, bounded with top element 1 and bottom element 0. The main definitions for GAPs were explained in the Background (Section 2.3). The encoding of a causal program into a GAP relies on a program where every positive rule R of the form:

$$l_R : A \leftarrow B_1, \ldots, B_m \tag{94}$$

is encoded as a GAP rule:

$$A : f_R(\mu_1, \ldots, \mu_m) \leftarrow (B_1 : \mu_1) \ \& \ \ldots \ \& \ (B_m : \mu_m) \tag{95}$$

where each body literal is v-annotated and the head is annotated with a complex annotation term that captures the meaning of the rule. In the following, we define the function f_R, starting by defining a function f_ψ for each test function $\psi \in \Psi$.

Definition 6.6 (Literal annotation function). *For every query function $\psi \in \Psi$, its* annotation function $f_\psi : \mathbf{V}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ *is given by:*

$$f_\psi(u) \ \overset{\text{def}}{=} \ \sum \{ \, G \in \mathbf{C}_{Lb} \mid G \leq u \text{ and } \psi(G) = 1 \, \}$$

Proposition 6.2. *For every causal literal of the form $(\psi :: A)$, its annotation function $f_\psi : \mathbf{V}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ is monotonic and continuous and, for every standard atom A and interpretation I, it holds that $I(\psi :: A) = f_\psi(I(A))$.* □

Proof. The proof can be found in Appendix A on page 270. □

Once we have defined an annotated function for each test function $\psi \in \Psi$ we may define an annotation function for each rule in a program.

Definition 6.7 (Rule annotation function). *For every positive rule R of the form* (94), *its* annotation function $f_R : \mathbf{V}_{Lb}^m \longrightarrow \mathbf{V}_{Lb}$ *is given by:*

$$f_R(u_1, \ldots, u_m) \overset{\text{def}}{=} \big(f_1(u_1) * \ldots * f_m(u_m)\big) \cdot l_R \cdot \delta(A)$$

where each $f_i : \mathbf{V}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ is the annotation function corresponding to the query ψ_i of the causal literal $(\psi_i :: B_i)$, l_R is the label of the rule R and $\delta(A)$ is the label assigned to the head A of the rule. □

Proposition 6.3. *For every positive rule R of the form* (94), *its annotation function $f_R : \mathbf{V}_{Lb}^m \longrightarrow \mathbf{V}_{Lb}$ is monotonic and continuous.* □

Proof. The proof relies on the results of Proposition 6.2 plus the fact that product and applications are also continuous (Proposition 3.11). □

Since any interpretation I and any causal literal of the form $(\psi :: A)$ satisfies that $I(\psi :: A) = f_\psi(I(A))$ we can easily verify the following relation between a rule annotation function and the fact that an interpretation satisfies that rule.

Proposition 6.4. *An interpretation I satisfies a positive rule R* (94), *in symbols $I \models R$, if and only if:*

$$f_R\big(I(B_1), \ldots, I(B_m)\big) \leq I(A)$$

where $f_R : \mathbf{V}_{Lb}^m \longrightarrow \mathbf{V}_{Lb}$ is the annotation function of R. □

Proof. The proof can be found in Appendix A on page 271. □

Definition 6.8 (GAP encoding). *For any positive rule R of the form* (94), *we denote by $GAP(R)$ a rule of the form* (95) *where each μ_i is an annotation variable and f_R is the annotation function of rule R. Given a positive program P, we denote by $GAP(P)$ a set containing a rule $GAP(R)$ for each rule R in P.* □

Proposition 6.5. *An interpretation I satisfies a positive rule R of the form (94), in symbols $I \models R$, iff I satisfies of $GAP(R)$ according to the GAP restricted semantics, in symbols $I \models_r R$.* □

Proof. The proof can be found in Appendix A on page 271. □

An immediate corollary of the above result is that:

Corollary 6.1. *An interpretation I is a model of a positive program P, in symbols $I \models P$, iff I is a model of $GAP(P)$ according to the GAP restricted semantics, in symbols $I \models_r GAP(P)$.* □

It is interesting to notice that the definition of the direct consequences operator T_P (Definition 4.3) may be now rewritten relying on the annotation function of a rule in the following way:

$$T_P(I)(A) \quad = \quad \sum \{ \, f_R\big(I(B_1), \ldots, I(B_m)\big) \mid R \in P \text{ and } head(R) = A \, \}$$

Since the function f_R is continuous, this definition matches the definition of the restricted direct consequences operator R_P (Definition 2.16).

Proposition 6.6. *For any interpretation I, $T_P(I) = R_P(I)$ where R_P is the restricted direct consequences operator (Definition 2.16).* □

Proof. The proof can be found in Appendix A on page 272. □

Hence, from the results of Theorems 2.3 and 2.4 plus Corollary 6.1, it immediately follows that:

Corollary 6.2. *The immediate consequences operator T_P is monotonic, continuous and $T_P^\omega = \mathtt{lfp}(T_P)$ is the least model of P.* □

We can use the result of Corollary 6.2 to prove the first half of Theorem 4.1, but it remains to prove that the T_P operator ends in a finite number of steps when P is finite.

Definition 6.9 (Causal graph height)**.** *Given a set of labels Lb, some literal labelling $\delta : Lit \longrightarrow Lb \cup \{1\}$ and a causal graph G, we denote by $height(G)$ the length of the longest simple (no repeated vertices) path in G formed by rule labels (that is, not in the image of δ).* □

Proposition 6.7. *Let P be a positive causal program, A be an atom, $k \in \{0, \ldots, \omega\}$ be an ordinal and G be a causal graph. Let λ be the number of unlabelled rules in P. If $G \leq_{\max} T_P^k(A)$ and $h = height(G) + \lambda \leq k$ then $G \leq T_P^h(A)$.* □

Proof. The proof can be found in Appendix A on page 273. □

Theorem 6.2 (Direct consequences (causal literals))**.** *Let P be a (possibly infinite) positive causal program. Then,*

i) $\texttt{lfp}(T_P)$ *is the least model of P, and*

ii) $\texttt{lfp}(T_P) = T_P^\omega(\mathbf{0})$.

If furthermore P is finite and has n rules, then

iii) $\texttt{lfp}(T_P) = T_P^n(\mathbf{0})$. □

Proof. By Corollary 6.2 it follows that T_P^ω is the least fixpoint of the T_P operator and the least model of the positive program P.

Furthermore, as we have seen in Section 3.4, for every literal A, we can write the value of $T_P^\omega(A)$ as $\sum_{G \in U} G$ for some set of $\leq$-maximal causal graphs U, that is, each $G \in U$ is a sufficient cause of A.

By Proposition 6.7, for every literal A and cause $G \leq_{\max} T_P^\omega(A)$ such that $h = height(G) + \lambda$ with λ the number of unlabelled rules in P it holds that $G \leq T_P^h(A)$. Note furthermore that $height(G)$ is the length of the longest single path in G, that is no repeated labels, so that $h = height(G) + \lambda \leq n$ with n the number of rules in the program. Hence $T_P^n(A) = T_P^\omega(A)$ for every atom A, and therefore $T_P^n = T_P^\omega$. □

As happened with above results, Theorem 4.1 is a particular case of Theorem 6.2 in the case that all causal literals are also standard atoms.

6.3 NON-MONOTONIC CAUSAL LITERALS FOR ACTION DOMAINS

In Chapter 5 we have used a predicate $noninertial(F)_s$ to capture that fluent F does not follow its inertial behaviour in situation s. There, predicate $noninertial(F)_s$ was syntactically defined by adding a rule of the form:

$$noninertial(F)_{s+1} \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$$

for each causal law (71). A more elegant way of defining the semantics of the predicate $noninertial(F)_s$ is by relying on causal literals. Under the *causal behaviour*, a fluent is an exception to the inertia when it is caused to have some value in the current situation (even if this value is the same that held in the previous situation). Hence, we may define a semantics for $\mathbf{noninertial}(F)_s$ as follows:

Definition 6.10 (NonInertial valuation)**.** *For any interpretation $I : At \longrightarrow \mathbf{V}_{Lb}$, the valuation of a causal literal of the form $\mathbf{noninertial}(F)_s$ is given by:*

$$I(\mathbf{noninertial}(F)_s) \stackrel{\text{def}}{=} \sum \{ \, G \in \mathbf{C}_{Lb} \mid G \leq_{\max} I(F_s) \text{ and } \psi_s(G) = 1 \, \}$$

where $\psi_s(G) = 1$ iff there is some label l_s in G whose timestamp situation is s, and where $G \leq_{\max} I(F_s)$ means that $G \leq I(F_s)$ and there not exists any G' such that $G < G' \leq I(F_s)$. □

Going back to the two shooters version of the Yale Shooting scenario, we have seen that $I(dead_5) \geq G_2$ where G_2 is the causal graph depicted in Figure 23. Furthermore, it is easy to see that G_2 contains the label d_5. Hence $I(\mathbf{noninertial}(dead)_5) \neq 0$, and consequently the inertia rule is not in the reduct. Thus, $I(dead_5) = G_2$, in the same way as we have seen with the syntactic definition.

Another point we have left open in Chapter 5 was analysing the *symmetric behaviour* when we have indirect effects. Recall that we had our suitcase (Example 1.4) which is open by lifting both locks and then turning the key. If *open* follows the symmetric behaviour, then (89) would be translated as:

$$k_{s+1} : \quad open_{s+1} \leftarrow key_s$$

plus the following inertia axiom:

$$open_{s+1} \leftarrow open_s, not\ \overline{open}_{s+1}$$

Now $\overline{open}_5$ does not hold, and therefore, from the inertia axiom, it follows:

$$I(open_5) \geq G_1$$

Furthermore, from the causal law k_{s+1} it also follows that:

$$I(open_5) \geq key_4 \cdot k_5$$

But we must also include more explanations. This is because $I(up(L)_5)) = I(up(L)_4))$ with $L \in \{a,b\}$ and, from causal law o_{s+1}, it follows that:

$$I(open_5) \geq \big(lift(a)_1 \cdot u(a)_2 * lift(b)_3 \cdot u(b)_4\big) \cdot o_5 \tag{96}$$

As a matter of fact, (96) is equal to G_1 but replacing the application of o_4 by o_5. Let G_1^s be a cause like G_1 but replacing the application of o_4 by o_s. Using this notation, $I(open_5) = key_4 \cdot k_4 + G_1 + G_1^5$. Moreover, if we do not perform any action after situation 5, then $I(open_s) = key_4 \cdot k_5 + G_1 + G_1^5 + \ldots + G_1^s$ for all $s \geq 5$. That is, for each situation that has passed, we accumulate a new cause corresponding to the application of rule o_s without the intervention of any action. This kind of explanation are against the intuition that a cause should point out some "new relevant event has happened." However, with the current representation the causal rule is fired every time, even though its condition is only fulfilled by inertia. A possible way of avoiding this behaviour is by translating causal law (87) as:

$$o_s : \quad open_s \leftarrow \mathbf{now}(up(a), up(b))_s$$

where $\mathbf{now}(\alpha)_s$ holds if and only if formula α is "fired" by some action in the current situation s. We may define the semantics of $\mathbf{now}(F)_s$ as follows:

Definition 6.11 (Now valuation). *For any interpretation $I : At \longrightarrow \mathbf{V}_{Lb}$, the valuation of a causal literal of the form $\mathbf{now}(\alpha)_s$ is given by:*

$$I(\mathbf{now}(\alpha)_s) \overset{\text{def}}{=} \sum \{\, G \in \mathbf{C}_{Lb} \mid G \leq_{\max} I(\alpha_s) \text{ and } \psi_s(G) = 1 \,\}$$

where $\psi_s(G) = 1$ iff there is some label l_s in G corresponding to an action and whose timestamp is $s - 1$. □

Note that, $I(\mathbf{now}(up(a), up(b))_4) = lift(a)_1 \cdot u(a)_2 * lift(b)_3 \cdot u(b)_4$ because of the action $lift(b)_3$. Hence, $I(open_4) = G_1$ where G_1 is the causal graph depicted in Figure 24. On the other hand, $I(\mathbf{now}(up(a), up(b))_5) = 0$ because $I(up(a)_5, up(b)_5) = I(up(a)_4, up(b)_4)$ and no action has the timestamp 4. Consequently, $I(open_5) = G_1 + key_4 \cdot k_5$ which is the desired result.

Unfortunately, both causal literals discussed in this section ($\mathbf{now}(\alpha)_s$ and $\mathbf{noninertial}(F)_s$) are non-monotonic and, thus, they do not fit in our current framework. Extending the definition of causal literals (Definition 6.1) to cover non-monotonic query functions is left for future work.

7 QUERIES AND COMPLEXITY ASSESSMENT

In this chapter, we consider how difficult is to compute the answer of some queries involving causal information. In LP, the most usual query is entailment, that is, deciding whether a given atom is satisfied by all models of a given program. For instance, given a program consisting of the following rules:

$$\begin{array}{llll} r_1: & a \leftarrow b & r_3: & b \leftarrow not\ c \\ r_2: & a \leftarrow c & r_4: & c \leftarrow not\ b \end{array}$$

and whose two causal stable models correspond to:

$$\begin{array}{lll} I_1(a) = r_3 \cdot r_1 & & I_2(a) = r_4 \cdot r_2 \\ I_1(b) = r_3 & & I_2(b) = 0 \\ I_1(c) = 0 & & I_2(c) = r_4 \end{array}$$

a is entailed, while b and c are not. In our case, entailment needs to be considered not only for standard, but also for causal literals. For instance, Program 6.1 entails the causal literal $\mathbf{hascaused}(billy, bomb_5)$. Besides computing entailment, it is also worth to ask whether a causal graph G is a sufficient cause of an atom (Definition 4.4), or to find the essential conditions without which a literal would not hold, that is, its necessary causes.

Definition 7.1 (Necessary cause)**.** *Given an interpretation I and an atom A, we say that a causal graph G is a* necessary cause *of A iff G is a subgraph of all sufficient causes of A and $I(A) \neq 0$.* □

For instance, causal graph G_1 in Figure 26 is a sufficient and necessary cause of $bomb_5$ with respect to Program 6.1. If we add the rules:

$$\begin{array}{lll} k_{s+1} & : & open_{s+1} \leftarrow key_s \\ \$ & : & key_3 \end{array}$$

to Program 6.1, G_1 is still a sufficient cause, but not a necessary one, because $key_3 \cdot k_4 \cdot b_4$ is also a sufficient cause. Now, the only necessary cause amounts to rule b_4 because this rule is required for both sufficient causes.

The most obvious way of answering these questions is by computing a model and then used it for deciding the answer. Unfortunately, causal values are sets of maximal causal graphs or, alternatively, causal terms in minimal disjunctive normal form and so, their direct representation may easily require an exponential amount of space. Fortunately, the obtained causal terms contain a great degree of redundancy that can be captured by a more compact (polynomial size), although less readable, representation. This representation comes with the undesired side effect that deciding whether a causal literal holds or not, and deciding whether a causal graph is a sufficient cause of a literal are not trivial task any more. Therefore, we also study the complexity of deciding these questions. Figure 28 shows completeness results for deciding different types of queries in programs that are only contain standard literals and causal literals of the form "$\mathbf{hascaused}(agent, A)$."

	positive	well founded	answer set	
			(brave)	(cautions)
entailment	P	P	NP	coNP
sufficient	P	P	NP	coNP
sufficient*	coNP	coNP	Σ_2^P	coNP
necessary	coNP	coNP	Σ_2^P	coNP

Figure 28: Completeness results for deciding different types of causation in causal logic programs. *The third row refers to deciding sufficient cause w.r.t. programs which are not head labelled.

Entailment and sufficient-cause queries under the different causal semantics are as hard as entailment under the their standard counterparts, although deciding whether a causal graph is a sufficient cause is harder if the program is not head labelled or does not have a unique atom labelling δ. Although, in this case, complexity for deciding sufficient causes rises for positive programs and normal programs under the well-founded semantics and brave reasoning under the stable model semantics, cautious reasoning under the stable model semantics is not affected by this assumption. This is because cautious reasoning ("for all models ...") and necessary causation ("for all sufficient causes..."), are universal properties and, while these two sources of complexity are not independent, a witnessing (polynomially checkable) counter-example to their conjunction can be found. Assuming that programs are head labelled programs is reasonable since, as far as this dissertation covers, all practical examples representing causal scenarios fall into this category.

The second assumption is that programs only contain standard literals and causal literals of the form "$\mathbf{hascaused}(agent, A)$." In general, programs may contain causal literals of the form $(\psi :: A)$ where $\psi : \mathbf{C}_{Lb} \longrightarrow \{0,1\}$ is an arbitrary function. The computational cost of all types of queries will depend on the complexity of evaluating those query functions ψ for causal literals. As we will see, deciding whether a causal literal of the form "$\mathbf{hascaused}(agent, A)$" holds is feasible in polynomial time. This fact allows us to obtain the results in Figure 28, without relying on external oracles for computing each function ψ. All these results can be extended to any program in which the valuation of all causal literals can be computed in polynomial time.

In the rest of the chapter we go through these results in detail. Section 7.1 presents a polynomial size representation of causal values and shows that this representation is useful to compute the least model of a positive program in polynomial time. Section 7.2 places the different queries into complexity classes relying on an external oracle O_Ψ to compute the test functions. Then, Section 7.3 uses the results of the previous section to specify complete characterizations for the case that all the test functions can be computed in polynomial time. Finally, Section 7.4 shows that, in fact, causal literals of the form "$\mathbf{hascaused}(agent, A)$" can be computed in polynomial time, obtaining the above complexity results in Figure 28 as a corollary.

7.1 REPRESENTING CAUSAL VALUES

As said above, comparing two arbitrary causal terms t and t' is not an easy task (in fact it is coNP-hard). A naive approach for making such comparison would be rewriting the causal terms t and t' in a minimal complete disjunctive normal form (Definition 3.11). In this normal form, each causal term t is of the form $\sum t_i$ where each t_i, in its turn, represents a causal graph. Then, the comparison is more or less straightforward, $\sum t_i \leq \sum t'_j$ iff for each term t_i there is some $t'_j \geq t_i$. Since t_i and t'_j represent causal graphs G_i, and G'_j and $t'_j \geq t_i$ iff G'_j is a subgraph of G_i, this task can be computed in polynomial time. However, applying distributivity may easily blow up complexity. In order to illustrate this fact, consider the following positive program:

Program 7.1.

$$\begin{array}{lll} m_i: & p_i \leftarrow p_{i-1},\, q_{i-1} & \text{for } i \in \{\, 2,\ldots,n \,\} \\ n_i\,: & q_i \leftarrow p_{i-1},\, q_{i-1} & \text{for } i \in \{\, 2,\ldots,n \,\} \end{array}$$

$$\begin{array}{llll} a: & p_1 & c: & q_1 \\ b: & p_1 & d: & q_1 \end{array}$$

It is easy to see that the least causal model I of this program satisfies that $I(p_1) = a + b$ and $I(q_1) = c + d$. Then, the interpretation for p_2 must satisfy:

$$\begin{aligned} I(p_2) &= (I(p_1) * I(q_1)) \cdot m_2 \\ &= ((a+b) * (c+d)) \cdot m_2 \\ &= (a*c) \cdot m_2 \ + \ (a*d) \cdot m_2 \ + \ (b*c) \cdot m_2 \ + \ (b*d) \cdot m_2 \end{aligned}$$

This addition cannot be further simplified. The four addends above correspond to the four sufficient causes of p_2. Analogously, $I(q_2)$ can be also expressed as a sum of four sufficient causes: we just replace m_2 by n_2 in the above expression for $I(p_2)$. But then, $I(p_3)$ corresponds to $(I(p_2) * I(q_2)) \cdot m_3$ and, applying distributivity, this yields a sum of 4×4 sufficient causes. In the general case, each atom p_n or q_n has $2^{2^{n-1}}$ sufficient causes. That is, expanding the complete causal value into this disjunctive normal form becomes intractable.

This program also reveals another issue. Even if distributivity is not applied, the causal terms directly obtained by the direct consequences operator T_P for p_2 and q_2 require 4 operators, the causal terms for p_3 and q_3 require 10, $I(p_3) = ((a+b)*(c+d)) \cdot m_2 * ((a+b)*(c+d)) \cdot n_2) \cdot m_3$ and, in general, the terms for p_n or q_n would require $2^n + 2^{n-1} - 2$ operators. However, an interesting observation is that subterm $(a+b)*(c+d)$ occurs twice in $I(p_3)$, and the same happens for $I(q_3)$. This subterm will occur four times in the causal terms for atoms p_4 and q_4. Avoiding repetitions will allow computing the least model of the program in polynomial time (and thus, using a polynomial number of operators to represent it).

Definition 7.2 (Term and interpretation graph)**.** *Given a set of labels Lb, a* term graph *(t-graph)* $\tilde{T} = \langle V, E, f_V, f_E, v_r \rangle$ *is a rooted, connected and labelled directed graph with a set of vertices* V*, edges* E*, root* $v_r \in V$*, a vertices label function*

$$f_V : V \longrightarrow Lb \cup \{0, 1, +, *, \cdot\}$$

and a partial label function $f_E : E \longrightarrow \{left,\ right\}$ *for edges such that:*

i) all leaves are labelled with unitary values *(a label in* Lb*,* 1 *or* 0*),*

ii) all non-leaf nodes are labelled with operators ($+$*,* $*$ *or* $\cdot$*)*

iii) for any vertex labelled with the application operator ($\cdot$*) there are exactly two outgoing edges labelled '*$left$*' and '*$right$*' being the target for the latter a leaf node. The rest of edges in the graph are unlabelled, that is* f_E *is undefined.*

The set of t-graphs is denoted by $\tilde{\mathbf{V}}_{Lb}$*.* □

Each vertex in a t-graph $\tilde{T}$ represents a term given as follows:

$$
\begin{array}{ll}
term(\tilde{T},v) \overset{\text{def}}{=} f_V(v) & \text{for any leaf } v \text{ in } \tilde{T} \\
term(\tilde{T},v) \overset{\text{def}}{=} \sum\{\, term(\tilde{T},v') \mid (v,v') \in E \,\} & \text{if } f_V(v) = (+) \\
term(\tilde{T},v) \overset{\text{def}}{=} \prod\{\, term(\tilde{T},v') \mid (v,v') \in E \,\} & \text{if } f_V(v) = (*) \\
term(\tilde{T},v) \overset{\text{def}}{=} term(\tilde{T},u) \cdot term(\tilde{T},w) & \text{if } f_V(v) = (\cdot) \\
 & \quad f_E(v,u) = left \text{ and} \\
 & \quad f_E(v,w) = right
\end{array}
$$

The term represented by t-graph $\tilde{T}$ is the term represented by its root v_r. In this case we just write $\tilde{T}$ instead of $term(\tilde{T},v_r)$. As an example, Figure 29 depicts the t-graph corresponding to $I(p_3)$ in our running example.

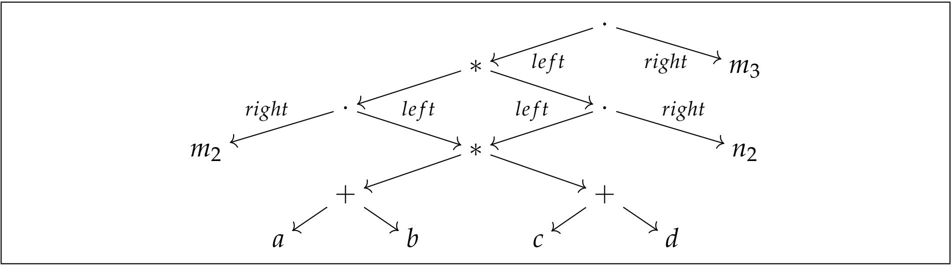

Figure 29: The t-graph associated to $I(p_3)$ in Program 7.1.

A term graph is just a graph representation of a term using pointers to avoid repeated subexpressions as can be done in any programming language or with Prolog terms. In order to avoid writing down the label functions f_V and f_E we use superindices to denote the labels of vertices and edges. That is, v^*, v^+, $v^{\odot}$ and v^l respectively denote a vertex v such that $f_V(v) = (*)$, $f_V(v) = (+)$, $f_V(v) = (\cdot)$ and $f_V(v) = l$. Furthermore an edge $(v_1,v_2)^r$ and $(v_1,v_2)^l$ respectively denote an edge (v_1,v_2) such that $f_E(v_1,v_2) = right$ and $f_E(v_1,v_2) = left$.

An edge without superindex denotes an unlabelled edge. We also define some abbreviations that work as constructors building new t-graphs. We denote by $\tilde{T}^0$ the empty t-graph and by $\tilde{T}^1$ and $\tilde{T}^l$ the t-graphs with only one vertex v respectively labelled by 1 and $l \in Lb$. Given a t-graph $\tilde{T}$ and a label l, we denote by $\tilde{T} \cdot l$ the t-graph with root $v_r^{\odot}$, set of vertices $V = V' \cup \{v_r^{\odot}, w_r^l\}$ and set of edges $E = E' \cup \{(v_r^{\odot}, v_r')^l, (v_r^{\odot}, w_r^l)^r\}$ where V', E' and v_r' are respectively the set of vertices, edges and the root of $\tilde{T}$. Note that every vertex and edge belonging to $\tilde{T}$ keeps the label that it has in $\tilde{T}$. It is easy to see that, if $\tilde{T}$ represents a causal term t, then $\tilde{T} \cdot l$ represents the causal term $t \cdot l$. Similarly, given a set $S \subseteq \tilde{\mathbf{V}}_{Lb}$ of t-graphs, we denote by $\sum S$ and $\prod S$ respectively the t-graph $\tilde{T}$ whose roots are v_r^+ and v_r^* and whose sets of vertices and edges correspond to:

$$\begin{aligned} V &\stackrel{\text{def}}{=} \bigcup \{ \, V(\tilde{T}) \mid \tilde{T}' \in S \, \} \cup \{ \, v_r \, \} \\ E &\stackrel{\text{def}}{=} \bigcup \{ \, E(\tilde{T}) \mid \tilde{T}' \in S \, \} \cup \{ \, (v_r, v_r') \mid 1 \leq i \leq m \, \} \end{aligned}$$

Note that $V(\tilde{T})$, $E(\tilde{T})$ and v_r' respectively denote the set of vertices, edges and the root of $\tilde{T} \in S$. It is easy to see that, if U is a set of terms such that $u \in U$ iff $\tilde{T} \in S$, and $\tilde{T}$ represent u, then the t-graphs $\sum S$ and $\prod S$ respectively represents the causal terms $\sum U$ and $\prod U$. We also assume the following simplifications:

1. If a vertex v is labelled with the operation $(*)$ or $(\cdot)$ and one of its children is labelled with 0, then the label of v is replaced by 0 and its children are removed.

2. If v is labelled with the operation $(+)$ and one of its children is labelled with 0, then the latter is removed. The label of a vertex v labelled with the operation $(+)$ without children is replaced by 0.

3. If a vertex v is labelled with the operation $(*)$ and one of its children is labelled with 1, then the latter is removed. The label of vertex v labelled with the operation $(*)$ without children is replaced by 1.

4. If a vertex v is labelled with the operation $(\cdot)$ and one of its children is labelled with 1 then the latter is removed and v replaced by the other children.

5. If a vertex v is labelled with the operations $(+)$ and one of its children is labelled with 1, then the label of v is replaced by 1 and all its children are removed.

6. If the root v is labelled with 0, then it is removed.

Proposition 7.1. *The causal value represented by a t-graph $\tilde{T}$ obtained after applying the above simplifications is the same of the one represented by the original t-graph. Furthermore, all these simplifications can be run in polynomial time.* □

Proof. The proof can be found in Appendix A on page 274. □

In the following we assume that every t-graph has been simplified according to the above rules. Consequently, a t-graph $\tilde{T}$ represents the causal value 0 if and only if its set of vertices is empty. Similarly, a t-graph $\tilde{T}$ represents the causal value 1 if and only if it has a unique vertex whose label is 1. We extend the notion of t-graph to interpretations in the following straightforward way.

Definition 7.3 (t-Interpretation)**.** *A* t-interpretation $\tilde{I} : At \longrightarrow \tilde{\mathbf{V}}_{Lb}$ *is a mapping from atoms into t-graphs. We say that a t-interpretation $\tilde{I}$ represents an interpretation I iff it holds that $\tilde{I}(A)$ represents $I(A)$ for every atom A. We denote by $\tilde{\mathbf{I}}_\sigma$ the set of t-interpretations over a signature σ.* □

For an interpretation I, there may be several different t-interpretations representing it. We can now use t-interpretations in order to define a new direct consequences operator.

Definition 7.4 (Test function)**.** *Given a test function $\psi \in \Psi$, by $\tilde{\psi} : \tilde{\mathbf{V}}_{Lb} \longrightarrow \tilde{\mathbf{V}}_{Lb}$ we denote a function such that $G \leq \tilde{\psi}(\tilde{T})$ iff $G \leq \tilde{T}$ and $\psi(G) = 1$ for all causal graph G and t-graph $\tilde{T}$. Furthermore, given a positive rule R of the form:*

$$l_R : A \leftarrow (\psi_1 :: B_1), \ldots, (\psi_m :: B_m)$$

we denote by $\tilde{f}_R : \tilde{\mathbf{V}}_{Lb} \times \ldots \times \tilde{\mathbf{V}}_{Lb} \longrightarrow \tilde{\mathbf{V}}_{Lb}$ a function such that:

$$\tilde{f}_R(\mu_1, \ldots, \mu_m) \overset{\text{def}}{=} \big(\tilde{\psi}_1(\mu_1) * \ldots * \tilde{\psi}_m(\mu_m)\big) \cdot l_R \cdot \delta(A)$$

□

Function $\tilde{f}_R$ is similar to the annotation function f_R defined in Chapter 6, but replacing each test function ψ_i by its corresponding $\tilde{\psi}_i$. From this definition and the evaluation of causal literals (Definition 6.3), the following result is immediate.

Proposition 7.2. *Given an interpretation I, a t-interpretation $\tilde{I}$ representing I and a test function ψ, it holds that $\tilde{\psi}(I(A))$ represents $I(\psi :: A)$ for every atom A.* □

We can now combine t-interpretations with an oracle which externally computes the functions $\tilde{\psi}$ for each test function $\psi \in \Psi$ in order to define a procedure to compute the direct consequences operator.

Definition 7.5 (t-Direct consequences operator)**.** *Given a positive logic program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, the* t-direct consequences operator *is a function $\tilde{T}_P : \tilde{\mathbf{I}}_\sigma \longrightarrow \tilde{\mathbf{I}}_\sigma$ given by:*

$$\tilde{T}_P(\tilde{I})(A) \overset{\text{def}}{=} \sum \{ \tilde{f}_R(\tilde{I}(B_1), \ldots, \tilde{I}(B_m)) \mid R \in P \text{ and } head(R) = A \}$$

for any t-interpretation $\tilde{I}$ and any atom $A \in At$. □

It is easy to see that this definition just translates the direct consequences operator (Definition 4.3) into the language of t-graphs.

For clarity sake, given a set S of decision functions, we denote by O_S an oracle that can decide all the decision functions $f \in S$. For instance, given the set of query functions Ψ, by O_Ψ we denote an oracle that can decide all query functions $\psi \in \Psi$, and by $O_{\Psi \cup \text{NP}}$ we denote an oracle that can decide all query functions $\psi \in \Psi$ plus all NP queries.

Proposition 7.3. *Given a program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, an interpretation I, any t-interpretation $\tilde{I}$ representing I holds that $\tilde{T}_P(\tilde{I})$ represents $T_P(I)$. Furthermore $\tilde{T}_P(\tilde{I})$ can be computed in polynomial time with an oracle O_Ψ.* □

Proof. The proof can be found in Appendix A on page 275. □

Theorem 6.2 asserted that computing the least model of a finite program by the direct consequences operator does not take more steps than the number of rules in the program. Combining this result with Proposition 7.3 implies that a t-interpretation representing the least model of a program can be computed in polynomial time by iterating the $\tilde{T}_P$ procedure from the bottom t-interpretation, denoted by $\tilde{\mathbf{0}}$ which maps each atom to the empty t-graph.

Corollary 7.1. *Given a positive program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, we can compute a t-interpretation $\tilde{I}$ representing the least model I of P in polynomial time with an oracle O_Ψ.* □

Furthermore, from Theorem 4.9, the causal well-founded model can be obtained from the standard well-founded model by a single application of the Γ_P operator. Hence, given the fact that the standard well-founded model can be computed in polynomial time, immediately follows the next result.

Corollary 7.2. *Given a normal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, we can compute a pair of t-interpretations $\langle \tilde{I}, \tilde{J} \rangle$ representing the causal well-founded model $\langle \mathtt{lfp}(\Gamma_P^2), \mathtt{gfp}(\Gamma_P^2) \rangle$ of P in polynomial time with an oracle O_Ψ.* □

7.2 MEMBERSHIP RESULTS

In this section, we place the entailment, and sufficient and necessary causal queries into complexity classes relying on an external oracle O_Ψ to compute the test functions and the result of Corollary 7.2.

7.2.1 Entailment

For showing that entailment of causal literals under the well-founded semantics is feasible in polynomial time with an oracle O_Ψ just notice that, by definition, $P \models_{wf} (\psi :: A)$ iff $I(\psi :: A) \neq 0$. Similarly, $P \not\models_{wf} (\psi :: A)$ iff $J(\psi :: A) = 0$ where $J = \mathtt{gfp}(\Gamma_P^2)$. Applying the above simplifications a t-graph $\tilde{T}$ represents the causal value 0 iff its set of vertices is empty. Then, $P \models_{wf} (\psi :: A)$ (resp. $P \models_{wf}$ *not* $(\psi :: A)$) holds iff a t-interpretation $\tilde{I}$ representing the least (resp. greatest) fixpoint $I = \mathtt{gfp}(\Gamma_P^2)$ (resp. $I = \mathtt{lfp}(\Gamma_P^2)$) of Γ_P^2 evaluates $\tilde{\psi}(I(A))$ to the empty t-graph. As we have seen, from Corollary 7.2, this is computable in polynomial time.

Theorem 7.1 (Entailment well-founded membership). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ and a causal literal $(\psi :: A)$, deciding whether $P \models_{wf} (\psi :: A)$ and $P \models_{wf}$* not *$(\psi :: A)$ is feasible in polynomial time with an oracle O_Ψ.* □

Since the well-founded and the least model semantics agree for positive programs, it is clear that deciding whether a causal literal is a consequence of a positive program under the least model semantics is also feasible in polynomial time with an oracle O_Ψ. For normal programs under the stable model semantics we distinguish, as usual, between brave and cautious reasoning.

Definition 7.6 (Brave/cautious entailment). *A causal literal $(\psi :: A)$ is a* brave *(resp.* cautious*) consequence of a given program P iff $(\psi :: A)$ holds with respect to some (resp. every) stable model I of P.* □

The following result shows that deciding entailment for both, standard and causal literals, with respect to causal programs is in NP^{O_Ψ} and coNP^{O_Ψ} for brave and cautious reasoning respectively.

Theorem 7.2 (Entailment stable model membership). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ and a causal literal $(\psi :: A)$, deciding whether $(\psi :: A)$ is a brave (resp. cautious) consequence of P is in* NP^{O_Ψ} *(resp.* coNP^{O_Ψ}*).* □

Proof. Consider the following procedure.

1. Obtain a program Q by replacing each literal of the form $(not\ \psi_i :: B_i)$ by $not\ aux\_psi_i\_B_i$ where $aux\_psi_i\_B_i$ is a new auxiliary atom. Let Aux be the set of all new auxiliary atoms introduced in Q. Then every negative literal in Q is a standard literal.
2. Guess a set of atoms $S \subseteq Lit \cup Aux$
3. Compute the reduct Q^S of Q w.r.t. the set of atoms S
4. Compute the least model I of Q^S
5. Fail if I is not a causal stable model of I, that is
   a) Fail if ($A \in S$ iff $I(A) \neq 0$) does not hold for every atom $A \in Lit$
   b) Fail if ($aux\_psi_i\_B_i \in S$ iff $I(\psi_i :: B_i) \neq 0$) does not hold for every auxiliary atom $aux\_psi_i\_B_i \in Aux$
6. Success if $I(\psi :: A) \neq 0$. Fail otherwise.

This procedure succeeded iff there is a causal stable model I of P such that $I(\psi_i :: B_i) \neq 0$, that is, iff $(\psi_i :: B_i)$ is a brave consequence of P. Replacing $\neq$ by $=$ in step 6, the new procedure success iff there is a causal stable model I of P such that $I(\psi :: A) = 0$, that is, iff $(\psi :: A)$ is not a cautious consequence of P. Hence deciding whether $(\psi :: A)$ is a brave (resp. cautious) consequence of P is in NP^{O_Ψ} (resp. coNP^{O_Ψ}).

Note that, steps 1-3 are clearly feasible in polynomial time. Step 4 is feasible in polynomial time with oracle O_Ψ (Corollary 7.1). Similarly, $I(\psi :: A) \neq 0$ iff $\tilde{\psi}(I(A)) \neq 0$ (Proposition 7.2) and this is feasible in polynomial time with oracle O_Ψ.

In order to show the soundness of the above procedure, let J be a causal interpretation over a sets of atoms $At \cup Aux$ such that $I(A) = J(A)$ for every atom $A \in At$ and $J(aux\_psi_i\_B_i) = I(\psi_i :: B_i)$. Let $S \subseteq At \cup Aux$ be the set of atoms that hold w.r.t. J. Then, $P^I = Q^J = Q^S$. Hence, I is a causal stable model of P iff I is the least model of Q^S. Step 5 checks and fails if the least model of Q^S did not coincide with guessed S, that is, iff I is not a causal stable model of P. Finally, step 6 check whether $(\psi :: A)$ holds or not with respect to I. □

Corollary 7.3. *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$, deciding whether a causal literal $(\psi :: A)$ is a brave (resp. cautious) consequence of P is in* NP *(resp.* coNP*). It is, furthermore, feasible in polynomial time w.r.t. the well-founded semantics or when a program is positive.* □

7.2.2 Sufficient Cause

Testing whether a causal graph G is a sufficient cause of an atom A will rely on the following result whose proof will be addressed in Section 7.4.

Proposition 7.4. *Given a pair of t-graphs, $\tilde{T}$ and $\tilde{T}'$, and a causal graph G, deciding whether $G \leq \tilde{T}$ is feasible in polynomial time. Deciding whether $G \leq_{\max} \tilde{T}$, $\tilde{T} \leq G$ and $\tilde{T} \leq \tilde{T}$ are in coNP.* □

Furthermore, in the particular case that causal graphs are acyclic, we can fix the following better bound. Recall that, from Theorem 4.8, the sufficient causes of every head labelled program are acyclic.

Proposition 7.5. *Given an acyclic causal graph G and a t-graph $\tilde{T}$, deciding whether $G \leq_{\max} \tilde{T}$ is feasible in polynomial time.* □

Proof. The proof can be found in Appendix A on page 278. □

Combining these results with Corollary 7.1 we can settle an upper bound for the complexity of deciding whether a causal graph is a sufficient cause with respect to the well-founded semantics.

Theorem 7.3 (Sufficient cause well-founded membership)**.** *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, deciding whether a causal graph G is a sufficient cause of an atom A with respect to the well-founded model of P is in* coNP^{O_Ψ}*. Furthermore, if either P is head labelled and δ is a unique atom labelling, or G is acyclic, then it is decidable in polynomial time with O_Ψ.* □

Proof. From Corollary 7.2, we can compute a pair of t-interpretations $\langle \tilde{I}, \tilde{J} \rangle$ representing the well-founded model of a program P in polynomial time with the oracle O_Ψ. Furthermore, from Proposition 7.4, deciding whether a causal graph G is a sufficient cause of an atom A, that is $G \leq_{\max} \tilde{I}(A)$, is in coNP. From

Proposition 7.5, if G is acyclic, then it is decidable in polynomial time. Moreover, from Theorem 4.8, if P is head labelled and has a unique atom labelling, then every sufficient causal is acyclic. Hence, if the causal graph G contains a non-reflexive cycle, G cannot be a sufficient cause. □

Definition 7.7 (Brave/cautious sufficient cause). *A causal graph G is a* brave *(resp.* cautious*)* sufficient cause *of an atom A with respect to a given labelled program P iff G is a sufficient cause of A with respect to some (resp. every) stable model I of P.* □

For instance, given a program consisting of the following rules:

$$\begin{array}{llllll} r_1: & a \leftarrow b & r_3: & b \leftarrow \mathit{not}\ c & r_5: & a \\ r_2: & a \leftarrow c & r_4: & c \leftarrow \mathit{not}\ b & & \end{array}$$

whose two causal stable models correspond to:

$$\begin{array}{rcl rcl} I_1(a) & = & r_3 \cdot r_1 + r_5 & I_2(a) & = & r_4 \cdot r_2 + r_5 \\ I_1(b) & = & r_3 & I_2(b) & = & 0 \\ I_1(c) & = & 0 & I_2(c) & = & r_4 \end{array}$$

$r_3 \cdot r_1$, $r_4 \cdot r_2$ and r_5 are brave sufficient causes of a, while only the later is a cautious one.

Theorem 7.4 (Cautious sufficient cause membership). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, a causal graph G and an atom A, deciding whether G is a cautious sufficient cause of an atom A is in* coNP$^{O_\Psi}$. □

Proof. In order to decide whether G is a cautious sufficient cause we can be do as follows.

1. Obtain program Q as in step 1 of Theorem 7.2.
2. Guess a set of atoms $S \subseteq Lit \cup Aux$ and a causal graph G' s.t. $G < G'$
3. Compute the reduct Q^S of Q w.r.t. the set of atoms S
4. Compute the least model I of Q^S
5. Fail if I is not a causal stable model of P as in step 5 of Theorem 7.2.
6. Success if $G \not\leq_{\max} I(A)$. Fail otherwise. That is
   a) Success if $G \not\leq I(A)$
   b) Success if $G' \leq I(A)$. Fail otherwise.

This procedure succeeds iff there is some causal stable model I and causal graph G' such that $G \not\leq I(A)$ or $G' \leq I(A)$. The latter holds iff there is some causal stable model I such that G is not a sufficient cause of A.

Steps 1-3 and step 5 are feasible in polynomial time as in Theorem 7.2. Steps 4 and 6 are feasible in polynomial time with oracle O_Ψ respectively due to Corollary 7.1 and Proposition 7.4. Hence, deciding whether G is a cautious sufficient cause of A is in coNP^{O_Ψ}. □

Theorem 7.5 (Brave sufficient cause membership)**.** *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$, a causal graph G and an atom A, deciding whether G is a brave sufficient cause of A is in $\text{NP}^{O_{\Psi \cup \text{NP}}}$. If P is head labelled and δ is a unique atom labelling, or G is acyclic, then it is in NP^{O_Ψ}.* □

Proof. In order to decide whether G is a brave sufficient cause we can do as follows.

1. Obtain program Q as in step 1 of Theorem 7.2.
2. Guess a set of atoms $S \subseteq Lit \cup Aux$
3. Compute the reduct Q^S of Q w.r.t. the set of atoms S
4. Compute the least model I of Q^S
5. Fail if I is not a causal stable model of P as in step 5 of Theorem 7.2.
6. Success if $G \leq_{\max} A$ w.r.t. I. Fail otherwise.

This procedure succeeds iff there is some causal stable model I such that $G \leq_{\max} I(A)$. The latter holds iff there is some causal stable model I such that G is a sufficient cause of A.

Steps 1-3 and step 5 are feasible in polynomial time as in Theorem 7.2. Step 4 is feasible in polynomial time with oracle O_Ψ due to Corollary 7.1. Step 6 is in coNP (Proposition 7.4). Hence, deciding whether G is a brave sufficient cause of A is in coNP with oracle O. Furthermore, if P is head labelled with a unique atom labelling, or G is acyclic, then it is decidable in polynomial time (Proposition 7.5) and, thus, the oracle only needs to compute the query functions ψ. □

Corollary 7.4. *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$, and an atom A, deciding whether G is*

i) a sufficient cause of A is in coNP *when P is positive. It is feasible in polynomial time when P is head labelled and δ is a unique atom labelling, or the causal graph G is acyclic.*

ii) a sufficient cause of A w.r.t. the well-founded model is in coNP*. It is feasible in polynomial time when either P is head labelled and δ is a unique atom labelling, or G is acyclic.*

iii) a cautious sufficient cause of A is in coNP*,*

iv) a brave sufficient cause of A is in Σ_2^P. It is in coNP *when either P is head labelled and δ is a unique atom labelling, or G is acyclic.* □

7.2.3 Necessary Cause

We may follow a similar process to show membership for queries about necessary causes.

Theorem 7.6 (Cautious necessary cause membership). *Given a causal program P, deciding whether a causal graph G is a cautious necessary cause of an atom A is in* coNP^{O_Ψ}*. Deciding whether a causal graph G is a cautious necessary cause of an atom A w.r.t. the well-founded model is also in* coNP^{O_Ψ}*.* □

Proof. Consider the following procedure.

1. Obtain the program Q as in step 1 of Theorem 7.2.
2. Guess a set of atoms $S \subseteq Lit \cup Aux$ and a causal graph G'
3. Compute the reduct Q^S of Q w.r.t. the set of atoms S.
4. Compute the least model I of Q^S.
5. Fail if I is not a causal stable model of P as in step 5 of Theorem 7.2.
6. Succeeds if $I(A) = 0$.
7. Fail if $G' \not\leq I(A)$.
8. Fail if $G \subseteq G'$. Success otherwise.

This procedure success iff there is a causal stable model I and a causal graph such that either $I(A) = 0$ or $G' \leq I(A)$ and $G \not\subseteq G'$. The later holds iff there is a causal stable model I such that G is not a necessary cause of A. Steps 1-5 are feasible in polynomial time with oracle O_Ψ as in Theorem 7.2. Steps 6 and 7 are feasible in polynomial time with oracle O_Ψ due to Proposition 7.4. Checking whether a graph is a subgraph (step 8) of another graph also is also computable in polynomial time. Hence, deciding whether a causal graph G is a cautious necessary cause of an atom A is in coNP with an oracle O.

Replacing steps $3-5$ by the procedure for computing the causal well-founded model $\langle I, J\rangle$ of P, which is feasible in polynomial time (Corollary 7.2), the obtained procedure decides whether G is a sufficient cause of A w.r.t. the well-founded model. □

Theorem 7.7 (Brave necessary cause membership)**.** *Given a labelled program P, deciding whether a causal graph G is a brave necessary cause of an atom A is* $\mathrm{NP}^{O_{\Psi \cup \mathrm{NP}}}$. □

Proof. We can decide whether G is a necessary cause of A as follows:

1. Obtain program Q as in step 1 of Theorem 7.2.
2. Guess a set of atoms $S \subseteq Lit \cup Aux$.
3. Compute the reduct Q^S of Q w.r.t. the set of atoms S.
4. Compute the least model I of Q^S.
5. Fail if I is not a causal stable model of P as in step 5 of Theorem 7.2.
6. Success if G is a necessary cause w.r.t. I. Fail otherwise.

This procedure success iff G there is a causal stable model I and G is a necessary cause of A w.r.t. I. The later holds iff G is a brave necessary cause of A. Steps 1-5 are computable in polynomial time with the oracle O_Ψ as in Theorem 7.2. Step 6 can be checked in the oracle O, since it solves NP queries (Theorem 7.6). Hence, deciding whether a causal graph G is a brave necessary cause of an atom A is NP with an oracle O. □

Corollary 7.5. *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi\rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$ and an atom A deciding whether a causal graph G is*

i) *a necessary cause of A is in* coNP *when P is positive.*

ii) *a necessary cause of A w.r.t. the well-founded model is in* coNP.

iii) *a cautious necessary cause of A is in* coNP.

iv) *a brave necessary cause of A is in* Σ_2^{P}. □

7.3 COMPLETENESS RESULTS

In this Section, we use the results of the previous section to specify complete characterisations for the case that all the test function can be computed in polynomial time.

7.3.1 Entailment

Theorem 7.8 (Entailment well-founded complete)**.** *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$ and a causal literal $(\psi :: A)$, deciding whether $P \models_{wf} (\psi :: A)$ and $P \models_{wf}$* not *$(\psi :: A)$ w.r.t. to the well-founded model is* P*-complete. It is* P*-hard even if P is positive.* □

Proof. Membership directly follows from Corollary 7.3. Hardness follows from the fact that every standard program is also a causal program and deciding whether a literal is a consequence of a standard positive program is P-complete. □

Theorem 7.9 (Entailment stable model complete)**.** *Given program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that the function $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$, deciding whether a causal literal $(\psi :: A)$ is a brave (resp. cautious) consequence of P is in NP-complete (resp. coNP-complete).* □

Proof. Similarly, membership follows directly from Corollary 7.3. Hardness follows from the fact that every standard program is also a causal program and deciding whether a literal is a brave (resp. cautious) consequence of a standard positive program is NP-complete (resp. coNP-complete). □

7.3.2 Sufficient Cause

Theorem 7.10 (Sufficient cause well-founded complete)**.** *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for every $\psi \in \Psi$, deciding whether a causal graph G is a sufficient cause of an atom A w.r.t. the well-founded model is* coNP*-complete. Hardness holds even if P is a positive labelled program.* coNP*-hardness holds even for positive programs. If P is head labelled and δ is a unique atom labelling, or G is acyclic, then it is P-complete.* □

Proof. Membership follows directly from Corollary 7.4. P-hardness follows from the fact that every standard program is also a causal program and deciding whether an atom is a consequence of a positive standard logic program is P-complete. An atom A is a consequence of a standard logic program iff 1 is a sufficient cause of A. Cabalar et al. [2014b] have shown that deciding whether a causal graph G is a sufficient cause of atom A with respect to the least model I of P is coNP-complete even in the case of positive labelled programs. In the following, we show such proof by building a log-space reduction for deciding the truth of a formula $\varphi = \forall x_1, \ldots, x_m\ \varrho$, where $\varrho = \psi_1 \vee \ldots \vee \psi_r$ and each $\psi_i = L_{i1} \wedge L_{i2} \wedge L_{i3}$ is a conjunction of three literals L_{ij} over atoms $x_1, \ldots, x_m$. Given φ, we construct a labelled program P_φ as follows:

Program 7.2.

$$\begin{array}{lllll}
f & : & f & & \\
x_k & : & x''_k & \leftarrow f & \text{for each } k \in \{1,\ldots,m\} \\
t & : & x'_k & \leftarrow x''_k & \text{for each } k \in \{1,\ldots,m\} \\
t & : & t & & \\
x_k & : & \psi_i & \leftarrow t & \text{if } L_{ij} = x_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
f & : & \psi_i & \leftarrow x_k & \text{if } L_{ij} = \overline{x}_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
x_k & : & x_k & & \text{for each } k \in \{1,\ldots,m\} \\
 & & \gamma & \leftarrow x'_1, \ldots, x'_m & \\
 & & \varrho & \leftarrow \psi_1, \ldots, \psi_r, \gamma &
\end{array}$$

This transformation can be done using logarithmic space. Moreover, it can be shown that φ is true if and only if the complete connected causal graph graph G_{tf} with a set of vertices $\{x_1, \ldots, x_m, t, f\}$ is a sufficient cause of atom ϱ.

To wit, first observe that any model I of the program P_φ must satisfy that $I(x'_k) = f \cdot x_k \cdot t$ for all $k \in \{1, \ldots, m\}$ due to the three first rule schemata. Similarly $I(x_k) = x_k$ for all $k \in \{1, \ldots, m\}$. Thus $I(\psi_i) = \sigma(L_{i1}) + \sigma(L_{i2}) + \sigma(L_{i3})$ for all $i \in \{\ 1, \ldots, r\ \}$ where

$$\begin{array}{ll}
\sigma(L_{ij}) = t \cdot x_k & \text{if } L_{ij} = x_k \\
\sigma(L_{ij}) = x_k \cdot f & \text{if } L_{ij} = \overline{x}_k
\end{array}$$

Intuitively $t \cdot x_k$ and $x_k \cdot f$ represent that variable x_k respectively appears positively or negatively in ψ_i. Hence, the valuation of atom ϱ captures the formula ϱ as follows.

$$I(\varrho) \ = \ \prod \{\ \sigma(L_{i1}) + \sigma(L_{i2}) + \sigma(L_{i3}) \ \big|\ 1 \leq i \leq r\ \} * I(\gamma)$$

and, applying the distributivity of products with respect to sums, it follows that:

$$I(\varrho) \ = \ \sum \{ \ \sigma(L_{1j_1}) * \ldots * \sigma(L_{rj_r}) * I(\gamma) \mid j_i \in \{ \ 1,2,3 \ \} \ \}$$

where $I(\gamma) \ \stackrel{\text{def}}{=} \ f \cdot x_1 \cdot t * \ldots * f \cdot x_m \cdot t$. Note that $I(\gamma)$ corresponds to the left graph depicted in Figure 30 and the rest of $I(\varrho)$ corresponds to the conjunctive

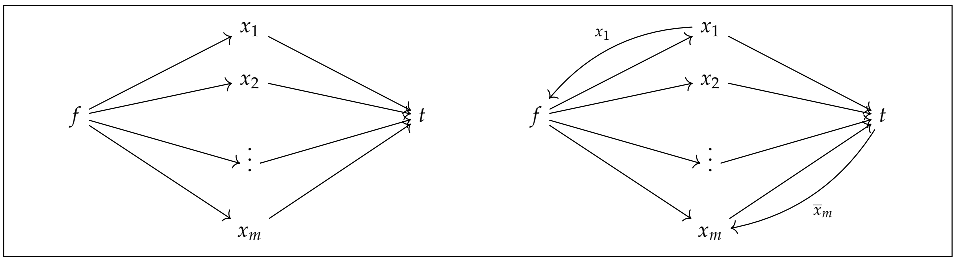

Figure 30: $I(\gamma)$ for all I and $I(\varrho)$ for $x_1 \vee \overline{x}_m$.

normal form of formula ϱ, replacing $\bigwedge$ by $\sum$, $\vee$ by $*$ and L_{ij_i} by $\sigma(L_{ij_i})$. For every disjunction in the conjunctive normal form

$$L_{1_j1} \vee \ldots \vee x_1 \ldots \vee \overline{x}_m \ldots \vee L_{rj_r}$$

there is a causal graph as the right one depicted in Figure 30. Furthermore, for every variable x_k occurring positively we add an edge from it to f and for every variable occurring negatively we add an edge from t to it. Hence, if both x_k and $\overline{x}_k$ occur in some disjunction we have an edge $t \cdot x_k$ and an edge $x_k \cdot f$. Moreover, since a causal graph is transitively closed, we must have an edge joining all the edges. That is, the graph must become the complete connected graph G_{tf}.

Clearly, $\forall x_1,\ldots,x_m \ \varrho$ is valid if and only if all disjunctions of its conjunctive normal form are valid. In turn, a disjunction is valid iff it contains two complementary literals for the same variable x_k, and the later holds iff its corresponding causal graphs is the completely connected graph. Moreover, suppose there is some non-valid disjunction, and let G be its corresponding causal graph. Since the completely connected graph is a supergraph of every possible one, then $G_{tf} < G$. That is, $G_{tf} \not\leq_{\max} I(\varrho)$. Consequently $\forall x_1,\ldots,x_m \ \varrho$ is true iff $G_{tf} \leq_{\max} I(\varrho)$. □

Theorem 7.11 (Cautious sufficient cause complete). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$, deciding whether a causal graph G is a cautious sufficient cause of A is* coNP*-complete.* □

Proof. Membership follows directly from Corollary 7.4. Hardness follows from the fact that deciding whether an atom is a cautious consequence of a standard logic program is coNP-complete and an atom is a cautious consequence of a standard logic program iff 1 is a cautious sufficient cause of such an atom. □

Theorem 7.12 (Brave sufficient cause complete)**.** *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for all $\psi \in \Psi$, deciding whether G is a brave sufficient cause of A is in Σ^P_2-complete. If P is head labelled or G is acyclic, then it is* NP*-complete.* □

Proof. Membership follows directly from Corollary 7.4. NP-hardness follows from the fact that every standard program is also a causal program and deciding whether an atom is a brave consequence of a standard logic program is NP-complete. Note that an atom is a brave consequence of a standard logic program iff 1 is a cautious sufficient cause of such an atom.

Program 7.3.

$$
\begin{array}{lllll}
f & : & f & & \\
x_k & : & x''_k \leftarrow f & & \text{for each } k \in \{1,\dots,m\} \\
t & : & x'_k \leftarrow x''_k & & \text{for each } k \in \{1,\dots,m\} \\
t & : & t & & \\
x_k & : & \psi_i \leftarrow t & & \text{if } L_{ij} = x_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\} \\
f & : & \psi_i \leftarrow x_k & & \text{if } L_{ij} = \overline{x}_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\} \\
x_k & : & x_k & & \text{for each } k \in \{1,\dots,m\} \\
 & & \gamma \leftarrow x'_1,\dots,\ x'_m & & \\
 & & \varrho \leftarrow \psi_1,\dots,\ \psi_r,\ \gamma & & \\
 & & & & \\
 & & y_k \leftarrow \mathit{not}\ \overline{y}_k & & \text{for each } k \in \{1,\dots,n\} \\
 & & \overline{y}_k \leftarrow \mathit{not}\ y_k & & \text{for each } k \in \{1,\dots,n\} \\
f & : & \psi_i \leftarrow y_k,\ t & & \text{if } L_{ij} = y_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\} \\
f & : & \psi_i \leftarrow \overline{y}_k,\ t & & \text{if } L_{ij} = \overline{y}_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\} \\
 & & \psi_i \leftarrow \overline{y}_k & & \text{if } L_{ij} = y_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\} \\
 & & \psi_i \leftarrow y_k & & \text{if } L_{ij} = \overline{y}_k \text{ for each } i \in \{1,\dots,r\},\ j \in \{1,2,3\}
\end{array}
$$

Cabalar et al. [2014b] have shown that deciding whether a causal graph G is a sufficient cause of atom A with respect to the least model I of P is Σ^P_2-complete in the case of head labelled programs. In the following, we show such proof by building a log-space reduction for deciding the truth of a formula $\varphi = \exists y_1, \ldots, y_n \forall x_1, \ldots, x_m\ \varrho$, where $\varrho = \psi_1 \vee \ldots \vee \psi_r$ and each $\psi_i = L_{i1} \wedge L_{i2} \wedge L_{i3}$ is a conjunction of three literals L_{ij} over atoms $x_1, \ldots, x_m$. Given φ, we construct a labelled program P_φ as showed in Program 7.3. This program is the result of adding the six last rules to Program 7.2. Hence, it can be shown, in a similar way as in Theorem 7.10, that φ is true if and only if the completely connected causal graph graph G_{tf} with a set of vertices $\{x_1, \ldots, x_m, t, f\}$ is a brave sufficient cause of atom ϱ. After applying distributivity, it follows that

$$I(\varrho) = \sum \{\ \sigma_I(L_{1j_1}) * \ldots * \sigma_I(L_{rj_r}) * I(\gamma) \mid j_i \in \{\ 1,2,3\ \}\ \}$$

for every causal stable model I and where $I(\gamma) \overset{\text{def}}{=} f \cdot x_1 \cdot t * \ldots * f \cdot x_m \cdot t$. Note that $I(\gamma)$ corresponds to the graph depicted in Figure 30 as in Theorem 7.10 and $\sigma_I(L_{ij_i})$ depends now on the interpretation I. In fact, if $L_{ij_i} = x_k$ of $L_{ij_i} = \overline{x}_k$, then $\sigma_I(L_{ij_i})$ does not depends on interpretation I and still has the same value as in Theorem 7.10.

$$\begin{aligned} \sigma_I(L_{ij}) &= t \cdot x_k && \text{if } L_{ij} = x_k \\ \sigma_I(L_{ij}) &= x_k \cdot f && \text{if } L_{ij} = \overline{x}_k \end{aligned}$$

On the other hand, if $L_{ij_i} = y_k$ of $L_{ij_i} = \overline{y}_k$, then

$$\begin{aligned} \sigma_I(L_{ij}) &= t \cdot f && \text{if } L_{ij} = y_k \text{ and } I \models y_k \\ \sigma_I(L_{ij}) &= 1 && \text{if } L_{ij} = y_k \text{ and } I \not\models y_k \\ \sigma_I(L_{ij}) &= t \cdot f && \text{if } L_{ij} = \overline{y}_k \text{ and } I \not\models y_k \\ \sigma_I(L_{ij}) &= 1 && \text{if } L_{ij} = \overline{y}_k \text{ and } I \models y_k \end{aligned}$$

That is, interpretation I encodes an assignment for the existential variables $y_1, \ldots, y_n$. If a disjunction contains a variable y_k of $\overline{y_k}$ which holds in I, then an edge from t to f is added to the causal graph in Figure 30 leading to the completely connected graph. The rest of the proof follows as in Theorem 7.10. □

7.3.3 Necessary cause

Theorem 7.13 (Cautious necessary cause complete). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi \rangle$ such that $\tilde{\psi}$ is computable in polynomial time for every $\psi \in \Psi$, deciding whether a causal graph G is a cautious necessary cause of an atom A is* coNP*-complete. Deciding whether a causal graph G is a necessary cause of an atom A with respect to the well-founded model is also* coNP*-complete. It is coNP-hard even if P is a positive labelled program.* □

Proof. Membership follows directly from Corollary 7.5. Cabalar et al. [2014b] have shown that deciding whether a causal graph G is a cautious necessary cause of atom A with respect to the least model I of P is coNP-complete even in the case of positive labelled programs. We will show next that proof consisting on the construction of a log-space reduction of deciding the truth of formula of the form $\varphi = \forall x_1, \ldots, x_m\ \varrho$ where $\varrho = \psi_1 \vee \ldots \vee \psi_r$ and each $\psi_i = L_{i1} \wedge L_{i2} \wedge L_{i3}$ is a conjunction of three literals L_{ij} over atoms $x_1, \ldots, x_m$. This formula ϱ will be valid iff causal graph G_{tf} with a unique edge (t, f) is a necessary cause of an atom ϱ with respect to the least model I of the following positive program.

Program 7.4.

$$
\begin{array}{lll}
x_k : x_k & & \text{for each } k \in \{1, \ldots, m\} \\
t\ \ : t & & \\
x_k : \psi_i \leftarrow t & & \text{if } L_{ij} = x_k \text{ for each } i \in \{1, \ldots, r\},\ j \in \{1,2,3\} \\
f\ \ : \psi_i \leftarrow x_k & & \text{if } L_{ij} = \overline{x}_k \text{ for each } i \in \{1, \ldots, r\},\ j \in \{1,2,3\} \\
\quad\ \varrho \ \leftarrow \psi_1, \ldots,\ \psi_r & &
\end{array}
$$

The idea behind this transformation is similar to that done in Program 7.2 in Theorem 7.10. Each conjunction $\psi_i = L_{i1} \wedge L_{i2} \wedge L_{i3}$ is captured by the evaluation of atom ψ_i in the following way:

$$I(\psi_i) \ = \ \sigma(L_{i1}) + \sigma(L_{i2}) + \sigma(L_{i3})$$

where

$$
\begin{array}{ll}
\sigma(L_{ij}) = t \cdot x_k & \text{if } L_{ij} = x_k \\
\sigma(L_{ij}) = x_k \cdot f & \text{if } L_{ij} = \overline{x}_k
\end{array}
$$

Hence, the valuation of atom ϱ captures the formula ϱ as follows.

$$I(\varrho) \ = \ \prod \{\ \sigma(L_{i1}) + \sigma(L_{i2}) + \sigma(L_{i3}) \ \big|\ 1 \leq i \leq r\ \}$$

and, applying distributivity of products with respect to sums, it follows that:

$$I(\varrho) \ = \ \sum \{\ \sigma(L_{1j_1}) * \ldots * \sigma(L_{rj_r}) \ \big|\ j_i \in \{\ 1,2,3\ \}\ \}$$

After replacing sums by conjunctions and products by disjunctions, it is easy to see that the above expression corresponds to the conjunctive normal form of the formula ϱ. The formula $\forall x_1, \ldots, x_m\ \varrho$ is true iff all disjunctions of the form $L_{1j_1} \vee \ldots \vee L_{rj_r}$ of its conjunctive normal form are valid, which in its turn, is the case iff there is a pair of complementary $L_{1j_i} = x_k$ and $L_{rj_l} = \overline{x}_k$. When this happens, $\sigma(L_{1j_1}) * \ldots * \sigma(L_{rj_r})$ satisfies:

$$\sigma(L_{1_j1}) * \ldots * t\cdot x_k * \ldots * x_k\cdot f * \ldots * \sigma(L_{rj_r}) \;\leq\; t\cdot x_k * x_k\cdot f$$

Furthermore, $t\cdot x_k * x_k\cdot f \;\leq\; t\cdot x_k\cdot f \;\leq\; t\cdot f \;\leq\; t\cdot x_k\cdot f \;\leq\; t\cdot f$. That is, a disjunction of the form $L_{1j_1} \vee \ldots \vee L_{rj_r}$ is valid iff it corresponding causal term satisfies $\sigma(L_{1j_1}) * \ldots * \sigma(L_{rj_r}) \;\leq\; t\cdot f$. Consequently, $I(\varrho) \leq t\cdot f$ iff all disjunctions are valid, and the later holds iff the formula is valid. □

Theorem 7.14 (Brave necessary cause complete). *Given a causal program P over a signature $\sigma = \langle At, Lb, \Psi\rangle$ such that $\tilde{\psi}$ is computable in polynomial time for every $\psi \in \Psi$, deciding whether a causal graph G is a brave necessary cause of an atom A with respect to the least model I of P is Σ^P_2-complete. It is Σ^P_2-hard even if P is a labelled program.* □

Proof. Membership follows directly from Corollary 7.5. In the following, we show the reduction introduced in Cabalar et al. [2014b] of a quantified Boolean formula $\varphi = \exists y_1,\ldots,y_n \forall x_1,\ldots,x_m\ \varrho$ where $\varrho = \psi_1 \vee \ldots \vee \psi_r$ and, in its turn, each $\psi_i = L_{i1} \wedge L_{i2} \wedge L_{i3}$ is a conjunction of three literals L_{ij} over atoms $x_1,\ldots,x_m$. The formula ϱ will be valid iff causal graph G_{tf} with a unique edge (t,f) is a necessary cause of an atom ϱ with respect to the least model I of the following positive program.

Program 7.5.

$$\begin{array}{lll}
x_k : x_k & & \text{for each } k \in \{1,\ldots,m\} \\
t \;\; : t & & \\
x_k : \psi_i \leftarrow t & & \text{if } L_{ij} = x_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
f \;\; : \psi_i \leftarrow x_k & & \text{if } L_{ij} = \overline{x}_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
\quad\;\; \varrho \;\leftarrow \psi_1,\ldots,\ \psi_r & & \\
 & & \\
\quad\;\; y_k \leftarrow not\ \overline{y}_k & & \text{for each } k \in \{1,\ldots,n\} \\
\quad\;\; \overline{y}_k \leftarrow not\ y_k & & \text{for each } k \in \{1,\ldots,n\} \\
f \;\; : \psi_i \leftarrow y_k,\ t & & \text{if } L_{ij} = y_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
f \;\; : \psi_i \leftarrow \overline{y}_k,\ t & & \text{if } L_{ij} = \overline{y}_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
\quad\;\; \psi_i \leftarrow \overline{y}_k & & \text{if } L_{ij} = y_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\} \\
\quad\;\; \psi_i \leftarrow y_k & & \text{if } L_{ij} = \overline{y}_k \text{ for each } i \in \{1,\ldots,r\},\ j \in \{1,2,3\}
\end{array}$$

Program 7.5 is the result of adding the last six rules to Program 7.4 in a similar manner as happened between Program 7.3 and Program 7.2 respectively. In the same way, each interpretation I encodes an assignment for the variables $y_1, \ldots, y_n$. In this case, if a disjunction contains a literal y_k or $\overline{y}_k$ which holds w.r.t. I, then the corresponding causal graph G will contain an edge from t to f and hence $G \leq t \cdot f$. The rest of the proof follows as in Theorem 7.7. □

It is worth to mention that proofs of Theorems 7.13 and 7.14 do not rely on a cyclic causal graph as the constructions for the case for sufficient causes. In fact, the causal graphs used in the proof only contain an edge joining label t with label f. However, there are two rules with different heads labelled with x_k, that is, the program is not head labelled. Our conjecture is that for head labelled programs with a unique atom labelling complexity will drop a level, that is to P and NP for cautions and brave reasoning respectively. The basis of this conjecture relies on showing that the $\leq$-maximal causal graphs computed by the T_P operator can be obtained even if the $(*)$ operation in its definition is replaced by the ideal formed by the causal graphs resulting from the union $\cup$ of the $\leq$-maximal causal graphs corresponding to the literals in the body of each rule. In case that this conjecture holds, a polynomial time procedure for deciding $\tilde{T} \leq G$ can be constructed because all the branches of $\tilde{T}$ can be checked independently.

7.4 HASCAUSED PREDICATE

In this section, we will analyse the particular case in which all causal literals are either standard atoms or causal literals "$\mathbf{hascaused}(agent, A)$" (Definition 6.2). As we commented in Chapter 6, standard atoms are a particular case of causal literals in which the test function ψ^1 holds for every causal graph. It is clear that a procedure $\tilde{\psi}^1$ corresponding to ψ^1 can be decided in polynomial time, in fact $\tilde{\psi}^1$ may be the identity, and so, it can be decided in constant time. As a consequence, all the results of the previous section are applicable to labelled programs, that is, those programs which only contain standard atoms.

In the rest of this section, we will show that we may define a function $caused(\mathcal{A}, G, \tilde{T})$ that computes $I(\mathbf{hascaused}(\mathcal{A}_a, A))$ with respect to a set of actions $\mathcal{A}$ and an interpretation I when $\tilde{T}$ is a tg-graph that represents $I(A)$ and $G = \prod \mathcal{A}_a$. For this purpose, we define a recursive function $caused(\mathcal{A}, G, \tilde{T}, v, l)$

where $\mathcal{A}$ is the set of labels corresponding to actions, G is a causal graph, $\tilde{T}$ is a t-graph corresponding to the evaluation of atom A with respect to a t-interpretation $\tilde{I}$ representing I, v is a vertex of $\tilde{T}$ and, finally, $l \in Lb \cup \{1\}$ is a label or 1. Then, we define $\mathit{caused}(\mathcal{A},G,\tilde{T}) \stackrel{\text{def}}{=} \tilde{T}^0$ if $\tilde{T} = 0$ and $\mathit{caused}(\mathcal{A},G,\tilde{T}) \stackrel{\text{def}}{=} \mathit{caused}(\mathcal{A},G,\tilde{T},v,1)$, where v is the root of $\tilde{T}$, otherwise. Consequently, this function must satisfy that:

$$\mathit{caused}(\mathcal{A},G,\tilde{T}) \ = \ \sum\{\ G' \in \mathbf{C}_{Lb} \mid G \leq \psi_{\mathcal{A}}(G') \text{ and } G' \leq \tilde{T}\ \}$$

which corresponds to the evaluation of $I(\mathbf{hascaused}(\mathcal{A},A))$ when $\tilde{T}$ is a t-graph representing $I(A)$.

Below, we define the function $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l)$. Recall that function $\psi_{\mathcal{A}}$ was introduced in Definition 6.2. For the clarity sake, we are making the following assumptions: $\psi_{\mathcal{A}}(1) = 1$ and $(1,w)$ and $(w,1)$ represent the edge (w,w). Then $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l)$ is recursively defined by:

1. $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \stackrel{\text{def}}{=} \ \tilde{T}^u \cdot l$
   if $f_V(v) = u \in Lb \cup \{1\}$ and $(w_u,w_l) \in G$
   with $\psi_{\mathcal{A}}(u) = w_u$ and $\psi_{\mathcal{A}}(l) = w_l$
2. $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \stackrel{\text{def}}{=} \mathit{caused}(\mathcal{A},G,\tilde{T},v_l,u) \cdot l$
   if $f_V(v) \in \{\cdot\}$ and $(w_u,w_l) \in G$
   with $\{(v,v_l)^l,(v,v_r^u)^r\} \subseteq E(\tilde{T})$, $f_V(v_r) = u$,
   $\psi_{\mathcal{A}}(u) = w_u$ and $\psi_{\mathcal{A}}(l) = w_l$.
3. $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \stackrel{\text{def}}{=} \sum\{\ \mathit{caused}(G,\tilde{T},v',l) \mid (v,v') \in E(\tilde{T})\ \}$
   if $f_V(v) \in \{+\}$
4. $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \stackrel{\text{def}}{=} \prod\{\ \mathit{caused}(G,\tilde{T},v',l) \mid (v,v') \in E(\tilde{T})\ \}$
   if $f_V(v) \in \{*\}$
5. $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \stackrel{\text{def}}{=} \tilde{T}^0$ otherwise

Intuitively, this procedure goes through all the branches of $\tilde{T}$ checking that all edges resulting from their evaluation are contained in G. If some branch contains an edge that is not in G means that such branch corresponds to causes G' that do not satisfies $G \supseteq \psi_{\mathcal{A}}(G')$ (that is $G \not\leq \psi_{\mathcal{A}}(G')$ holds). Therefore, G' needs to be removed, that is, the branch containing G' is replaced by $\tilde{T}^0$.

Proposition 7.6. *Given a set of labels $\mathcal{A} \subseteq Lb$ and causal graph G, it holds that*

$$\mathit{caused}(\mathcal{A},G,\tilde{T},v,l) \ = \ \sum\{\ G' \in \mathbf{C}_{Lb} \mid G \leq \psi_{\mathcal{A}}(G') \textit{ and } G' \leq \tilde{T}\ \} \cdot l$$

for any t-graph $\tilde{T} \in \tilde{\mathbf{V}}_{Lb}$. Furthermore, $\mathit{caused}(\mathcal{A},G,\tilde{T},v,l)$ can be computed in polynomial time. □

Proof. The proof follows by structural induction assuming that it holds for every children of a vertex and showing then that it holds for such a vertex. The details can be found in Appendix A on page 275. □

Corollary 7.6. *Given a set of labels $\mathcal{A} \subseteq Lb$ and causal graph G, it holds that*

$$caused(\mathcal{A}, G, \tilde{T}) \ = \ \sum\{\ G' \in \mathbf{C}_{Lb} \mid G \leq \psi_{\mathcal{A}}(G') \text{ and } G' \leq \tilde{T}\ \}$$

for any t-graph $\tilde{T} \in \tilde{\mathbf{V}}_{Lb}$. Hence $caused(\mathcal{A}, G, I(A)) = I(\mathbf{hascaused}(G, A))$. Furthermore, $caused(\mathcal{A}, G, \tilde{T})$ can be computed in polynomial time. □ □

Proof. The proof directly follows from Proposition 7.6 by noting that $l = 1$ in the definition of $caused(\mathcal{A}, G, \tilde{T})$ and 1 is the application identity. □

Now, we can use the function $caused(\mathcal{A}, G, \tilde{T})$ to prove Proposition 7.4.

Proof of Proposition 7.4. Note that by taking $\mathcal{A} = Lb$ in Corollary 7.6 it follows that $\psi_{\mathcal{A}}$ is the identity function, and hence

$$caused(Lb, G, \tilde{T}) \ = \ \sum\{\ G' \in \mathbf{C}_{Lb} \mid G \leq G' \leq \tilde{T}\ \}$$

Then, $caused(Lb, G, \tilde{T}) = 0$ iff $G \leq \tilde{T}$. Hence, deciding whether $G \leq \tilde{T}$ is computable in polynomial time.

Furthermore, $G \leq_{\max} \tilde{T}$ iff $G \leq \tilde{T}$ and $G \nless G'$ for all G' such that $G' \leq \tilde{T}$, and the last holds iff $G \leq \tilde{T}$ and either $G \not\supseteq G'$ or $G' \nleq \tilde{T}$ for all $G' \in \mathbf{C}_{Lb}$. Hence, we can guess a causal graph G' and check in polynomial time whether either $G \not\supseteq G'$ or $G' \nleq \tilde{T}$ for all causal graph G'. That is, deciding whether $G \leq_{\max} \tilde{T}$ is in coNP.

In order to show that $\tilde{T} \leq \tilde{T}'$ is in coNP, just note that $\tilde{T} \leq \tilde{T}'$ iff it holds that $G' \leq \tilde{T}$ implies $G' \leq \tilde{T}'$ for all causal graph G'. Hence, we may just guess a causal graph G' and then test whether ($G' \leq \tilde{T}$ implies $G' \leq \tilde{T}'$) holds by a pair of calls to the *caused* function. Furthermore, building a t-graph $\tilde{T}'$ that represents a causal graph G is trivial, and hence, deciding whether $\tilde{T} \leq G$ is also in coNP. □

Finally, we get the results for programs which only contain standard atoms and causal literals of the form "$\mathbf{hascaused}(\mathcal{A}_a, A)$".

Theorem 7.15 (HasCaused complete)**.** *Given a program P only containing standard atoms and causal literal of the form "$\mathbf{hascaused}(\mathcal{A}_a, A)$" and giving a causal graph G and an atom A,deciding whether* *i)* *an atom A is a consequence of P is* P*-complete when P is positive.*

ii) *an atom A is a consequence of P w.r.t. the well-founded model is* P*-complete.*

iii) *an atom A is a brave consequence of P is* NP*-complete.*

iv) *an atom A is a cautions consequence of P is* coNP*-complete.*

v) *G is a sufficient cause of A is* coNP*-complete when P is positive. It is P-complete when furthermore either P is head labelled and δ is a unique atom labelling, or G is acyclic.*

vi) *G is a sufficient cause of A w.r.t. the well-founded model is* coNP*-complete. It is P-complete when, in addition, either P is head labelled and δ is a unique atom labelling, or G is acyclic.*

vii) *G is a cautions sufficient cause of A is* coNP*-complete.*

viii) *G is a brave sufficient cause of A is Σ_2^{P}-complete. It is* NP*-complete when in addition P is head labelled and δ is a unique atom labelling, or G is acyclic.*

ix) *G is a necessary cause of A is* coNP*-complete when P is positive.*

x) *G is a necessary cause of A w.r.t. the well-founded model is* coNP*-complete.*

xi) *G is a cautions necessary cause of A is* coNP*-complete.*

xii) *G is a brave necessary cause of A is Σ_2^{P}-complete.* □

Proof. From Corollary 7.6, computing $I(\textbf{hascaused}(\mathcal{A}_a, A))$ is feasible in polynomial time. Hence, (i) and (ii) directly follows from Theorem 7.8, while (iii) and (iv) follows from Theorem 7.9. Similarly (v) and (vi) follow from Theorem 7.10, while (vii) and $(viii)$ respectively follow from Theorems 7.11 and 7.12. Finally, (ix), (x) (xi) follow from Theorem 7.13, while (xii) follows from Theorem 7.14. □

The table in Figure 28 at the beginning of this chapter summarised these results in Theorem 7.15.

8 RELATED WORK

There exists a vast literature on causal reasoning in Artificial Intelligence. Papers on reasoning about actions and change by Lin [1995], Thielscher [1997], McCain and Turner [1997] or Turner [1997] have been traditionally focused on using causal inference to solve representational problems (mostly, the frame, ramification and qualification problems) without paying much attention to the derivation of cause-effect relations. Chapter 5 shows how some of the traditional problems of this area can be represented as a logic program. On the other hand, perhaps the most established AI approach for causality is relying on *causal structural models* [Pearl, 2000, Halpern and Pearl, 2005, Halpern, 2008] (see [Halpern and Hitchcock, 2011] for a more actualized view). As commented in the introduction, in this approach, it is possible to conclude cause-effect relations like "A has caused B" from the behaviour of structural equations by applying, under some contingency, the counterfactual interpretation from Hume [1748]: "had A not happened, B would not have happened." As discussed by Hall [2004, 2007], this approach relays on the idea that counterfactual-dependence is enough for causation. As opposed to this, Hall considers a different (and incompatible) definition, called *production*, where causes must be connected to their effects via *sequences of causal intermediates*, something that, as we will see in Section 8.4, is closer to our explanations in terms of causal graphs. In that section, we explore how the idea of *contributory cause* can be captured by our formalism, and the relation between this concept and Halpern and Pearl's concept of *actual cause*. In particular, we show that both concepts coincide in those examples in which the causal concepts of dependence and production introduced by Hall also agree, and differs in those in which production and dependence do not agree. To the best of our knowledge, the problem of deriving further conclusions for statements of the form "$\mathit{hascaused}(A,B)$" has not been considered in the literature yet.

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

Focusing on LP, our work obviously relates to explanations as provided by approaches to debugging and justifications in ASP by Gebser et al. [2008], Pontelli et al. [2009], Schulz and Toni [2013, 2014] or Damásio et al. [2013]. In particular, from a technical point of view, the why-not provenance approach followed by Damásio et al. [2013] is the most close of them. This approach also relies on a multivalued semantics in which values form a Boolean algebra. Section 8.1 summarises this approach and proposes a new semantics that generalises both, ours and why-not provenance. As a byproduct of this new semantics we obtain a formal relation between both approaches. Furthermore, in Section 8.4, we will see that this new semantics seems useful for capturing, in a single formalism, the above concepts of production and dependence, although a formal study of dependence is left for future work. In Section 8.2, we explore the relation between the sufficient causes of a literal and its off-line justifications as defined by Pontelli et al. [2009]. Intuitively, sufficient causes can be seen as the subgraph of positive dependences of an off-line justification. It is worth to note that sufficient causes are required to be minimal (causes do not contain non necessary events for being sufficient), while off-line justifications are not. From a technical point of view, off-line justifications are built externally to the logic and, although they are provided as terms for being handled by an external Prolog program, they are not first class citizens of the ASP semantics. Section 8.3 explores the relation between our work and the concept of ABAS justifications introduced by Schulz and Toni [2013, 2014]. Schulz and Toni depart from an argument based reading of logic programs instead of the causal reading we are assigning to them. In particular, in an argument based reading, negation as failure is understood as a counterargument to the conclusion, and therefore those counterarguments (negative dependences) are required to be included in justifications. This contrasts with our reading of negation as failure as representing default behaviour, which leads us to the non inclusion of the abnormal behaviour (negative dependences) in sufficient causes.

Pereira et al. [1991] and Denecker and Schreye [1993] also define different semantics in terms of justifications, but do not provide calculi for them. As in the above works, justifications usually contain all possible ways to derive a given fact. Vennekens [2011] translates Halpern and Pearl's concept of actual cause in the context of CP-logic and shares a similar causal orientation where only the cause-effect relations that “break the norm” should be considered relevant.

8.1 WHY-NOT PROVENANCE JUSTIFICATIONS

Why-not provenance justifications [Damásio et al., 2013] is a declarative and logical approach for helping to understand why a given literal is true or false in a model of some program. In this sense, it is clear that our approach and why-not provenance justifications share a similar aim. However, a more close look shows a major difference in what is considered to be a justification. For instance, consider a program formed by the following labelled rule:

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

It is easy to see that this program does not entail p, and so, in our approach there is no cause for it. On the other hand, the formula $\neg not\ p$ is a why-not provenance justification of p pointing out that p would be true if the program would have contained a fact p. In this sense, we may distinguish two different kinds of why-not provenance justifications. For instance, $\neg not\ p$ is actually a *non-real* justification explaining why p is entailed by the program (in fact the program does not entail p) but rather an *hypothetical justification* explaining how we can make p hold by modifying the program. The second kind, that we would call *real justifications*, are those that actually justify why p is entailed by the program without modifying it. It is easy to see that our approach is focused only in real justifications, so most of this section will be focused on real justifications.

Another major difference is how both approaches deal with negation. As we will see later, treatment of negation will also be the major difference when compared to other approaches of justifications in LP like off-line and ABAS justifications. We assume that negation represents an exception, or deviation from the default behaviour. Hence, sufficient causes do not include dependences through negation. For instance, we consider the following variation of Program 4.4 obtained by removing the fact *oxygen*:

Program 8.1.

$$\begin{array}{llll} f: & fire \leftarrow match, not\ ab & \qquad match: & match \\ d: & ab \quad \leftarrow \overline{oxygen} & & \end{array}$$

This program captures the match-oxygen scenario (Example 1.7) and entails $fire$. Accordingly, in our approach, there is a single cause for it, which corresponds to the causal term $match \cdot f$. This cause do not makes any reference to any abnormality that could, but did not, prevent the effect, in particular the absence of $oxygen$. On the other hand, the why-not provenance information of $fire$ is:

$$\begin{aligned}(\neg not\ fire)\ &\vee\ (match \wedge f \wedge not\ ab \wedge not\ \overline{oxygen})\\ &\vee\ (match \wedge f \wedge not\ ab \wedge \neg d)\end{aligned}$$

Each conjunction of this formula is a why-not provenance justification. The first, $\neg not\ fire$ is an hypothetical justification in similar manner as $\neg not\ p$ in the previous example. Similarly, the third one points out that $fire$ would be justified if the fact ab (stated by a conjunct $not\ ab$) were kept absent and rule d had been removed. The second one, $match \wedge f \wedge not\ ab \wedge not\ \overline{oxygen}$ is, in its turn, the real justification of $fire$, that is, it holds because of rule f and the fact $match$ are in the program and the facts ab and $\overline{oxygen}$ are not. Compared to $match \cdot f$, this "real" why-not provenance justification, further includes $ab \wedge \overline{oxygen}$ meaning that ab and $\overline{oxygen}$ cannot be in the program. This information may be useful for understanding why $fire$ holds or not, for instance for debugging a program, but it is somehow redundant from a causal point of view. For instance, consider the Yale Shooting scenario where, in addition we have the possibility of unloading the gun. The program representing this scenario may contain the following rules for loading and unloading the gun.

Program 8.2.

$$\begin{aligned}
& & loaded_{s+1} &\leftarrow loaded_s, not\ \overline{loaded}_{s+1}\\
u_{s+1} &: & \overline{loaded}_{s+1} &\leftarrow unload_s\\
l_{s+1} &: & loaded_{s+1} &\leftarrow load_s\\
load_1 &: & load_1 &
\end{aligned}$$

In the unique causal answer set of this program, for any situation $s \geq 1$, the cause of $load_s$ is just $load_1 \cdot l_2$. On the other hand, the unique real why-not provenance justification will be:

$$\begin{aligned}load_1 \wedge l_2 \wedge not\ \overline{loaded}_2 \wedge \ldots \wedge not\ \overline{loaded}_s&\\ \wedge\, not\ unload_1 \wedge \ldots \wedge not\ unload_{s-1}&\end{aligned} \tag{97}$$

which explicitly makes reference to the absence of the action $unload_i$ for every situation between the load and the current state. In addition to this one, there will be several hypothetical why-not provenance justifications as:

$$\begin{aligned}
&\neg not\ \overline{load_2} \wedge o_2 \wedge not\ \overline{loaded_2} \wedge \ldots \wedge not\ \overline{loaded_s} \wedge not\ unload_1 \wedge \ldots\\
&\neg not\ \overline{load_3} \wedge o_3 \wedge not\ \overline{loaded_2} \wedge \ldots \wedge not\ \overline{loaded_s} \wedge not\ unload_1 \wedge \ldots\\
&\ldots
\end{aligned}$$

each one stating that the gun would be loaded if it had been loaded in situations 2,3,... Going back to (97), by removing every conjunct of the form *not A*, we will obtain $load_1 \wedge l_2$, whose conjuncts are exactly the vertices of the causal graph represented by $load_1 \cdot l_2$. However, in general, this is not always the case. Consider for instance, that we replace the above rule u_{s+1} by the rules

$$\begin{aligned}
a_{s+1}: &\quad \overline{loaded_s} \leftarrow not\ p_s\\
b_{s+1}: &\quad p_s
\end{aligned}$$

This does not affect the causes of $load_s$. The only cause still is $load_1 \cdot l_2$. But the unique real why-not provenance justification is now:

$$load_1 \wedge l_2 \wedge b_2 \wedge \ldots \wedge b_{s-1} \wedge not\ \overline{loaded_2} \wedge \ldots \wedge not\ \overline{loaded_s} \tag{98}$$

In this example l_2 is indistinguishable from $b_2, \ldots, b_{s-1}$, although only l_2 is a vertex of the cause of $load_s$. That is, given a why-not provenance justification, a corresponding cause cannot be built. Furthermore, from the cause $load_1 \cdot l_2$, the rest of conjuncts that forms the why-not provenance justification (97) cannot be obtained either.

Despite these differences, it is interesting to notice that from a technical point of view both approaches are also quite similar, relying both on a multivalued semantics with a set of values that form a completely distributive lattice. In particular, provenance values form a free Boolean algebra.

Definition 8.1 (Provenance values)**.** *Given a set of labels Lb, we denote by* Lb^{not} *the set* $Lb^{not} \overset{\text{def}}{=} \{\text{not } A \mid A \in At\}$. *A* provenance term, *t is recursively defined as one of the following expressions* $t ::= l \mid \bigwedge S \mid \bigvee S \mid \neg t$ *where* $l \in Lb \cup Lb^{not}$, *t is a provenance term and S is a (possibly empty and possible infinite) set of provenance terms.* Provenance values *are the equivalence classes of provenance terms under the equivalences of the Boolean algebra. Given a signature* $\langle At, Lb \rangle$, *we denote by* $\mathbf{B}_{Lb}$ *the set of provenance values generated by a set of labels Lb.* □

Recall that $(+)$ and $(*)$ operators, in our approach, are respectively the least upper bound and the greatest lower bound of a completely distributive lattice, and so they respectively correspond to the conjunction $(\wedge)$ and disjunction $(\vee)$ Boolean operators. Then, the difference between values of both approaches relies on the lack of the application $(\cdot)$ operator and the availability of a negation $(\neg)$ operator by provenance terms. On the one hand, the lack of the application $(\cdot)$ operator means that why-not provenance justifications cannot represent the causal structure of dependences, but just which literals and rules are involved in that justification. On the other hand, the availability of a negation $(\neg)$ operator is useful for answering questions of the form "why the fire has not started?" when, for instance, the oxygen is absent in the match example.

The why-not provenance approach assumes that there is a label of the form *not* A for every literal A, so that, for any given signature $\langle At, Lb\rangle$ the set of labels must be extended with the labels in Lb^{not}. Labels of the form *not* A, are used to state that the fact A cannot be added to the program, and labels of the form $\neg$*not* A are used to state that by adding the fact A we will obtain a justification.

In order to illustrate how the negation operator $(\neg)$ allows answering questions of the form "why the fire has not started?" consider the Program 4.3 from Chapter 4 consisting of the following rules:

Program 8.3.

$$\begin{array}{llcl@{\qquad\qquad}rcl} f: & fire & \leftarrow & match, not\ ab & match & : & match \\ d: & ab & \leftarrow & \overline{oxygen} & \overline{oxygen} & : & \overline{oxygen} \end{array}$$

In this program the literal $fire$ does not hold and, consequently, there is no cause for it. So the answer to the question "why the fire has not started?" could be "because, by default, there are not fires." However, we may be still interested in a more detailed answer: "by default, there are not fires, but also by default, matches cause fires." On the other hand, there are why-not provenance justifications for $fire$. In fact, in this program, the why-not provenance information of $fire$ is:

$$(\neg not\ fire) \ \vee\ (f \wedge match \wedge \neg d) \ \vee\ (f \wedge match \wedge \neg\overline{oxygen}) \tag{99}$$

stating that $fire$ would be true if either the fact $fire$ would have been added to the program or either the rule d or the fact $\overline{oxygen}$ would have been removed. The negation of (99):

$$(not\ fire \wedge \neg f) \ \vee\ (not\ fire \wedge \neg match) \ \vee\ (not\ fire \wedge a \wedge \overline{oxygen}) \tag{100}$$

constitutes the why-not provenance information for *not* $fire$. Each conjunction of (100) is a why-not provenance justification for *not* $fire$. The first two, *not* $fire \wedge \neg f$ and *not* $fire \wedge \neg match$ are hypothetical justifications that points out that $fire$ would be false if the fact $fire$ were not in the program and either the fact $match$ or the rule f had been removed. The last one, *not* $fire \wedge d \wedge \overline{oxygen}$, explains that $fire$ does not hold because the fact $fire$ is not in the program and rule d and the fact $\overline{oxygen}$ are in it.

In the rest of this section we will extend causal values in order to capture both the causes of a literal and explanations for negative literals. This formalism takes advantage of the strengths of both approaches and will allow us to relate them formally.

Definition 8.2 (CP-values). *A* CP-term, *t is recursively defined as one of the following expressions $t ::= l \mid \sum S \mid \prod S \mid t_1 \cdot t_2 \mid {\sim} t_1$ where $l \in Lb \cup Lb^{not}$, t_1 and t_2 are CP-terms and S is a (possibly empty and possible infinite) set of CP-terms.* CP-values *are the equivalence classes of provenance terms under the equivalences of causal values (equations in Figures 13 and 14 and 15) plus:*

i) pseudo-complement and DeMorgan laws

$$\begin{array}{ccc} t * {\sim}t = 0 & \qquad & {\sim}(t+u) = ({\sim}t * {\sim}u) \\ {\sim}{\sim}{\sim}t = {\sim}t & \qquad & {\sim}(t * u) = ({\sim}t + {\sim}u) \end{array}$$

ii) weak excluded middle $\;{\sim}t + {\sim}{\sim}t = 1$

iii) negation of application $\;{\sim}(t \cdot u) = {\sim}(t * u)$

The set of CP-values generated by a set of labels B is denoted by $\tilde{\mathbf{V}}_{Lb}$. □

CP-values extend the notion of causal values (Definition 3.9) with a negation (${\sim}$) operator which, in contrast to the classical negation ($\neg$) of the why-not provenance approach, does not satisfy the exclude in the middle principle. In fact $\langle \tilde{\mathbf{V}}_{Lb}, +, *, {\sim}, 0, 1 \rangle$ forms a pseudo-complemented completely distributive lattice, which is an algebraic structure weaker than a Boolean algebra. To illustrate the importance of this fact we appeal to the above load-unload example where the why-not provenance justification of $load_s$ corresponded to the conjunction (98). Instead, of this conjunction we will assign to $load_s$ the CP-term:

$$\begin{aligned} (\; & load_1 * {\sim}{\sim}b_2 * \ldots * {\sim}{\sim}b_{s-1} \\ & \qquad * {\sim}{\sim}not\ \overline{loaded}_2 * \ldots * {\sim}{\sim}not\ \overline{loaded}_{s-1} \;) \cdot l_2 \end{aligned} \tag{101}$$

By replacing the product ($*$) and application ($\cdot$) symbols in CP-terms by conjunctions ($\wedge$) and removing the double negation ($\sim\sim$) for each conjunct in (101) we obtain the previous why-not provenance justification in (98). Furthermore, by removing from CP-terms every subterm containing doubly negated labels we obtain $load_1 \cdot l_2$ which is the cause of $load_s$. That is, by removing some subterms we may be able to obtain causes and why-not justifications from CP-terms.

It is also worth to note that by following the equivalences in Definition 8.2, every CP-term t can be rewritten as an equivalent CP-term u where negation ($\sim$) is not in the scope of any other operation. That is, negations are only applied to labels or negations of labels.

Definition 8.3 (Negated normal form)**.** *A CP-term where negation is not in the scope of any other operation (and it is only applied to labels or negated labels) is said to be in* negation normal form*. Furthermore, the terms* 0*,* 1 *and terms that only contain labels and negations are said to be* atomic*.* □

CP-interpretations are functions $\tilde{I} : At \longrightarrow \tilde{\mathbf{V}}_{Lb}$ mapping each atom into a CP-value. We will denote by $\tilde{\mathbf{I}}_\sigma$ the set of CP-interpretations over a signature $\sigma = \langle At, Lb \rangle$. Since causal values are those CP-values that have some representative term t with no negation ($\sim$), causal interpretations are those CP-interpretations that map each literal into a causal value. Recall that, as usual in lattices, we say that a CP-term t is smaller than other u, in symbols $t \leq u$, iff $t * u = t$ iff $t + u = u$. This relation is extended to CP-interpretations as it has been extended for causal interpretations, that is, $\tilde{I} \leq \tilde{J}$ iff $\tilde{I}(A) \leq \tilde{J}(A)$ for any atom A. The evaluation of a term t with respect to a CP-interpretation $\tilde{I}$, in symbols $\tilde{I}(t)$ is the equivalence class of t. The value assigned to a negative literal *not* A by an CP-interpretation $\tilde{I}$, denoted as $\tilde{I}(not\ A)$, is defined as: $\tilde{I}(not\ A) \overset{\text{def}}{=} {\sim}\tilde{I}(A)$. The evaluation of negative literals with respect to CP-interpretations is different that their evaluation with respect to causal interpretations. For a causal interpretation I the evaluation of a negative literal *not* A was defined as $I(not\ A) \overset{\text{def}}{=} 1$ if $I(A) = 0$ and $I(not\ A) \overset{\text{def}}{=} 0$ otherwise. This difference is what will allow capturing justifications for negative literals. We may then, introduce the notion of reduct for CP-interpretations in the following way.

Definition 8.4 (CP-reduct)**.** *Given a program* P *and a CP-interpretation* $\tilde{I}$*, we denote by* $P^{\tilde{I}}$ *the positive program containing a rule of the form*

$$l_R :\ A \ \leftarrow\ B_1, \ldots, B_m,\ \tilde{I}(\text{not } C_1), \ldots, \tilde{I}(\text{not } C_n) \tag{102}$$

for each rule in P of the form (92). □

Furthermore, for positive programs, we may define CP-models in the same way as we have defined causal stable models (Definition 4.2), but replacing causal interpretations by CP-interpretations.

Definition 8.5 (CP-model). *Given a positive labelled rule R of the form* (48) *(with $m = 0$), we say that an CP-interpretation $\tilde{I}$* satisfies *R, in symbols $\tilde{I} \models R$, if and only if the following condition holds:*

$$\big(\ \tilde{I}(B_1) * \ldots * \tilde{I}(B_m)\ \big) \cdot l_R \cdot \delta(A) \ \leq\ \tilde{I}(A) \tag{103}$$

Given a positive program P, we say that a CP-interpretation $\tilde{I}$ is a CP-model of P, in symbols $I \models P$, if and only if, I is the $\leq$-minimal interpretation that satisfies all rules R in P. □

For positive programs, we may also define a *direct consequences operator* [van Emden and Kowalski, 1976] for computing the least CP-model of a program by iterating it from the bottom CP-interpretation which maps each literal to 0.

Definition 8.6 (CP direct consequences). *Given a positive labelled program P over a signature $\sigma = \langle At, Lb, \delta \rangle$, the operator of* direct consequences *is the function $\tilde{T}_P : \mathbf{\tilde{I}}_\sigma \longrightarrow \mathbf{\tilde{I}}_\sigma$ such that $\tilde{T}_P(\tilde{I})(A)$ is given by*

$$\sum \big\{\ \big(\ \tilde{I}(B_1) * \ldots * \tilde{I}(B_m)\ \big) \cdot l_R \cdot \delta(A)\ \big|\ R \in P \text{ and } head(R) = A\ \big\}$$

for any CP-interpretation $\tilde{I}$ and any literal $A \in At$. □

Definition 8.6 is exactly the same as Definition 4.3 but replacing causal interpretations by CP-interpretations. Hence, similar results are obtained.

Theorem 8.1 (CP direct consequences). *Let P be a positive labelled program. Then,*

i) $\mathtt{lfp}(\tilde{T}_P)$ *is the least model of P, and*

ii) $\mathtt{lfp}(\tilde{T}_P) = T_P^\omega(\mathbf{\tilde{0}})$.

If furthermore P is finite and has n rules, then $\mathtt{lfp}(\tilde{T}_P) = \tilde{T}_P^n(\mathbf{\tilde{0}})$. □

Proof. The proof can be found in Appendix A on page 280. □

In order to define the CP well-founded model of a program, we define first an operator $\tilde{\Gamma}_P : \mathbf{\tilde{I}}_\sigma \longrightarrow \mathbf{\tilde{I}}_\sigma$ mapping CP-interpretations into the least model of the positive program $P^{\tilde{I}}$. As the Γ_P operator, the $\tilde{\Gamma}_P$ operator is also antimonotonic and, therefore, $\tilde{\Gamma}_P^2$ is monotonic having a least and greatest fixpoint respectively denoted by $\mathtt{lfp}(\tilde{\Gamma}_P^2)$ and $\mathtt{gfp}(\tilde{\Gamma}_P^2)$.

Definition 8.7 (CP well-founded model)**.** *Given a labelled program P, its CP well-founded model is given by the pair $\tilde{W} = \langle \texttt{lfp}(\tilde{\Gamma}^2_P), \texttt{gfp}(\tilde{\Gamma}^2_P)\rangle$.* □

For the sake of readability, for every atom A, we will write:

- $\tilde{W}(A) \overset{\text{def}}{=} \texttt{lfp}(\tilde{\Gamma}^2_P)(A)$,
- $\tilde{W}(\mathit{not}\ A) \overset{\text{def}}{=} {\sim}\texttt{gfp}(\tilde{\Gamma}^2_P)(A)$, and
- $\tilde{W}(\texttt{undef}\ A) \overset{\text{def}}{=} {\sim}\tilde{W}(A) * {\sim}\tilde{W}(\mathit{not}\ A)$.

We can see now that, in Programs 8.1 and 4.3, $\tilde{W}(\mathit{fire}) = \texttt{lfp}(\tilde{\Gamma}^2_P)(\mathit{fire})$ are respectively $\mathit{match} \cdot f$ and

$$({\sim}\overline{\mathit{oxygen}} * \mathit{match}) \cdot f \tag{104}$$

The value $\mathit{match} \cdot f$ is also the cause of fire according to our causal semantics and the well-founded model of Program 8.1. On the other hand, fire does not hold in Program 4.3 and, consequently, there is no cause for it. (104) provides a further explanation which points out that fire is false because $\overline{\mathit{oxygen}}$ has prevented the match to cause the fire.

Definition 8.8 (CP-Justification)**.** *A query literal (q-literal) L is either an atom A, its default negation* not A *or* $\texttt{undef}\ A$*. Given a program P and a q-literal L we say that a term with no sums E is a* CP-justification *for L iff $E \leq \tilde{W}(L)$. Every label $l \in Lb$ that occurs negated (${\sim}l$) in E is said to be a* disabling condition *of E and every label that occurs doubly negated (${\sim}{\sim}l$) is said to be an* enabling condition*. A justification is said to be* disabled *if it contains some disabling condition and it is said to be* non-disabled *otherwise.* □

Intuitively, a disabled CP-justification is a cause that, by default would have caused the effect, but it was prevented to do it. For instance, in Program 4.3, the CP-term (104) is a disabled CP-justification indicating that match would have caused a fire but for the absence of oxygen. Clearly, disabled CP-justifications do not correspond to actual causes, but to causes that could have been. Hence, to obtain the causes of an atom from its CP well-founded model we must remove all disabled CP-justifications. Furthermore, as we have informally seen above, for obtaining a cause from a CP-justification, we must remove every enabling condition, that is doubly negated labels. The following mapping formalises this transformation.

Definition 8.9 (CP to causal term mapping). *For a CP-term in negation normal form u we define*

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

For any CP-term we define $\lambda^c(t) \overset{\text{def}}{=} \lambda^c(u)$ where a u is CP-term in negation normal form equivalent to t. Then, λ^c may also be applied to CP-values by applying it to any of the representatives in the equivalence class. We also extend the application of λ^c to interpretations and pairs of interpretations in the following way: $\lambda^c(\tilde{I}) \overset{\text{def}}{=} \tilde{J}$ iff $\tilde{J}(A) = \lambda^c(\tilde{I}(A))$ for any literal A and $\lambda^c(\langle \tilde{I}, \tilde{J} \rangle) \overset{\text{def}}{=} \langle \lambda^c(\tilde{I}), \lambda^c(\tilde{J}) \rangle$ for any CP-interpretations $\tilde{I}$ and $\tilde{J}$. □

Definition 8.9 formalises the above intuitive translation from CP-terms into causal terms and provides a way of obtaining the causal well-founded model of a program from its CP well-founded model. For instance, applying λ^c to the CP-term associated to $fire$ in the CP well-founded model of Program 8.1, it follows that $\lambda^c(match \cdot f) = match \cdot f$ which corresponds to the value assigned in its causal well-founded model. That is, λ^c maps those CP-terms that are also causal terms, like $match \cdot f$, into themselves. Similarly, applying λ^c to the CP-term (104) we will obtain:

$$(0 * match) \cdot f \quad = \quad 0$$

which corresponds the value assigned to $fire$ in the causal well-founded model of Program 4.3. Mapping λ^c removes all disabled CP-justifications and all enabling conditions from the non-disabled ones. The following theorem formalises the relation between the causal well-founded model and the CP well-founded model.

Theorem 8.2 (CP-causal well-founded model correspondence). *Let P be a labelled program, W its causal well-founded model (Definition 4.13) and $\tilde{W}$ is CP well-founded model (Definition 8.7). Then $\lambda^c(\tilde{W}) = W$.* □

Proof. The proof can be found in Appendix A on page 286. □

Theorem 8.2 shows that the CP well-founded model (Definition 8.7) extends the notion of causal well-founded model (Definition 4.13) by allowing terms with negations.

Let us address now the relation between the CP well-founded model and why-not provenance information of a logic program. *Provenance interpretations* are functions $\mathfrak{I} : At \longrightarrow \mathbf{B}_{Lb}$ mapping each atom into a provenance value. Furthermore $t \leq u$ iff $t \wedge u = t$ iff $t \vee u = u$ iff $t \models_{cl} u$ where $\models_{cl}$ stands for classical satisfaction. As we did with CP-interpretations, this relation is extended to provenance interpretations so that $\mathfrak{I} \leq \mathfrak{J}$ iff $\mathfrak{I}(A) \leq \mathfrak{J}(A)$ for any literal A. The value assigned to a negative literal *not* A by an interpretation $\mathfrak{I}$, denoted as $\mathfrak{I}(not\ A)$, is defined as: $\mathfrak{I}(not\ A) \overset{\text{def}}{=} \neg\mathfrak{I}(A)$.

Definition 8.10 (Provenance model). *Given a positive labelled rule R of the form* (48) *(with $m = 0$), we say that a provenance interpretation $\mathfrak{I}$* satisfies *R, in symbols $\mathfrak{I} \models R$, if and only if the following condition holds:*

$$\mathfrak{I}(B_1) \wedge \ldots \wedge \mathfrak{I}(B_m) \wedge l_R \ \leq\ \mathfrak{I}(A) \tag{105}$$

Given a positive program P, we say that a provenance interpretation $\mathfrak{I}$ is a provenance model *of P, in symbols $\mathfrak{I} \models P$, if and only if, I is a $\leq$-minimal provenance interpretation that satisfies all rules in P.* □

The *direct consequences operator* for provenance interpretations is defined in the following way.

Definition 8.11 (Provenance direct consequences). *Given a positive labelled program P over a signature $\sigma = \langle At, Lb, \delta \rangle$, the operator of* direct consequences *is the function $\mathfrak{T}_P : \tilde{\mathbf{I}}_\sigma \longrightarrow \tilde{\mathbf{I}}_\sigma$ such that $\mathfrak{T}_P(\mathfrak{I})(A)$ is given by*

$$\bigvee \{\ \mathfrak{I}(B_1) \wedge \ldots \wedge \mathfrak{I}(B_m) \wedge l_R \ \big|\ R \in P \text{ and } head(R) = A\ \}$$

for any CP-interpretation $\mathfrak{I}$ and any literal $A \in At$. □

Theorem 8.3 (From [Damásio et al., 2013]). *A program P has a least model which can be obtained by iterating the direct consequences operator of Definition 8.11 on the bottom interpretation which maps every atom to $\bot$.* □

Comparing Definitions 8.10 and 8.11 of provenance models and provenance direct consequences operator to the respective Definitions 8.5 and 8.6 of CP-models and CP-direct consequences operator, it is easy to see that, in the former two, products ($*$) and applications ($\cdot$) are replaced by conjunctions ($\wedge$) and that each label associated to the literal in the head of the rule, $\delta(A)$, is missing. Furthermore, sum ($+$) and disjunction ($\vee$) are respectively the least upper bound of the $\leq$ relations. Then, we may define a function $\lambda^p : \tilde{\mathbf{V}}_{Lb} \longrightarrow \mathbf{B}_{Lb}$ that transforms CP-values into their corresponding provenance values.

Definition 8.12 (CP to provenance terms mapping)**.** *Given a term t we define a mapping λ^p in the following recursive way:*

$$\lambda^p(t) \stackrel{\text{def}}{=} \begin{cases} \lambda^p(u) \vee \lambda^p(w) & \textit{if } t = u + v \\ \lambda^p(u) \wedge \lambda^p(w) & \textit{if } t = u \otimes v \textit{ with } \otimes \in \{*, \cdot\} \\ \neg\lambda^p(u) & \textit{if } t = \sim u \\ 1 & \textit{if } t = \delta(A) \textit{ for some } A \in At \\ l & \textit{otherwise} \end{cases}$$

We define a mapping $\lambda^p : \tilde{\mathbf{V}}_{Lb} \longrightarrow \mathbf{B}_{Lb}$ by mapping each CP-value into the equivalence class of the mapping of any of its representatives. □

Theorem 8.4 (CP-provenance least model correspondence)**.** *Let P be positive labelled program, $\tilde{I}$ and $\mathfrak{I}$ be respectively the least CP-model and least provenance model of P. Then $\mathfrak{I} = \lambda^p(\tilde{I})$.* □

Proof. The proof can be found in Appendix A on page 284. □

Theorem 8.4 shows that our CP approach is also an extension of why-not provenance approach under the least model semantics. Note, however, that the CP well-founded model of a program containing the single rule $a : p \leftarrow p$ assigns the value 0 to p. In contrast, the why-not provenance information of p is $\neg not\ p$. Capturing hypothetical why-not provenance has been addressed in [Damásio et al., 2013] by adding new facts to the program in the following way:

Definition 8.13 (Provenance Program)**.** *Given a positive labelled program P, its* provenance program *is $\mathfrak{P}(P) \stackrel{\text{def}}{=} P \cup Q$ where Q contains a labelled fact of the form*

$\sim\text{not } A : \ A$

for each atom $A \in At$ such that there is not a fact A in P. We will write just $\mathfrak{P}$ instead of $\mathfrak{P}(P)$ when the program P is clear by the context. □

The why-not provenance information of a positive program can be obtained by a two step procedure. First, we build the corresponding provenance program and, then, we compute its provenance least model. The following formalises the idea of why-not provenance information.

Definition 8.14 (Why-not provenance (positive programs))**.** *Let P be a positive labelled program and $\mathfrak{I}$ be the least model of $\mathfrak{P}(P)$. Then, the why-not provenance information $Why_P(A)$ for a positive NAF-literal A is given by $Why_P(A) \overset{\text{def}}{=} \mathfrak{I}(A)$ and for a negative NAF-literal* not A *is given by $Why_P(\text{not } A) \overset{\text{def}}{=} \neg\mathfrak{I}(A)$. A* why-not provenance justification *of a literal A is a conjunction D such that $D \leq Why_P(\text{not } A)$.* □

We distinguish two kinds of why-not provenance justifications. Those that do not contain negated conjuncts are called *real*. Those that contain negated conjuncts are called *hypothetical*. Recall also that, $\mathfrak{P}(P)$ is just a labelled program augmented with new facts and, therefore, from Theorem 8.4 the next result immediately follows.

Corollary 8.1. *Let P be positive labelled program, and let $\tilde{I}$ and $\mathfrak{I}$ be the least CP-model and least provenance model of $\mathfrak{P}(P)$, respectively. Then $\mathfrak{I} = \lambda^p(\tilde{I})$ and, therefore, $Why_P(A) = \lambda^p(\tilde{I}(A))$ and $Why_P(\text{not } A) = \lambda^p({\sim}\tilde{I}(A))$.* □

Corollary 8.1 states that an alternative way of obtaining the why-not provenance information of a program is by building its provenance program, computing its CP well-founded model and then applying the mapping λ^p.

The why-not provenance information for a normal program under the well-founded semantics can be obtained, in its turn, by a fixpoint construction of the style of [Van Gelder, 1989]. The reduct of a program P with respect to a provenance interpretation $\mathfrak{I}$ is defined as the CP-reduct (Definition 8.4). Note that, for a provenance interpretation $\mathfrak{I}(\textit{not } C_i) = \neg\mathfrak{I}(C_i)$ instead of $I(\textit{not } C_i) = {\sim}I(C_i)$. The $\mathfrak{G}_P(\mathfrak{I})$ operator returns the least provenance model of program $P^{\mathfrak{I}}$. The provenance well-founded model is given by the pair $\langle \texttt{lfp}(\Gamma_P^2), \texttt{gfp}(\Gamma_P^2)\rangle$ where $\texttt{lfp}(\mathfrak{G}_P^2)$ and $\texttt{lfp}(\mathfrak{G}_P^2)$ are respectively the least and greatest fixpoints of operator $\mathfrak{G}_P^2$ given by $\mathfrak{G}_P^2(\mathfrak{I}) \overset{\text{def}}{=} \mathfrak{G}_P(\mathfrak{G}_P(\mathfrak{I}))$.

Theorem 8.5 (CP-provenance well-founded model correspondence)**.** *Let P be a labelled program, and let $\tilde{W}$ and $\mathfrak{W}$ be the CP and provenance well-founded model of P, respectively. Then $\mathfrak{W} = \lambda^p(\tilde{W})$.* □

Proof. The proof can be found in Appendix A on page 174. □

Theorem 8.5 extends the correspondence stated by Theorem 8.4 for positive programs to normal programs under the well-founded semantics. The why-not provenance information under the well-founded semantics is defined in a similar manner as for positive programs, but information for undefined literals is also defined.

Definition 8.15 (Why-not provenance)**.** *Let P be a labelled program. Then the why-not provenance information $Why_P(A)$ for a positive NAF-literal A is given by $Why_P(A) \overset{\text{def}}{=} \mathtt{lfp}(\mathfrak{G}^2_{\mathfrak{P}})(A)$ and for a negative NAF-literal* not A *is given by $Why_P(\text{not } A) \overset{\text{def}}{=} \neg\mathtt{gfp}(\mathfrak{G}^2_{\mathfrak{P}})(A)$. For an undefined literal A, its why-not provenance information $Why_P(\mathtt{undef}\, A)$ is defined as $Why_P(\mathtt{undef}\, A) \overset{\text{def}}{=} \neg Why_P(A) \wedge \neg Why_P(\text{not } A)$.* □

Corollary 8.2. *Let P be labelled program and $\tilde{W}$ be the CP well-founded model of $\mathfrak{P}(P)$, respectively. Then $Why_P(L) = \lambda^p(\tilde{W}(L))$ for L in $\{A, \text{not } A, \mathtt{undef}\, A\}$ and any atom A.* □

Corollary 8.2 relates why-not provenance information of a literal with its value in the CP well-founded model of its provenance program. Finally, in order to relate the why-not provenance information of a program to its causes we have to relate the CP well-founded model of a program to the CP well-founded model of its provenance program. The provenance program only adds facts of the form $(\sim\! not\ A : A)$, and so, by mapping each term of the form $\sim\! not\ A$ to 0 we will obtain the CP well-founded model of the original program.

Proposition 8.1. *Let P be a logic program, $\tilde{W}$ the CP well-founded model of P and $\tilde{I}$ be the CP well-founded model of $\mathfrak{P}(P)$. Then $\tilde{W}$ is the result of replacing each occurrence of labels of the form* not A *by* 1. □

Proof. The proof can be found in Appendix A on page 283. □

Taking into account the result of Proposition 8.1 we may extend the definition of λ^c to provenance programs in the following way.

Definition 8.16 (Hypothetical CP to causal mapping)**.** *Given a CP-term t in negation normal form we define a mapping λ^{cp} in the following recursive way:*

$$\lambda^{cp}(u) \overset{\text{def}}{=} \begin{cases} \lambda^{cp}(v) \otimes \lambda^{cp}(w) & \text{if } u = v \otimes w \text{ with } \otimes \in \{+, *, \cdot\} \\ 1 & \text{if } u = \sim\sim l \text{ with } l \in Lb \cup Lb^{not} \\ 0 & \text{if } u = \ \ \sim l \text{ with } l \in Lb \cup Lb^{not} \\ l & \text{if } u = \quad l \text{ with } l \in Lb \\ 1 & \text{if } u = \quad l \text{ with } l \in Lb^{not} \end{cases}$$

We define a mapping $\lambda^{cp} : \tilde{\mathbf{V}}_{Lb} \longrightarrow \mathbf{B}_{Lb}$ by mapping each CP-value into the equivalence class of the mapping of any of its representatives in negated normal form. □

Proposition 8.2. *Let P be a logic program, $\tilde{W}$ the CP well-founded model of $\mathfrak{P}(P)$ and W the causal well-founded model of P. Then $W = \lambda^{cp}(\tilde{W})$.* □

Proof. The proof can be found in Appendix A on page 286. □

Taking together the results of Corollary 8.2 and Proposition 8.2 we can build a formal correspondence between the causes of a literal and its real why-not provenance justifications.

Theorem 8.6 (Causal-provenance well-founded correspondence)**.** *Let P be a labelled program, and A be a literal. Then a conjunction of labels D is a real why-not provenance justification of A, that is $D \leq Why_P(A)$, iff there is some causal graph G sufficient for A with respect to the causal well-founded model of P, that is $G \leq \mathtt{lfp}(\Gamma_P^2)(A)$, and D is a conjunction containing all the labels in G excepting those in the image of δ.* □

Proof. The proof can be found in Appendix A on page 286. □

Theorem 8.6 states that, under the well-founded semantics, for every cause, there is a real why-not provenance justification containing all its vertices as conjuncts, and vice-versa.

Why-not provenance justifications have also been defined for the answer set semantics in the following way.

Definition 8.17 (Answer set why-not provenance)**.** *The answer set why-not provenance for A is: $AnsWhy_P(A) \overset{\text{def}}{=} Why_P(A) \ \wedge \ \bigwedge_{a \in At} \neg Why_P(\text{undef}A)$.* □

In contrast to our approach, answer-set why-not provenance information has been defined with respect to a unique "canonical model" of the program instead of with respect to each of the stable models. To illustrate how this difference influences the obtained explanations, consider the following program.

Program 8.4.

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

Ignoring the labels, Program 8.4 has two (standard) answer sets $\{a,c\}$ and $\{b,c\}$, and consequently, it also has two causal answer sets. The fact c is the unique sufficient cause of the literal c with respect to any of the causal stable models. Furthermore, $c \cdot r_1$ is the unique sufficient cause of a with respect to the former and r_2 is the unique sufficient cause of b with respect to the latter. On the other hand the answer-set why-not provenance of such atoms correspond to the prime implicants of the following Boolean formulas:

$$\begin{array}{rcl} AnsWhy_P(a) & = & \neg not(a) \vee not(b) \wedge \neg r_2 \wedge r_1 \\ AnsWhy_P(b) & = & \neg not(b) \vee not(a) \wedge \neg r_1 \wedge r_2 \\ AnsWhy_P(c) & = & c \wedge \neg not(a) \\ & & \vee\ c \wedge \neg not(b) \\ & & \vee\ c \wedge \neg r_1 \\ & & \vee\ c \wedge \neg r_2 \end{array}$$

It is easy to see that there is no real answer-set why-not provenance justification of neither a nor b nor even c, despite the fact that c is true in all the answer sets. In general, there is not any real answer set why-not provenance justification, unless the (standard) well-founded model is two-valued. This fact implies that the above correspondence stated between our approach and why-not provenance justifications under the well-founded semantics cannot be extended to the case of the answer set semantics. The correspondence does not hold even if the program has a unique answer set, but not a two-valued well-founded model. Consider the following two program obtained by adding a constraint r_3 to the Program 8.4.

Program 8.5.

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

Program 8.5 has a unique answer set $\{a, c\}$ and the unique cause of a in it is $c \cdot r_1$. On the other hand

$$\begin{array}{rcl} AnsWhy_P(a) & = & \neg not\ a \wedge \neg r_3 \\ & & \vee \neg not\ a \wedge \neg not\ d \\ & & \vee \neg not\ a \wedge not\ b \\ & & \vee\ not\ b \wedge \neg r_2 \wedge c \wedge r_1 \end{array}$$

Hence, there is no real answer set why-not provenance justification for a. In fact, each of the four why-not provenance justifications — corresponding to the four prime implicants of $AnsWhy_P(a)$ — points out a modification of the program that will lead to a two-valued well-founded model where a holds. This shows that, contrarily to what has happen under the well-founded semantics, the approach followed by Damásio et al. [2013] with respect to answer sets is quite different from our approach, under which every literal has a cause if and only if it holds with respect to that answer set.

8.2 OFF-LINE JUSTIFICATIONS FOR ASP

Off-line justifications have been introduced by Pontelli et al. [2009] as a tool for helping users to understand the program's behaviour and debugging it. In particular, our approach and off-line justifications share two features. First, both approaches are useful for understanding why a given literal holds or not with respect to some answer set. Second, both approaches are graph based. In fact, an off-line justification is a labelled, directed graph whose vertices are literals. For instance, Figure 31 depicts the two off-line justifications of the literal *bomb* in Program 3.1. Comparing these off-line justifications to the causal

$$\begin{tikzcd}
 & bomb^{+} \arrow[d, "+"'] & & bomb^{+} \arrow[d, "+"'] \\
 & open^{+} \arrow[dl, "+"'] \arrow[dr, "+"] & & open^{+} \arrow[dd, "+"'] \\
up(a)^{+} \arrow[d, "+"'] & & up(b)^{+} \arrow[d, "+"'] & \\
lift(a)^{+} \arrow[dr, "+"] & & lift(b)^{+} \arrow[dl, "+"'] & key^{+} \arrow[d, "+"'] \\
 & \top & & \top
\end{tikzcd}$$

Figure 31: Off-line justifications of *bomb* in Program 3.1.

graphs depicted in Figure 3, the first thing to notice is that rule labels have been replaced by the literals in the head of the corresponding rules. Conversely, rule labels can be obtained from the off-line justification by looking at the children of each vertex (under the assumption that the same rule is not repeated with different labels). Furthermore, head and tail of edges are interchanged. Also off-line justifications are usually depicted turned upside down with respect to how causal graphs have been displayed. It is also worth to mention that vertices and edges, in Figure 3, are labelled with a (+) symbol. Vertices labelled with a (+) symbol mean that their corresponding literals hold with respect to the stable model. On the other hand, vertices labelled with the (−) symbol will mean that their corresponding literals do not hold with respect to the stable model. For instance, Figure 32 depicts the off-line justification of *fire* with respect to

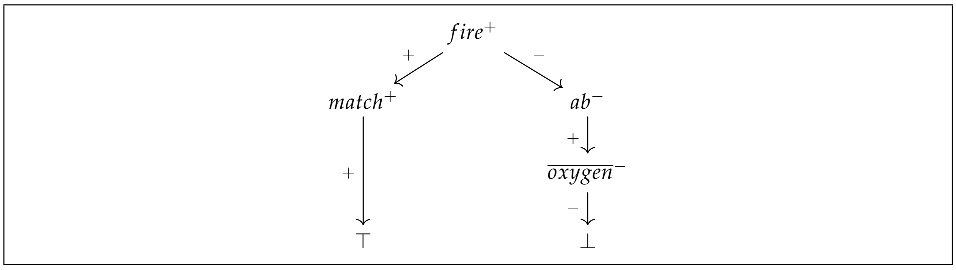

Figure 32: Off-line justifications of *bomb* in Program 3.1.

Program 8.1. In this off-line justification, literals *ab* and $\overline{oxygen}$ are labelled with the (−) symbol. This points out that they do not hold in the unique answer set of Program 8.1. Furthermore, the edge from *fire* to *ab* is also labelled with (−) symbol pointing out that *fire* negatively depends on *ab*. On the other hand, *fire* positively depends on *match*. That is, the rule corresponding to vertex *fire* is of the form $fire \leftarrow match, not\ ab$. Hence, we may check the program, and replace $fire^{+}$ by label f to obtain a graph of rule labels. By contrast, causes only contain literals that hold with respect to the answer set and do not contain negative dependences, therefore every vertex and edge in a causal graph can be considered as (+) labelled in off-line justifications. Off-line justifications have three additional special vertices, $\top$, $\bot$ and *assume*. The $\top$ vertex is used to justify facts, while the $\bot$ vertex is used to justify why literals that have not rules in the program are false. Vertex *assume* is used for literals that have been assumed to be false. For instance, consider the following program obtained by labelling a program taken from Pontelli et al. [2009].

Program 8.6.

$$\begin{array}{llllll}
a: & a \leftarrow f, not\ b & b: & b \leftarrow e, not\ a & e: & e \\
f: & f \leftarrow e & d: & d \leftarrow c, e & c: & c \leftarrow d, f
\end{array}$$

Program 8.6 has two answer sets: $\{a,e,f\}$ and $\{b,e,f\}$. Figure 33 shows the off-line justification for the literal b w.r.t. the answer set $\{b,e,f\}$. Here, a is just assumed to be false. In off-line justifications, atoms may be false because

$$\begin{tikzcd}
 & b^{+} \arrow[dl, "+"'] \arrow[dr, "-"] & \\
e^{+} \arrow[d, "+"'] & & a^{-} \arrow[d, "-"'] \\
\top & & \mathit{assume}
\end{tikzcd}$$

Figure 33: Off-line justifications of b w.r.t. $\{b,e,f\}$ in Program 8.6.

they have been assumed to be false, which is the case of a in this example, or because they are justified to be false as $\overline{oxygen}$ in Figure 32. In our semantics, negative literals are always assumed as being in their default state.

Taking into account these differences, a causal graph corresponding to an off-line justifications can be obtained by the following procedure.

1. Replace each vertex by the corresponding rule label by looking at their children.
2. Remove every vertex and edge labelled with the $(-)$ symbol.
3. Ignore the $(+)$ label for the remaining vertices and edges.
4. Remove the special vertices $\top$, $\bot$ and *assume*.
5. Remove all vertices and edges that are unreachable from the root.
6. Reverse the direction of the remaining edges.

For instance, Figure 34 depicts the only cause of b with respect to $\{b,e,f\}$ in Program 8.6.

$$\begin{tikzcd}
e \arrow[d] \\
b
\end{tikzcd}$$

Figure 34: Cause of b w.r.t. $\{b,e,f\}$ in Program 8.6.

Despite these similarities, there are also three main differences. The first, which we have already seen, is the treatment of negation. In off-line justifications negation propagates negative dependences, whereas in our semantics, negation represents a default behaviour. A default behaviour can be achieved in off-line justifications by removing negative edges as in the above procedure. For the other way around, we have seen in the previous section how to extend our semantics with CP-terms, which extend the causal algebra with a negation operator $(\sim)$, allowing the representation of negative dependences. The second feature that differentiates off-line justifications from our approach, and also from the why-not provenance approach, is that off-line justifications are built externally to the semantics. That is, program interpretations do not carry any justification information, making difficult to study properties of literals referring to this information. The last major difference is related to the principle

$$\begin{tikzcd}
 & & alarm^{+} \arrow[dll, "+"'] \arrow[dr, "+"] & & \\
sw_3^{+} \arrow[ddddr, bend right, "+"'] & & & current(d)^{+} \arrow[d, "+"'] & \\
 & & & current(b)^{+} \arrow[dll, "+"'] \arrow[dr, "+"] & \\
 & sw_4^{+} \arrow[dd, "+"'] & & & current(c)^{+} \arrow[d, "+"'] \\
 & & & & sw_2^{+} \arrow[dlll, bend left, "+"'] \\
 & \top & & &
\end{tikzcd}$$

Figure 35: Off-line justifications of *alarm* in Program 3.2.

that "causes do not include unnecessary events for being sufficient." In this sense, off-line justifications may provide non-relevant information for explaining a literal. For instance, Figure 35 depicts an off-line justification of the literal *alarm* with respect to Program 3.2. It is easy to see, by applying the above translation, that this off-line justification corresponds to the causal graph G_3 in Figure 5. Recall from the discussion of Example 3.1, that sw_4 is not relevant for *alarm*, and consequently, G_3 is not a cause of it.

8.3 ABAS JUSTIFICATIONS

Schulz and Toni [2013, 2014] propose a method for obtaining justifications with respect to an answer set, relying on the relation between the stable model semantics and the *stable extension semantics for Assumption-Based Argumentation* (ABA) [Bondarenko et al., 1997]. Then, literals are justified by means of ABA arguments and attacks between them. Consider, for instance, the following example taken from Schulz and Toni [2014].

Example 8.1 (Dr. Smith)**.** *Program 8.7 below represents the decision support system used by Dr. Smith. It encodes some general world knowledge as well as an ophthalmologist's specialist knowledge about the possible treatments of shortsightedness. It also captures the additional information that Dr. Smith has about his shortsighted patient Peter.* □

Program 8.7.

$$
\begin{array}{lll}
m: & tightOnMoney & \leftarrow student, not\ richParents \\
p: & caresAboutPracticality & \leftarrow likesSports \\
r: & correctiveLens & \leftarrow shortSighted, not\ laserSurgery \\
s: & laserSurgery & \leftarrow shortSighted, not\ tightOnMoney, \\
 & & not\ correctiveLens \\
g: & glasses & \leftarrow correctiveLens, not\ caresAboutPracticality, \\
 & & not\ contactLens \\
c: & contactLens & \leftarrow correctiveLens, not\ afraidToTouchEyes, \\
 & & not\ longSighted, not\ glasses \\
i: & intraocularLens & \leftarrow correctiveLens, not\ glasses, not\ contactLens
\end{array}
$$

$$
\begin{array}{ll}
\$: & shortSighted \\
\$: & afraidToTouchEyes \\
\$: & student \\
\$: & likesSports
\end{array}
$$

We have labelled rules and facts with respect to the original example. This example will be useful for showing, besides the technical differences and similarities, the epistemological differences in the interpretation of negation between our approach and ABAS. For instance, in rule with label g, the literal $not\ caresAboutPracticality$ does not represent an abnormality or exception to the possibility that $correctiveLenss$ causes $glasses$, like the absence of $oxygen$ represents an exception to the fact that scratching a $match$ causes $fires$ in Example 1.7, but is rather a counterargument against giving $glasses$ to a patient that cares about practicality. In this sense, it is important to justify why we have decided not to give $glasses$ to the patient: that is, negative literals. Justifying negative literals, instead of treating them as *defaults*, will be one of the majors differences between our approach and ABAS justifications.

Technically, ABAS justifications are directed labelled graphs which have two kinds of edges, support and attack edges, respectively denoted by dotted and solid lines. Furthermore, vertices and edges are labelled with either the symbol (+) or the symbol (−). Vertices labelled with the symbol (+) correspond to literals that hold in the answer set similarly as in off-line justifications. Vertices labelled with the symbol (−) correspond to literals that do not hold in the answer set. Edges take the label of their origin vertices. Figure 36 shows the ABA justification explaining why *laserSurgey* does not hold in the unique answer set of Program 8.7. Besides labels (+) and (−), each vertex also contains a label of

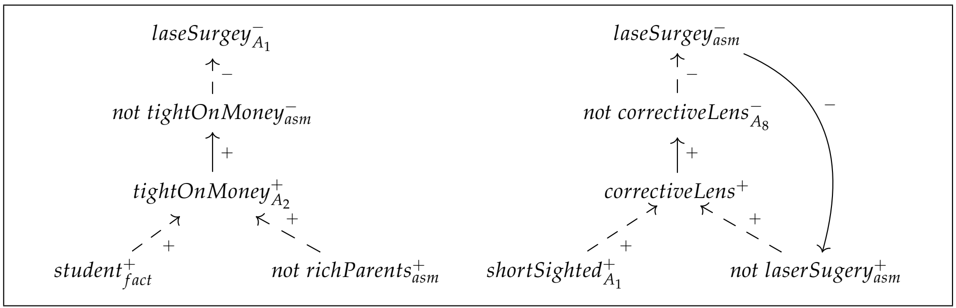

Figure 36: Negative ABA Justification of *laserSurgey* with respect to Program 8.7.

the form A_i, *asm* or *fact*. A label *fact* means that such vertex is a fact. Similarly, a label *asm* indicates that it is an assumption. A label of the form A_i indicates that such vertex is justified by argument A_i. We will discuss arguments below.

The first difference to notice with respect to our approach is that Figure 8.7 represents a justification explaining why the stable model does not satisfies the atom *laserSurgey*. On the other hand, Figure 37 depicts the ABA justification explaining why the stable model satisfy *intraocularLens*. This justification includes the negative literals *not laserSurgey*, *not glasses*, *not contactLens*... Justifications for literals that do not hold in a stable model and explanations containing negative literals can be incorporated in our semantics by means of a negation (∼) operator as we have seen in Section 8.1.

The justification depicted in Figure 37 is also useful to illustrate another important difference between ABAS and our approach. ABAS justifications only contain the facts and the negated literals necessary to derive the literal in question, but not the intermediate rules. For instance, if we remove all negative dependences from justification in Figure 37, in a similar manner as we did with off-line justifications, we obtain the graph depicted in Figure 38. The unique cause of *intraocularLens* with respect to Program 8.7 is depicted in Figure 39.

$$\begin{tikzcd}
 & & \mathit{intraocularLens}^{+}_{A_3} & \\
\mathit{not\ laserSurgey}^{+}_{asm} \arrow[urr, dashed, "-"] & \mathit{shortSighted}^{+}_{fact} \arrow[ur, dashed, "+"'] & \mathit{not\ glasses}^{-}_{asm} \arrow[u, dashed, "-"] & \mathit{not\ contactLens}^{+}_{A_4} \arrow[ul, dashed, "+"'] \\
\mathit{not\ tightOnMoney}^{-}_{A_2} \arrow[u, dashed, "-"] & & \mathit{glasses}^{-}_{A_6} \arrow[u, "-"'] & \mathit{contactLens}^{-}_{A_4} \arrow[u, dashed, "-"'] \\
\mathit{tightOnMoney}^{+}_{A_2} \arrow[u, "+"] & & \mathit{not\ caresAboutPracticality}^{-}_{asm} \arrow[u, "-"'] & \mathit{not\ afaidToTouchEyes}^{-}_{asm} \arrow[u, dashed, "-"'] \\
\mathit{student}^{+}_{fact} \arrow[u, dashed, "+"] & \mathit{not\ richParents}^{+}_{asm} \arrow[ul, dashed, "+"'] & \mathit{caresAboutPracticality}^{+} \arrow[u, "+"'] & \mathit{afaidToTouchEyes}^{+}_{fact} \arrow[u, dashed, "+"'] \\
 & & \mathit{likesSports}^{+}_{fact} \arrow[u, dashed, "+"'] &
\end{tikzcd}$$

Figure 37: ABA Justification of *intraocularLens* with respect to Program 8.7.

$$\begin{tikzcd}
\mathit{intraocularLens}^{+} \\
\mathit{shortSighted}^{+} \arrow[u, dashed, "+"']
\end{tikzcd}$$

Figure 38: Positive dependences of *intraocularLens* with respect to Program 8.7.

$$\begin{tikzcd}
\mathit{shortSighted} \arrow[d] \\
r \arrow[d] \\
i
\end{tikzcd}$$

Figure 39: Sufficient cause *intraocularLens* with respect to Program 8.7.

As pointed out by Schulz and Toni [2014], these rules can be extracted from the argument of *intraocularLens*. An *ABA Argument* is a finite tree where every node satisfies a literal such that

- the leaves contain assumptions or facts,
- if a non-leaf contains A, then there is a rule $A \leftarrow B_1, \ldots, B_m$ and it has m children, containing $B_1, \ldots, B_m$, respectively.

For instance, Figure 40 shows the ABA argument A_3 which has the atom *intraocularLens* as its conclusion. Leaves with the *asm* label, *not laserSurgey*,

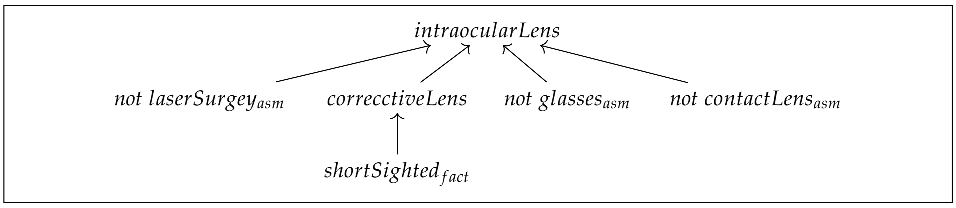

Figure 40: ABA Argument for *intraocularLens* with respect to Program 8.7.

not glasses and *not contanctLens* are assumptions, while *shortSighted* is a fact. If we now remove all the negative dependences, it is easy to see that we obtain a graph that corresponds to the cause depicted in Figure 39 when rule labels are replaced by their heads. In this sense, our concept of cause is technically closer

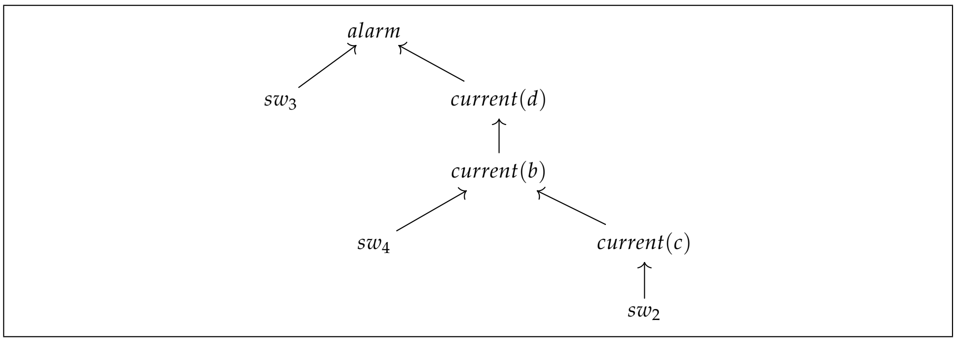

Figure 41: ABA argument A of *alarm* in Program 3.2.

to the concept of ABA argument than to the concept of ABA justification. In a similar manner as has happens with off-line justifications, ABA arguments are not required to be minimal or non-redundant. For instance, Figure 41 depicts an ABA argument of *alarm* in Program 3.2. It is easy to see that, by reversing the edges, labelling vertices and edges with the symbol $(+)$ and connecting the facts sw_3, sw_4 and sw_2 to the $\top$ vertex, we obtain the off-line justification depicted depicted in Figure 35. Recall that the latter corresponds to the redundant causal graph G_3 in Figure 5. Similarly, ABA justifications are not required to be minimal either. Figure 42 depicts the corresponding ABA justification, which also includes switch 4.

$$\begin{tikzcd}
& alarm^{+}_{A} & \\
sw^{+}_{3\ fact} \arrow[ur, "+"] & sw^{+}_{4\ fact} \arrow[u, "+"] & sw^{+}_{2\ fact} \arrow[ul, "+"']
\end{tikzcd}$$

Figure 42: ABA justification of *alarm* in Program 3.2.

8.4 ACTUAL CAUSATION

In this section, we will analyse some traditional examples of the actual causation literature, and relate the results obtained by our approach to the results obtained by some of the established approaches in the field. We have briefly discussed the literature of *actual causation* in the introduction, so we directly introduce the language of structural equations as a first step to provide the formal the definition of *actual causation* given by Halpern and Pearl [2001].

A signature σ is a tuple $\langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$, where $\mathcal{U}$ is a finite set of exogenous variables, $\mathcal{V}$ is a finite set of endogenous variables, and $\mathcal{R}$ associates with every variable $Y \in \mathcal{U} \cup \mathcal{V}$ a non-empty set $\mathcal{R}(Y)$ of possible values for Y (that is, the set of values over which Y ranges). A structural model over signature σ is a tuple $M = \langle \sigma, \mathcal{F} \rangle$, where $\mathcal{F}$ associates to each endogenous variable $X \in \mathcal{V}$ a function denoted F_X such that:

$$F_X : \Big(\bigtimes_{U \in \mathcal{U}} \mathcal{R}(U) \Big) \times \Big(\bigtimes_{Y \in \mathcal{V} \setminus \{X\}} \mathcal{R}(Y) \Big) \longrightarrow \mathcal{R}(X)$$

A function F_X tells us the value of X given the values of all the other variables in $\mathcal{U} \cup \mathcal{V}$. From now on, we will assume that models do not have exogenous variables, that is $\mathcal{U} = \emptyset$, and that they are *binary*, that is $\mathcal{R}(X) = \{0,1\}$ for all $X \in \mathcal{V}$.

For instance, Example 1.5 can be captured by a model with the following set of endogenous binary variables:

$$\mathcal{V} = \{shattered, hit(billy), hit(suzy), throw(suzy), throw(billy)\}$$

and a function F_X for each $X \in \mathcal{V}$ that captures the assignments shown in the equation to the Structural Model 1.1. It is easy to see that the unique solution to these equations must satisfy $shattered = 1$.

Given a signature $\sigma = \langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$, a formula of the form $(X = x)$ for $X \in \mathcal{V}$ and $x \in \mathcal{R}(X)$, is called a *primitive event*. A *basic causal formula* (over a signature σ) has the form $[Y_1 \leftarrow y_1, \dots, Y_k \leftarrow y_k]\varphi$ where φ is a Boolean combination of primitive events, $Y_1, \dots, Y_k$ are distinct variables in $\mathcal{V}$, and $y_i \in \mathcal{R}(Y_i)$. Furthermore, given a set of variables $\vec{Y} = \{Y_1, \dots, Y_k\} \subseteq \mathcal{U} \cup \mathcal{V}$ we write $\vec{y} \in \mathcal{R}(\vec{Y})$ for an assignment of variables $[Y_1 \leftarrow y_1, \dots, Y_k \leftarrow y_k]$ such that $y_i \in \mathcal{R}(Y_i)$ for $1 \le i \le k$. Such a formula is abbreviated as $[\vec{Y} \leftarrow \vec{y}]$. The special case where $k = 0$ is abbreviated as φ. As we have assumed that $\mathcal{R}(X) = \{0,1\}$, we will write just X instead of $(X = 1)$ and $\overline{X}$ instead of $(X = 0)$. Similarly, we write $[y'_1, \dots, y'_k]\varphi$ instead of $[Y_1 \leftarrow y_1, \dots, Y_k \leftarrow y_k]\varphi$ where $y'_i = Y_i$ if $y_1 = 1$ and $y_i = \overline{Y_i}$ otherwise. Intuitively, $[y'_1, \dots, y'_k]\varphi$ says that φ holds in the counterfactual world that would arise if Y_i is set to the value indicated by y'_i for all $1 \le i \le k$. A causal formula is a Boolean combination of basic causal formulas. Furthermore, structural models are restricted to be *recursive* (or acyclic) equations; these are ones that can be described with a directed acyclic graph.

Given some *context* $\vec{u}$ or setting for the variables in U, we write $\langle M, \vec{u} \rangle \models \varphi$ iff φ is true in a causal model M given context $\vec{u}$. $\langle M, \vec{u} \rangle \models [\vec{y}]x$ if and only if $\langle M^{\vec{y}}, \vec{u} \rangle \models x$ where $M^{\vec{y}} = \langle \sigma', \mathcal{F}^{\vec{y}} \rangle$, and σ' is the result of removing from σ the all variables Y in the assignment $\vec{y}$ for all Y in $\vec{Y}$. Similarly, $\mathcal{F}^{\vec{y}}$ is the result of removing the function F_{Y_i} and replacing Y_i by y_i in the remaining equations with $\vec{y} = [Y_1 \leftarrow y_1, \dots, Y_k \leftarrow y_k]$. $\langle M, \vec{u} \rangle \models [\vec{y}]\varphi$ for an arbitrary Boolean combination of primitive events φ is defined similarly. Since we have assumed $\mathcal{U}$ to be empty, the context $\vec{u}$ is irrelevant, so we will just write $M \models [\vec{y}]\vec{x}$ instead of $\langle M, \vec{u} \rangle \models [\vec{y}]\vec{x}$.

We may now make use of this notation for presenting the definition of *actual cause* given by Halpern and Pearl [2001].

Definition 8.18 (HP01). *An assignment $\vec{x}$ for a set of variables $\vec{X}$ is an* actual cause *of formula φ iff the following three conditions hold:*

1. *$M \models \vec{x} \wedge \varphi$. That is, both $\vec{X}$ and φ are true in the actual world.*

2. *There exists a partition $\langle \vec{Z}, \vec{W} \rangle$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $\langle \vec{x}', \vec{w}' \rangle$ of the variables in $\langle \vec{X}, \vec{W} \rangle$, then*

   a) *$M \models [\vec{x}', \vec{w}'] \neg \varphi$. In words, changing $\langle \vec{X}, \vec{W} \rangle$ for the values corresponding to $\langle \vec{x}, \vec{w} \rangle$ changes φ from true to false,*

   b) *$M \models [\vec{x}, \vec{w}', \vec{z}^*] \varphi$ for all subsets $\vec{Z}' \subseteq \vec{Z}$ where $\vec{z}^*$ is the assignment given by M to variables in Z', that is $M \models z^*$. In words, setting $\vec{W}$ to $\vec{w}$ should have no effect on φ as long as $\vec{X}$ is kept at its current value $\vec{x}$, even if all the variables in an arbitrary subset of $\vec{Z}'$ are set to their original values.*

3. *$\vec{X}$ is minimal, no subset of $\vec{X}$ satisfies the other conditions.* □

Eiter and Lukasiewicz [2002] have shown that the following is an equivalent definition of *actual cause* when the structural model is binary:

Theorem 8.7 (From Eiter and Lukasiewicz [2002]). *Let $\sigma = \langle \mathcal{U}, \mathcal{V}, \mathcal{R} \rangle$ and $M = \langle \sigma, \mathcal{F} \rangle$ respectively be an binary signature and a model over such signature. Let $\vec{X} \subseteq \mathcal{V}$ and $\vec{x} \in \mathcal{R}(\vec{X})$, and let φ be an event. Then, $\vec{x}$ is a an actual cause of φ under a context $\vec{u}$ iff following conditions:*

1. *condition 1 in HP01,*

2. *Some set of endogenous variables $\vec{W} \in \mathcal{V} \backslash \vec{X}$ and some setting $\langle \vec{x}', \vec{w}' \rangle$ of variables $\langle \vec{X}, \vec{W} \rangle$ exist such that:*

   a) *$M \models [\vec{x}', \vec{w}'] \neg \varphi$,*

   b) *$M \models [\vec{x}, \vec{w}'] \varphi$,*

   c) *$M \models [\vec{x}, \vec{w}'] \vec{z}^*$ where $\vec{z}^*$ is the assignment given by M to the variables in $Z = \mathcal{V} \backslash (\vec{X} \cup \vec{W})$, that is $M \models \vec{z}^*$;*

3. *$\vec{X}$ is minimal, no subset of $\vec{X}$ satisfies the other conditions.* □

The definition provided by Theorem 8.7 is easier to apply than HP01 because we do not need to check condition 2b for all subsets $\vec{Z}' \subseteq \vec{Z}$. We may then apply this definition for showing that, as we have claimed in the introduction, the event *throw*$(suzy)$ is an actual cause of *shattered* with respect to Example 1.5. It is clear that *throw*$(suzy)$ and *shattered* hold with respect to the structural model represented by the Structural Model 1.1, that is condition 1 of HP01 holds. This conditions consist in checking that:

$$M \models throw(suzy) \wedge shattered$$

For checking the second condition, take the witness $\langle \vec{W}, \vec{x}', \vec{w}' \rangle$ given by:

$$\begin{aligned}
\vec{W} &= \{\ \mathit{throw}(\mathit{billy}), \mathit{hit}(\mathit{billy})\ \} \\
\vec{x}' &= \{\ \overline{\mathit{throw}}(\mathit{suzy})\ \} \\
\vec{w}' &= \{\ \overline{\mathit{throw}}(\mathit{billy}), \overline{\mathit{hit}}(\mathit{billy}))\ \}
\end{aligned}$$

The Structural Model 8.8 below shows the equations of the structural model $M^{\vec{y}}$ with $\vec{y} = \vec{x}' \cup \vec{w}'$.

Structural Model 8.8.

$$\begin{aligned}
\mathit{shattered} &= \mathit{hit}(\mathit{suzy}) \quad \vee \ 0 \\
\mathit{hit}(\mathit{billy}) &= 0 \qquad\qquad \wedge \neg \mathit{hit}(\mathit{suzy}) \\
\mathit{hit}(\mathit{suzy}) &= \mathit{throw}(\mathit{suzy}) \\
\mathit{throw}(\mathit{suzy}) &= 0 \\
\mathit{throw}(\mathit{billy}) &= 1
\end{aligned}$$

It is easy to see that *shattered* does not hold in the Structural Model 8.8. Hence condition 2a of Theorem 8.7, consisting on $M \models [\vec{x}', \vec{w}'] \neg \mathit{shattered}$, holds. Similarly, the Structural Model 8.9 below shows the model $M^{\vec{y}}$ where $\vec{y} = \vec{x} \cup \vec{w}'$.

Structural Model 8.9.

$$\begin{aligned}
\mathit{shattered} &= \mathit{hit}(\mathit{suzy}) \quad \vee \ 0 \\
\mathit{hit}(\mathit{billy}) &= 0 \qquad\qquad \wedge \neg \mathit{hit}(\mathit{suzy}) \\
\mathit{hit}(\mathit{suzy}) &= \mathit{throw}(\mathit{suzy}) \\
\mathit{throw}(\mathit{suzy}) &= 1 \\
\mathit{throw}(\mathit{billy}) &= 1
\end{aligned}$$

It is also clear now that $M \models [\vec{x}, \vec{w}'] \mathit{shattered}$ and $M \models [\vec{x}, \vec{w}'] \mathit{hit}(\mathit{suzy})$. Note that $Z = \{\mathit{hit}(\mathit{suzy}), \mathit{shattered}\}$, and then conditions 2b and 2c of Theorem 8.7 also hold. Furthermore, since $\vec{X}$ is a primitive event, no subset of it satisfies these conditions, therefore we may conclude that $\mathit{throw}(\mathit{suzy})$ is an actual cause of *shattered* with respect to HP01. It can also be shown that such witness cannot be found for $\mathit{throw}(\mathit{billy})$. Note that any witness holding condition 2a must assign $\mathit{hit}(\mathit{suzy}) = 0$ and therefore any witness holding condition 2b must satisfy $\{\mathit{hit}(\mathit{billy})\} \subseteq Z$ and $\mathit{hit}(\mathit{billy}) = 1$ which is not the value in the actual model.

We may represent this example by the following labelled program:

Program 8.10.

$$
\begin{aligned}
\$:\quad & shattered(T+1) \leftarrow throw(X,T),\ not\ shattered(T) \\
& shattered(T+1) \leftarrow shattered(T) \\
\$:\quad & throw(suzy,2) \\
\$:\quad & throw(billy,4)
\end{aligned}
$$

In the least model I of Program 8.10, $I(shattered) = G_{suzy}$ where G_{suzy} is the causal graph depicted in Figure 8.10. It is easy to see that $throw(suzy)$ is a

$$\begin{tikzcd}
throw(suzy,2) \arrow[d] & & throw(john,0) \arrow[d] \\
shattered & & shattered \\
G_{suzy} & & G_{john}
\end{tikzcd}$$

Figure 43: Sufficient cause G_{suzy} of *shattered* with respect to Program 8.10 and G_{john} obtained after adding the labelled fact $\$: throw(john,0)$.

vertex of the unique sufficient cause of *shattered*, and so an actual cause of it, whereas $throw(billy)$ is not. Is is also easy to see that by adding the labelled fact:

$$\$:\quad throw(john,0)$$

to Program 8.10 we will obtain $I(shattered) = G_{john}$ instead, which points out that, in such case, $throw(john)$ is the actual cause of *shattered* and not $throw(suzy)$.

A second example that we have presented in the introduction was the desert traveller scenario of Example 1.8 that has put in trouble Mackie's approach. Program 8.11 bellow labels the rules of Program 1.2 presented there.

Program 8.11.

$$
\begin{aligned}
\$:\quad & death \leftarrow \overline{shoot}, poison \\
\$:\quad & death \leftarrow shoot \\
\$:\quad & shoot \\
\$:\quad & poison
\end{aligned}
$$

Recall that, in Mackie's approach, *death* was explained by the Boolean formula $shoot \vee poison$ that does not allow distinguishing which of the actions is the actual cause of the *death*. We have claimed in the introduction that *death* will be explained by the term $shoot \cdot death$ pointing out that was *shoot* and not *poison* the responsible of the traveller's death. Now it is easy to see that the least model I of Program 8.11 verifies $I(\overline{shoot}) = 0$ because there exists no rule with $\overline{shoot}$ in the head, and therefore $I(death) = shoot \cdot death$. On the other hand, the direct translation of Program 1.2 into the language of structural models would be:

Structural Model 8.12.

$$\begin{aligned} death &= shoot \vee \neg shoot \wedge poison \\ shoot &= 1 \\ poison &= 1 \end{aligned}$$

It is easy to see that according to Halpern and Pearl definition of actual cause (HP01) both, *shoot* and *poison*, are actual causes. We just show that $\langle \vec{W}, x', w' \rangle$ given by:

$$\begin{aligned} \vec{W} &= \{\ shoot\ \} \\ \vec{x}' &= \{\ \overline{poison}\ \} \\ \vec{w}' &= \{\ \overline{shoot}\ \} \end{aligned}$$

is a witness of *poison* being an actual cause of *death*. It is clear that this model holds $M \models poison \wedge death$. To show condition 1a ($M \models [\vec{x}', \vec{w}'] \neg death$) we build the corresponding model $M^{\vec{y}}$ with $\vec{y} = \vec{x}' \cup \vec{w}'$ in the following way:

Structural Model 8.13.

$$\begin{aligned} death &= 0\ \vee\ \neg 0 \wedge 0 = 0 \\ shoot &= 1 \\ poison &= 1 \end{aligned}$$

It is easy to see that *death* do not hold in this program. Similarly, we show condition 2a ($M \models [\vec{x}, \vec{w}'] death$) by building the following model in which *dead* holds.

Structural Model 8.14.

$$\begin{aligned} death &= 0\ \vee\ \neg 0 \wedge 1 = 1 \\ shoot &= 1 \\ poison &= 1 \end{aligned}$$

Hence, as in Mackie's approach, *poison* is wrongly pointed out to be an actual cause of *death*. It is easy to see that, by applying the same procedure, *shoot* is correctly recognised to be an actual cause of *death*.

Pearl [2000] circumnavigate this issue by introducing two new intermediate variables, *dehydration* and *intake*, in the following way:

Structural Model 8.15.

$$\begin{aligned} death &= dehydration \vee intake \\ dehydration &= shoot \\ intake &= \neg shoot \wedge poison \\ shoot &= 1 \\ poison &= 1 \end{aligned}$$

In this case *poison* is correctly not considered to be an actual cause of *death*. Note that in order to fulfil condition 2a *shoot* or *dehydration* must be fixed to false, and both cases have the consequence of *dehydration* being false. If only *dehydrating* is fixed to false, then *shoot* will still be true, and therefore *intake* would not hold. Hence *death* does not hold either. If, on the contrary, *shoot* if fixed to false, then *intake* holds, but in the actual model it has the value 0, so that, condition 2c does not hold. This example reflect the importance of choosing the "right model," a fact that has pointed out by Halpern and Hitchcock [2011]:

> *"Once the modeler has selected a set of variables to include in the model, the world determines which equations among those variables correctly represent the effects of* interventions. *By contrast, the choice of variables is subjective; in general, there need be no objectively 'right' set of exogenous and endogenous variables to use in modeling a problem."*

By contrast, we may include variables *dehydration* and *intake* in Program 8.11 as follows:

Program 8.16.

$$\begin{aligned} \$: \quad & death && \leftarrow dehydration \\ \$: \quad & death && \leftarrow intake \\ \$: \quad & dehydration && \leftarrow shoot \\ \$: \quad & intake && \leftarrow \overline{shoot}, poison \\ \$: \quad & shoot && \\ \$: \quad & poison && \end{aligned}$$

and we still conclude that *shoot* is the actual cause of *death*, whereas *poison* is not. That is, logic programs are not sensitive to the intermediate variables that are included or not. This behaviour coincides with the idea of *locality*. Hall [2004, 2007] says that a causal relation is *local* if causes are connected to their effects via spatiotemporally continuous sequences of causal intermediates. In this sense *dehydration* is a causal intermediate between *shoot* and *death*. Similarly, *intake* is a causal intermediate between *poison* and *death*. As this example shows, our approach does not depend on the inclusion or not of these intermediate variables. In contrast, the definition given by Halpern and Pearl [2001] does.

It is also interesting to analyse the following example introduced by Hopkins and Pearl [2003] which is also related to this same issue:

Example 8.2 (Three men Firing Squad)**.** *For a firing squad consisting of shooters Suzy and Billy, it is John's job to load Suzy's gun. Billy loads and fires his own gun. On a given day, John loads Suzy's gun. When the time comes, Suzy and Billy shoot the prisoner.* □

Hopkins and Pearl [2003] represent this example as follows:

Structural Model 8.17.

$$\begin{aligned} death &= shoot(suzy) \wedge load(john) \ \vee \ shoot\_load(billy) \\ shoot(suzy) &= 1 \\ load(john) &= 1 \\ shoot\_load(billy) &= 1 \end{aligned}$$

and we will obtain that $shoot(suzy)$, $load(john)$ and $shoot\_load(billy)$ are the actual causes of *dead*. Similarly, we may represent this scenario by the following labelled program:

Program 8.18.

$$\begin{aligned} \$: &\quad death \leftarrow shoot(suzy), load(john) \\ \$: &\quad death \leftarrow shoot\_load(billy) \\ \$: &\quad shoot(suzy) \\ \$: &\quad load(john) \\ \$: &\quad shoot\_load(billy) \end{aligned}$$

and obtain the same results. Note that, in this example there are two sufficient causes of *death* depicted in Figure 44.

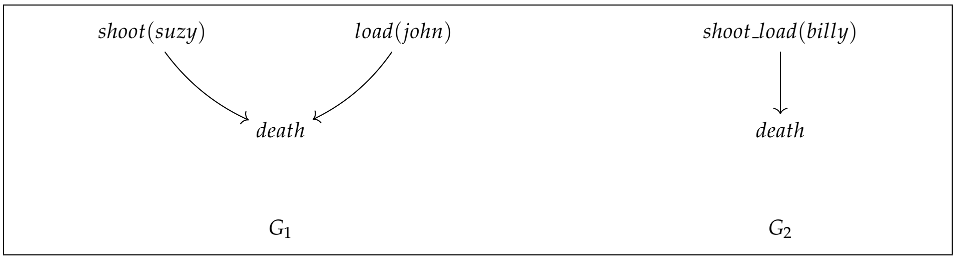

Figure 44: Graphs G_1 and G_2 associated to *dead* in Program 8.18.

We may define a *contributory cause* by incorporating the concept of INUS condition given by Mackie [1965].

Definition 8.19 (Contributory Cause)**.** *Given a labelled program P and a sufficient cause G of some literal A, the head B of any rule R whose label l_R is in G is a* contributory cause *of A.*

It is easy to see that $shoot(suzy)$, $load(john)$ and $shoot\_load(billy)$ are the contributory causes of *death*. That is, the contributory causes and the actual causes of *death* coincide in this example. The interesting feature of this example comes with the following variation introduced also by Hopkins and Pearl [2003]:

Example 8.3 (Ex. 8.2 continued)**.** *Consider now that John still loads Suzy's gun, but Suzy decides not to shoot. Billy still loads and shoots his gun, and the prisoner still dies.* □

This story can be modelled in the same way as above in Program 8.17 by changing the equation $shoot(suzy) = 1$ by $shoot(suzy) = 0$. As Hopkins and Pearl [2003] pointed out, $load(john)$ is surprisingly an actual cause of *death* according to HP01. Just consider a witness $\langle \vec{W}, x', w' \rangle$ given by:

$$\begin{aligned}
\vec{W} &= \{\ shoot(suzy), shoot(billy)\ \} \\
\vec{x}' &= \{\ \overline{load(john)}\ \} \\
\vec{w}' &= \{\ shoot(suzy), \overline{shoot}(billy)\ \}
\end{aligned}$$

By contrast, the least model I of the program obtained by removing the fact $shoot(suzy)$ from Program 8.18 satisfies that:

$$I(death) = shoot\_load(billy) \cdot death$$

That is, our semantics recognises, as expected, that Billy is an actual cause of the prisoner's death while neither Suzy nor John are. In order to deal with this example Halpern and Pearl [2005] propose the following revision of HP01.

Definition 8.20 (HP05). *An assignment $\vec{x}$ for a set of variables $\vec{X}$ is an actual causal formula φ iff the following three conditions hold:*

1. *$M \models \vec{x} \wedge \varphi$. That is, both $\vec{X}$ and φ are true in the actual world.*
2. *There exists a partition $\langle \vec{Z}, \vec{W} \rangle$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $\langle \vec{x}', \vec{w}' \rangle$ of the variables in $\langle \vec{X}, \vec{W} \rangle$, then*
   a) *$M \models [\vec{x}', \vec{w}'']\neg\varphi$. In words, changing $\langle \vec{X}, \vec{W} \rangle$ for the values corresponding to $\langle \vec{x}, \vec{w} \rangle$ changes φ from true to false,*
   b) *$M \models [\vec{x}, \vec{w}'', \vec{z}^*]\varphi$ for all subsets $\vec{W}' \subseteq \vec{W}$ and $\vec{Z}' \subseteq \vec{Z}$ where $\vec{w}''$ is a assignment to variables in $\vec{W}'$ that agrees with $\vec{w}'$ in all those variables and $\vec{z}^*$ is the assignment give by M to variables in Z', that is $M \models z^*$. In words, setting $\vec{W}$ to $\vec{w}$ should have no effect on φ as long as $\vec{X}$ is kept at its current value $\vec{x}$, even if all the variables in an arbitrary subset of $\vec{Z}'$ are set to their original values.*
3. *$\vec{X}$ is minimal, no subset of $\vec{X}$ satisfies the other conditions.* □

Definitions 8.18 and 8.20 are equal but for condition 2b, which additionally impose that φ must hold even if all subsets of $\vec{W}$ are not fixed to their values in $\vec{w}'$. According to HP05, we obtain the expected result that event $load(john)$ is not an actual cause of $death$: now (8.4) is not a witness of $load(john)$ being an actual cause of $death$ because, in addition to the above criteria, $death$ would have to hold when $shoot(suzy)$ is fixed to its original value 0, which is not the case.

Alternatively, we may solve this issue by adding an intermediate variable as Pearl [2000] did by introducing the *dehydration* and *intake* variables in the desert traveller scenario of Example 1.8. Hence, we will have the following model:

Structural Model 8.19.

$$\begin{aligned}
death &= bullet \lor shoot\_load(billy)\\
bullet &= shoot(suzy) \land load(john)\\
shoot(suzy) &= 1\\
load(john) &= 1\\
shoot\_load(billy) &= 1
\end{aligned}$$

In Program 8.19, the above witness for $load(john)$ being an actual cause of $death$ is not valid any more because $bullet$ has to take the value 1 to fulfil condition 2b of Theorem 8.7 but its actual value is 0. That is, this kind of examples may be correctly captured by HP01 by ensuring that the right hand side of each equation contains either only disjunctions or only conjunctions. On the other hand, Example 1.8 cannot be solved with this new definition without the intermediate variables. In the model shown by Structural Model 8.12, once the value of $poison$ is fixed to 1, the value of $shoot$ does not matter. Consequently, given that we may solve the above issue by providing an *appropriate model* in which conjunctions and disjunctions are not mixed in the same equation, and that HP05 does not solve the issue in all cases, it seems reasonable to remain with HP01. To the best of our knowledge, no further evidence in favour of HP05 over binary models has been provided. Therefore, in the following we continue the comparative with respect to HP01 and avoid HP05.

Till this point, we have considered scenarios in which default knowledge is absent, so that we may represent them by means of positive programs. For introducing default knowledge, consider now the following example due to Hiddleston [2005] which has also been discussed by Halpern [2008].

Example 8.4 (Poison). *Assassin is in possession of a lethal poison, but has a last-minute change of heart and refrains from putting it in Victim's coffee. Bodyguard puts antidote in the coffee, which would have neutralized the poison had there been any. Victim drinks the coffee and survives.* □

As Halpern [2008] points out, the interesting feature of this example consists in determining whether:

> *"Is Bodyguard's putting in the antidote a cause of Victim surviving? Most people would say no, but according to the preliminary HP definition, it is. For in the contingency where Assassin puts in the poison, Victim survives iff Bodyguard puts in the antidote."*

This example is represented in Halpern [2008] by the following set of structural equations:

Structural Model 8.20.

$$\begin{aligned} survive &= antidote \vee \neg poison \\ antidote &= 1 \\ poison &= 0 \end{aligned}$$

As commented above, it is easy to see that, by fixing a contingency where $poison = 1$, survive becomes counterfactually dependent on *antidote*, so that *antidote* is an actual cause of *survive*.

In order to deal with *default knowledge*, Halpern [2008] introduces *extended causal models* to be a tuple $M = \langle \sigma, \mathcal{F}, \kappa \rangle$, where $\langle \sigma, \mathcal{F} \rangle$ is a causal model, and κ is a ranking function that associates a rank to each world. A *world* is a complete description of the values of all the random variables. Each world has an associated rank, which is just a natural number or ∞. Intuitively, the higher the rank, the less likely the world. A world with rank 0 is reasonably likely, one with a rank of ∞ is somewhat unlikely, one with a rank of 2 is quite unlikely, and so on.

Definition 8.21 (HP01 defaults)**.** *An assignment $\vec{x}$ for a set of variables $\vec{X}$ is an actual causal formula φ iff the following three conditions hold:*

1. *condition 1 in HP01,*
2. *There exists a partition $\langle \vec{Z}, \vec{W} \rangle$ of $\mathcal{V}$ with $\vec{X} \subseteq \vec{Z}$ and some setting $\langle \vec{x}', \vec{w}' \rangle$ of the variables in $\langle \vec{X}, \vec{W} \rangle$, then*
   a) *$M \models [\vec{x}', \vec{w}'] \neg \varphi$ and $\kappa(s) \leq \kappa(s^*)$ where s is the world obtained by applying the assignment $\langle \vec{x}', \vec{w}' \rangle$ and s^* is the actual world. In words, changing $\langle \vec{X}, \vec{W} \rangle$ for the values corresponding to $\langle \vec{x}, \vec{w} \rangle$ changes φ from true to false in a more normal world,*
   b) *condition 2b in HP01,*
3. *$\vec{X}$ is minimal, no subset of $\vec{X}$ satisfies the other conditions.* □

Hence, to deal with Example 8.4 we have to define a ranking over the possible worlds for the variables $\{survive, antidote, poison\}$. We may define the raking of worlds in this example to be a function κ mapping each world in the following way:

$$\begin{array}{lcl}
\{survive = 0, antidote = 0, poison = 0\} & \mapsto & 0 \\
\{survive = 1, antidote = 0, poison = 0\} & \mapsto & 0 \\
\{survive = 0, antidote = 1, poison = 0\} & \mapsto & 1 \\
\{survive = 1, antidote = 1, poison = 0\} & \mapsto & 1 \\
\{survive = 0, antidote = 0, poison = 1\} & \mapsto & 1 \\
\{survive = 1, antidote = 0, poison = 1\} & \mapsto & 1 \\
\{survive = 0, antidote = 1, poison = 1\} & \mapsto & 2 \\
\{survive = 1, antidote = 1, poison = 1\} & \mapsto & 2
\end{array}$$

The mapping κ points out that antidote and poison do not happen by default. World s required to fulfil $M \models [\vec{x}', \vec{w}']\neg\varphi$ has a raking 2, whereas the actual world s^* has ranking 1 and, consequently, the new condition 2a does not hold.

The following labelled program captures the intended meaning given by HP01 to this example[1]:

Program 8.21.

$$\begin{array}{lll}
\$: & survive & \leftarrow antidote \\
\$: & survive & \leftarrow not\ poison \\
\$: & antidote &
\end{array}$$

In the (total) well-founded causal model and unique causal stable model I of this program $I(survive) = survive$, that is, it just holds by default, and there is not any actual cause for it. If we add the labelled fact $(\$: poison)$, then

$$I(survive) = antidote \cdot survive$$

that is, *antidote* becomes a contributory cause of *survive*. Note that, *default negation* is used here for capturing the idea that *not poison* is the normal situation or, alternatively, that it holds by default.

Another interesting example has been provided by Hall and Paul [2003]:

1 It may seem counter-intuitive that *antidote* causes *survive*, but note the difference between *survive* and *alive*. When we are talking about surviving, we are assuming a death threat, and so, the expected (default) outcome becomes *death* instead of *alive*, which would be the default in the absence of the threat. A more elaborated representation of this example should include the existence of that threat, in its turn, caused by *poison*.

Example 8.5 (Water Plant)**.** *Suppose Suzy goes away on vacation, leaving her favourite plant in the hands of Billy, who has promised to water it. Billy fails to do so. The plant dies but would not have, had Billy watered it... Billy's failure to water the plant caused its death. But Vladimir Putin also failed to water Suzy's plant. And, had he done so, it would not have died. Why do we also not count his omission as a cause of the plant's death?*

Halpern and Pearl [2005] argue that:

> *"Billy is clearly a cause in the obvious structural model. So is Vladimir Putin, if we do not disallow any settings and include Putin watering the plant as one of the endogenous variables. However, if we simply disallow the setting where Vladimir Putin waters the plant. [T]hen Billy's failure to water the plants is a cause, and Putin's failure is not. We could equally well get this result by not taking Putin's watering the plant as one of the endogenous variables in the model. (Indeed, we suspect that most people modeling the problem would not include this as a random variable.)"*

On the contrary, any elaboration tolerant representation should be unaffected by the inclusion of unrelated events like Putin not watering the plant. In this sense, we may represent this scenario as the following labelled program:

Program 8.22.

$$\begin{array}{lll}
 & \mathit{death} & \leftarrow \mathit{not}\ \overline{\mathit{death}}, \mathit{not}\ \mathit{promise} \\
 & \overline{\mathit{death}} & \leftarrow \mathit{not}\ \mathit{death}, \mathit{not}\ \overline{\mathit{promise}} \\
 & & \\
 & \mathit{water} & \leftarrow \mathit{water}(X) \\
 & \mathit{promise} & \leftarrow \mathit{promise}(X) \\
 & \overline{\mathit{promise}} & \leftarrow \mathit{not}\ \mathit{promise} \\
 & & \\
\$: & \overline{\mathit{death}} & \leftarrow \mathit{water}(X) \\
\$: & \mathit{death} & \leftarrow \overline{\mathit{water}}(X), \mathit{not}\ \overline{\mathit{promise}}(X), \mathit{not}\ \mathit{water} \\
 & & \\
 & \overline{\mathit{water}}(X) & \leftarrow \mathit{not}\ \mathit{water}(X) \\
 & \overline{\mathit{promise}}(X) & \leftarrow \mathit{not}\ \mathit{promise}(X) \\
 & & \\
\$: & \mathit{promise}(\mathit{billy}) & \\
\$: & \overline{\mathit{water}}(\mathit{billy}) &
\end{array}$$

The first two rules of this program state the default behaviour of the plant. On the one hand, the plant will die by default unless somebody has promised to water it. On the other hand, if somebody has promised to water it, we will expect the plant to continue alive. Literal *promise* means that somebody has promised to water the plant. Similarly, literal *water* means that somebody has watered the plant. The two next rules state that, the plant will survive anyway if it is watered, and that it will die if who promise water it does not fulfil her promise. The following two rules state that, by default, people do not water the plant nor promise to do it. Last, Billy promised to water the plant and failed to do it. The only causal stable model I of this program satisfies that:

$$I(death) \quad = \quad \overline{water}(billy) \cdot death \tag{106}$$

regardless of whether Putin is considered in the program or not, as long as Putin did not promise to water the plant. Hence, we may add the following labelled fact:

$$\$: \quad \overline{water}(putin)$$

to Program 8.22 and we will still conclude (106) as the sufficient cause of *death*. In this sense Billy alone, and not Putin, is a contributory cause of *death*. Of course, if we add now the labelled fact

$$\$: \quad promise(putin)$$

indicating that Putin has promised to water Suzy's plant, then Putin will become a contributory cause in the same sense as Billy.

Another interesting example is the gear scenario of Example 5.4 that we have discussed in Chapter 5. This example is an usual benchmark in the AI literature for testing the behaviour of causal cycles. We have seen that when the first motor is started at situation 1 and the gears are coupled at situation 3, we obtain that the second wheel is turning because of causal graph G_1 in Figure 25. This causal graph contains the labels $start(1)_1$ and $couple_3$ corresponding to the homograph facts, so that both are contributory causes of $spinning(2)_4$ according to Definition 8.19. On the other hand, structural models are usually assumed to be acyclic, fact that will complicate the representation of this example. Halpern and Pearl [2005] provide, in Appendix A.4, an attempt to define actual causes for structural model containing cycles. We will follow here their attempt and represent this scenario by the following model.

Structural Model 8.23.

$$\begin{array}{lll}
spinning(a)_{s+1} = & trans(a)_{s+1} \lor start(a)_s \lor inert(a)_{s+1} & \text{for } t = 0,1,2,3 \\
spinning(b)_{s+1} = & trans(b)_{s+1} \lor start(b)_s \lor inert(b)_{s+1} & \text{for } t = 0,1,2,3 \\
inert(a)_{s+1} = & spinning(a)_s \land \neg spinning(a)_{s+1} & \text{for } t = 0,1,2,3 \\
inert(b)_{s+1} = & spinning(b)_s \land \neg spinning(b)_{s+1} & \text{for } t = 0,1,2,3 \\
trans(a)_{s+1} = & spinning(b)_s \land coupled_s & \text{for } t = 0,1,2,3 \\
trans(b)_{s+1} = & spinning(a)_s \land coupled_s & \text{for } t = 0,1,2,3 \\
spinning(a)_0 = & 0 & \\
spinning(b)_0 = & 0 & \\
inert(a)_0 = & 0 & \\
inert(b)_0 = & 0 & \\
trans(a)_0 = & 0 & \\
trans(b)_0 = & 0 & \\
start(a)_1 = & 1 & \\
start(a)_s = & 0 & \text{for } t = 0,2,3,4 \\
couple_3 = & 1 & \\
couple_s = & 0 & \text{for } t = 0,1,2,4
\end{array}$$

However, there exists a solution of these equations in which $spinning(a)_3 = 0$, and hence $spinning(a)_4 = 0$ and $spinning(b)_4 = 0$ too. Since the condition 2b of the new definition requires that $spinning(b)_4$ holds in all solutions, it is clear that $start(a)_2$ is not recognised to be an actual cause of $spinning(b)_4$. As a conclusion, we can see that this definition is not adequate for representing examples that are inherently cyclic.

In all the previous examples our definition of *contributory cause* coincides with Halpern and Pearl's concept of *actual cause*. Hall [2004, 2007] pointed out the existence of at least two different concepts that are usually covered by the wider concept of *causation*. He calls these more fine concepts *production* and *dependence*. The concept of *production* relies on the following idea: causality is *transitive*, *local* and *intrinsic*. In our approach, intrinsicness is reflected in the following idea: a sufficient cause suffices to produce the effect under circumstances that are *not more abnormal* than the current circumstances. Furthermore, by definition we have assumed that causality is transitively propagated by rules, so that our definition is transitive *a priory*. However, this does means that it is transitive *a posteriori* due to the minimality criterion. For instance, in the circuit of Example 3.2, switch 4 is a contributory cause of the current at point b, and this, in its turn, is a contributory cause of the *alarm* firing, but, as we have already discussed, switch 4 is not part of any sufficient cause of the firing *alarm*, and so it is not a contributory cause of such firing either. Finally, locality refers to the fact that causes and effects must be connected via spatiotemporally continuous sequences of causal intermediates. We have already commented how the principle of locality is manifested in our approach when we discussed the role played by variables *dehydration* and *intake* in the desert-travel scenario of Example 1.8.

As opposed to *production*, *dependence* relies on the idea that counterfactual dependence between wholly distinct events is sufficient for causation. In these sense, it is easy to see that our concept of *contributory cause* follows the concept of *production* rather than the *dependence* one. On the other hand, Halpern and Pearl's concept of *actual causation* follows the concept of *dependence* rather than *production*. Hall [2007] illustrates the difference between these two concepts by the following two examples relying on the so-called neuron diagrams.

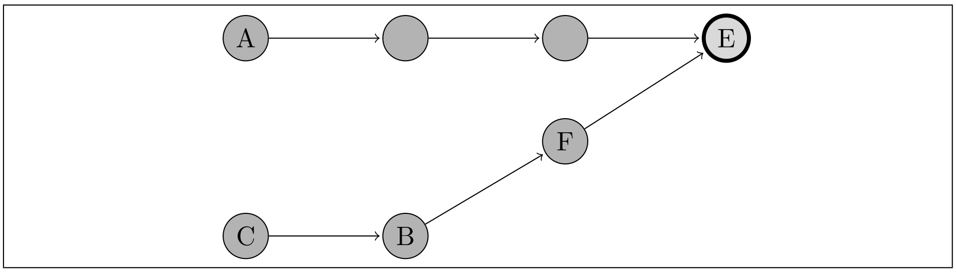

Figure 45: A and C are joint productive causes of E.

Each circle on those neuron diagrams represents a neuron. A grey filled circle means that such neuron is active. Arrows are stimulatory paths, that is, if the origin neuron is active, so it will be its destination. An arrow with a circled head is an inhibitory path. Inhibitory paths take preference over stimulatory ones, that is, if one inhibitory path is active, the neuron will remain inactive, regardless of the incoming stimulatory paths. A thicker circle around a neuron represents an *stubborn neuron* which requires at least two active stimulatory paths to be active. In the neuron diagram represented by Figure 45, neurons A and C are joint productive causes of E. On the other hand, in Figure 46, A is the only productive cause of E, while E still depends on C. Note the

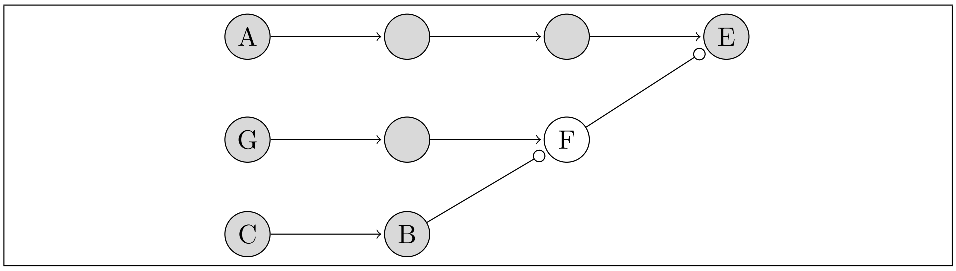

Figure 46: *E* depends on *C* which is not productive causes of it.

difference in the relation between C and E in these two cases. In Figure 45, C is required to activate the stubborn neuron E, while in Figure 46, C just allows A to produce E by preventing that G prevents E. To further illustrate the intuition of Figure 46, consider the following history matching its structure. Neuron A represents a family which is sleepy, and which in the end causes them to go to sleep (neuron E). Neuron C represents the watching police that captures a gang of thieves that would disturb the family sleeping, allowing, but not *producing*, them to continue sleeping. As we have commented, since E depends on C, Halpern and Pearl's approach will count C to be an *actual cause* of E, while, since C is not a producer of E, we will not count it as a *contributory cause*. Both approaches will recognise A as a cause of E. Figure 47 shows the

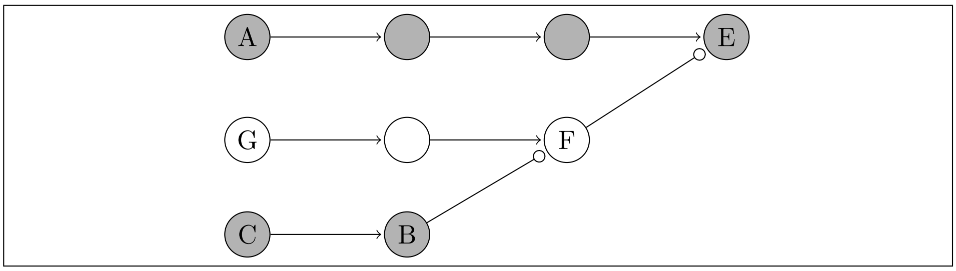

Figure 47: *E* depends on *C* which is not productive causes of it.

case in which the police is watching but there are no thieves. In such a case, E does not depend on C. This shows that the dependence relation between E and C is not intrinsic, that is, it depends on the existence of some kind of external threat G. Despite the fact that E does not depend on C, both HP01 and HP05 definitions will still count C as an actual cause of E. To overcome this issue, Hall [2007] has made use of *default knowledge*, requiring that dependence is only considered in a "less abnormal" world. Considering that neurons are non-firing by default, a contingency in which G fires is "more abnormal" than the current state. Consequently, the dependence in this world does not allow counting C as an actual cause of E.

A dependence relation may be captured in our formalism using the CP-values algebra from Section 8.1. Using that semantics on the structure of Figure 46 we get the value $A * \sim\sim C$ pointing out that A is a productive cause of E which counterfactually depends on C. For Figure 47, E will be assigned the value A, that is, A is a productive cause of E without further dependences. Another point where the CP-values algebra may be useful is in the following example due to Hall [2000]:

Example 8.6 (The Engineer)**.** *An engineer is standing by a switch in the railroad tracks. A train approaches in the distance. She flips the switch, so that the train travels down the right-hand track, instead of the left. Since the tracks reconverge up ahead, the train arrives at its destination all the same; let us further suppose that the time and manner of its arrival are exactly as they would have been, had she not flipped the switch.* □

Hall [2007] characterises this kind of examples with the neuron diagram depicted in Figure 48. Neuron S acts as a "switch:" B would fire interdependently

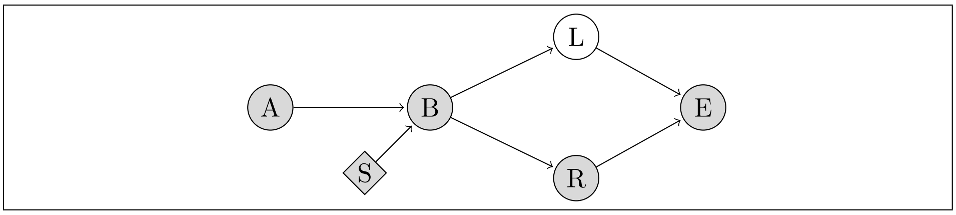

Figure 48: The train is deliver to the right-hand track.

of whether S fires or not, but if S fires it will activate the lower stimulatory path to R, and if not, it will activate the upper one to L. Neuron A represents the train approaching, L and R respectively represent the train going through the left and right tracks. Neuron E represents the arrival of the train to its destination. Figure 48 shows the neuron diagram corresponding to the scenario were the engineer does not switch the railroad tracks. This has been a controversial example. For instance, Hall [2000] has argued that the switch should be considered a cause of the arrival, while in [Hall, 2007] he argues otherwise. If the switch is intended to be considered a cause, we may just represent this scenario by the following logic program.

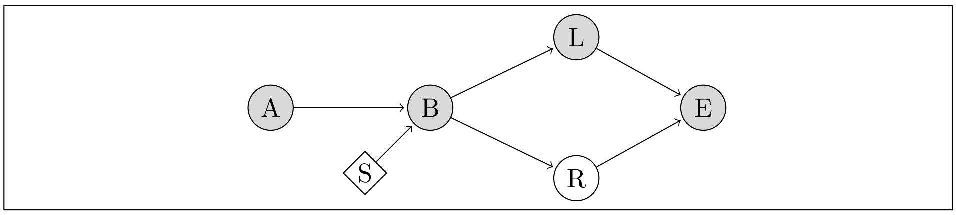

Figure 49: The train is deliver to the left-hand track.

Program 8.24.

$$\begin{aligned}
\$: &\quad arrive \leftarrow left\\
\$: &\quad arrive \leftarrow right\\
\$: &\quad left \leftarrow train, \overline{switch}\\
\$: &\quad right \leftarrow train, switch\\
\$: &\quad train\\
\$: &\quad switch
\end{aligned}$$

The least model of this program satisfies that:

$$I(arrive) = (train * switch) \cdot right \cdot arrive$$

Hence *switch* is considered to be contributory cause of *arrive*. However, common sense seems to point out that moving the switch is only a cause for the chosen route but not for the arrival itself. This can be captured by the CP-algebra, by adding the following two rules to Program 8.24:

$$\begin{aligned}
switch &\leftarrow not\ \overline{switch}\\
\overline{switch} &\leftarrow not\ switch
\end{aligned}$$

stating that the switch behaves "classically," that is, the law of the excluded in the middle holds. The well-founded model of this program satisfies that:

$$\begin{aligned}
I(arrive) = & (train * switch) \cdot left \cdot arrive\\
& + (train * {\sim}switch) \cdot right \cdot arrive
\end{aligned}$$

Note that *switch* is a contributory cause with respect to the first addend and, at the same time, it is preventing the same effect in the second addend. A definition of actual cause can be based in a refinement of Definition 8.19 which takes into account that a contributory cause cannot be an actual cause if it is simultaneously preventing the same effect that is helping to produce. A deeper study of *causal dependence* and switching is left for future work.

9 IMPLEMENTATION

In this chapter, we present a prototype that implements some of the concepts presented in this dissertation. To illustrate the input syntax of the prototype consider, for instance, the suitcase-bomb scenario of Example 1.3.

```
1   time(1..4).
2   agent(suzy).
3   agent(billy).
4
5   b    :: bomb(T)        :- open(T), not bomb(T-1).
6   u    :: open(T)        :- up(T,1), up(T,2), not open(T-1).
7   l(L) :: up(T+1,L)      :- lift(T,A,L), not up(T,L).
8   p    :: prison(T+1,A) :- #hascaused(A, bomb(T)), agent(A).
9
10          up(T+1,L)      :- up(T,L), not down(T+1,L), time(T).
11          open(T+1)      :- open(T), not close(T+1), time(T).
12          bomb(T+1)      :- bomb(T), time(T).
13
14  :: lift(1,suzy,1).
15  :: lift(3,suzy,2).
16
17  :: lift(3,billy,1).
18  :: lift(4,billy,2).
19
20  suzy  #does lift(T,suzy,L).
21  billy #does lift(T,billy,L).
```

Labels are separated from rules by two colons "::" instead of a single colon ":" for avoiding the confusion with other uses of the colon in standard ASP. If the name of the rule label is omitted, we assume that the rule is labelled with the same name as the head atom. Hence, actions (lines 14-18) have been labelled by an homonymous label. Unlabelled rules in lines 10-12 represent the inertia axioms for fluents *up*, *open* and *bomb* respectively. Note that the fact that the *bomb* has exploded cannot be reversed, so there is not exception to its inertia. The rule in line 8 captures the statement that "whoever causes a bomb explosion will be punished with imprisonment." The causal literal "$\#hascaused(A, bomb(T))$"

is true when A has caused the $bomb(T)$ according to Definition 6.2. The *#does* directives in lines 20-21 specify which actions have been performed by each agent. This program has a unique causal stable model, which can be computed by the following command line[1].

```
python ./cgraphs ./examples/judge.lp
```

Lines 22-26 in the below listing show the literals that holds in the only causal stable model of this program, that is those for which there exists some cause to hold. For instance, *suzy* will be in prison from situation 4 as the cause of the *bomb* explosion, while *billy* will not.

```
22 Answer 1:
23 agent(billy) agent(suzy) bomb(4) bomb(5) lift(1,suzy,1)
24 lift(3,billy,1) lift(3,suzy,2) lift(4,billy,2) open(4) open(5)
25 prison(5,suzy) prison(6,suzy) time(1) time(2) time(3) time(4)
26 up(2,1) up(3,1) up(4,1) up(4,2) up(5,1) up(5,2)
27
28 bomb(4)          = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b
29 bomb(5)          = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b
30 lift(1,suzy,1)   = lift(1,suzy,1)
31 lift(3,billy,1)  = lift(3,billy,1)
32 lift(3,suzy,2)   = lift(3,suzy,2)
33 lift(4,billy,2)  = lift(4,billy,2)
34 open(4)          = (lift(3,suzy,2).l(2)*lift(1,suzy,1).l(1)).u
35 open(5)          = (lift(3,suzy,2).l(2)*lift(1,suzy,1).l(1)).u
36 prison(5,suzy)   = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b.p
37 prison(6,suzy)   = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b.p
38 up(2,1)          = lift(1,suzy,1).l(1)
39 up(3,1)          = lift(1,suzy,1).l(1)
40 up(4,1)          = lift(1,suzy,1).l(1)
41 up(4,2)          = lift(3,suzy,2).l(2)
42 up(5,1)          = lift(1,suzy,1).l(1)
43 up(5,2)          = lift(3,suzy,2).l(2)
```

The remaining lines show the causes of those literals that hold as causal terms. Literals whose cause is 1 are conveniently hidden for the sake of compactness. The above program also has a total causal well-founded model that can be computed by the following command line[2]:

```
python ./cgraphs --well-founded ./examples/judge.lp
```

1 The program *cgraphs* runs over phython, www.python.org, and makes use of the *gringo* Python library, version 4.4.0, which is assumed to be accessible in the Python path. The *gringo* library is free accessible at the Potassco web page: http://potassco.sourceforge.net.

2 The causal well-founded model can be computed even in the absence of the *gringo* library.

The following listing shows the result of computing the causal well-founded model of the above program, which coincides with the unique answer set of such program.

```
Well-Founded Total Model
agent(billy) agent(suzy) bomb(4) bomb(5) lift(1,suzy,1)
lift(3,billy,1) lift(3,suzy,2) lift(4,billy,2) open(4) open(5)
prison(5,suzy) prison(6,suzy) time(1) time(2) time(3) time(4)
up(2,1) up(3,1) up(4,1) up(4,2) up(5,1) up(5,2)

bomb(4)          = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b
bomb(5)          = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b
lift(1,suzy,1)   = lift(1,suzy,1)
lift(3,billy,1)  = lift(3,billy,1)
lift(3,suzy,2)   = lift(3,suzy,2)
lift(4,billy,2)  = lift(4,billy,2)
open(4)          = (lift(3,suzy,2).l(2)*lift(1,suzy,1).l(1)).u
open(5)          = (lift(3,suzy,2).l(2)*lift(1,suzy,1).l(1)).u
prison(5,suzy)   = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b.p
prison(6,suzy)   = (lift(1,suzy,1).l(1)*lift(3,suzy,2).l(2)).u.b.p
up(2,1)          = lift(1,suzy,1).l(1)
up(3,1)          = lift(1,suzy,1).l(1)
up(4,1)          = lift(1,suzy,1).l(1)
up(4,2)          = lift(3,suzy,2).l(2)
up(5,1)          = lift(1,suzy,1).l(1)
up(5,2)          = lift(3,suzy,2).l(2)
```

If lines 14-15, containing Suzy's actions, are removed from the above program, the bomb would still explode, but this time due to Billy's actions. The following listing shows the answer in such case.

```
Answer 1:
agent(billy) agent(suzy) bomb(5) lift(3,billy,1)
lift(4,billy,2) open(5) prison(6,billy) time(1) time(2)
time(3) time(4) up(4,1) up(5,1) up(5,2)

bomb(5)          = (lift(3,billy,1).l(1)*lift(4,billy,2).l(2)).u.b
lift(3,billy,1)  = lift(3,billy,1)
lift(4,billy,2)  = lift(4,billy,2)
open(5)          = (lift(4,billy,2).l(2)*lift(3,billy,1).l(1)).u
prison(6,billy)  = (lift(3,billy,1).l(1)*lift(4,billy,2).l(2)).u.b.p
up(4,1)          = lift(3,billy,1).l(1)
up(5,1)          = lift(3,billy,1).l(1)
up(5,2)          = lift(4,billy,2).l(2)
```

As expected, in this case it is Billy who ends in prison instead of Suzy. This example illustrates the degree of elaboration tolerance of our formalism: a change in the actions that have occurred do not require any change in the model but for those facts representing them.

It is also possible to obtain the causal information associated to atoms in a graph format instead of the term format[3]

```
python --img=graphs ./cgraphs ./examples/judge.lp
```

By using the above command line, a folder graphs/answer_n is created for each answer set (where n is a number). Inside this folder, a file is created with name atom.pdf for each atom. This file contains the causal value of literal depicted as a graph. For instance, Figure 50 depicts the only cause of $\mathit{prison}(6,\mathit{suzy})$ in our former example. The vertices corresponding to the causal graph are hexagonal. In addition, an oval vertex with the corresponding literal is added. When a literal has more than one associated cause, each one is represented by a different connected component. For instance, consider the program corresponding to the symmetric behaviour representation of Example 1.4. The following listing shows the program capturing this variation.

```
time(1..5).

o(S)     ::  caused(open(S))    :- up(a,S), up(b,S), caused(o,S).
k(S)     ::  caused(open(S+1)) :- key(S).
u(L,S)   ::  caused(up(L,S+1)) :- lift(L,S).

caused(o,S) :- caused(up(a,S)).
caused(o,S) :- caused(up(b,S)).

open(S)     :- caused(open(S)).
up(L,S)     :- caused(up(L,S)).


open(S+1)   :- open(S), not caused(nopen(S+1)), time(S).

up(A,S+1)   :- up(A,S), not caused(nup(A,S+1)), time(S).

lift(a,1) :: lift(a,1).
lift(b,3) :: lift(b,3).

key(4)    :: key(4).

#hide caused(_).
#hide caused(_,_).
```

3 Requires *graphviz*, version 2.38 www.graphviz.org and the python *graphviz library*, version 0.4.3 (install with command: pip install graphviz, if python pip is not installed run before: python get-pip.py).

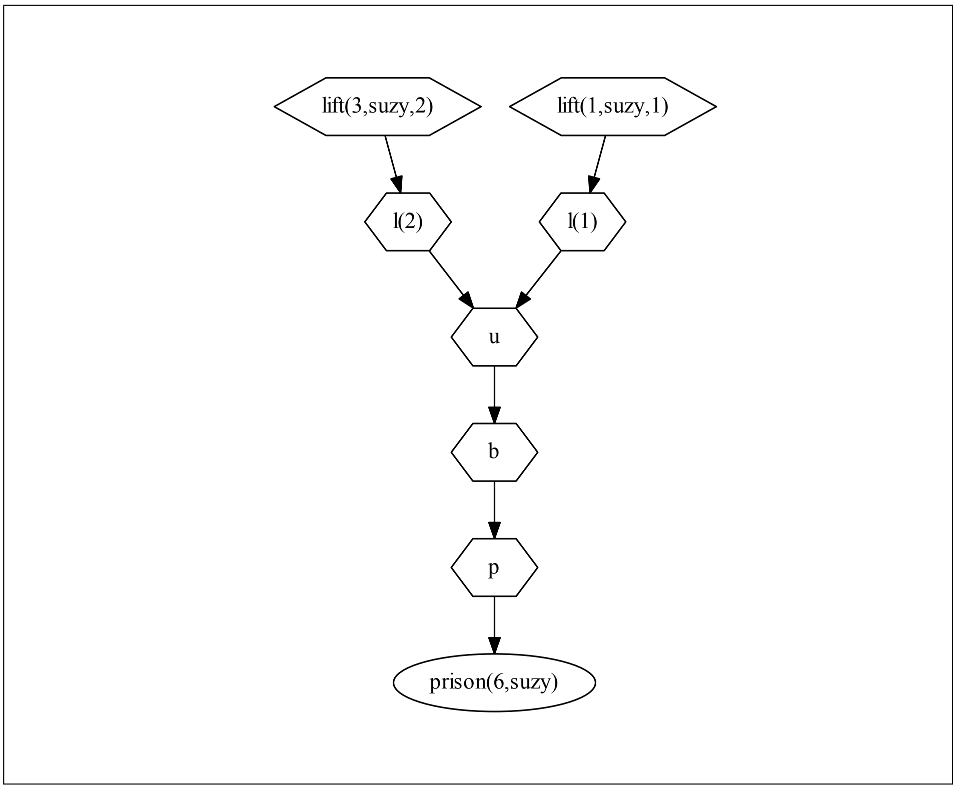

Figure 50: The cause of $prison(6,suzy)$.

The following listing shows the unique causal answer set of this program:

```
Answer 1:
key(4) lift(a,1) lift(b,3) open(4) open(5) open(6) time(1)
time(2) time(3) time(4) time(5) up(a,2) up(a,3) up(a,4)
up(a,5) up(a,6) up(b,4) up(b,5) up(b,6)

key(4)    = key(4)
lift(a,1) = lift(a,1)
lift(b,3) = lift(b,3)
open(4)   = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(5)   = key(4).k(4)
          + (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(6)   = key(4).k(4)
          + (lift(a,1).u(a,1)*lift(b,3).u(b,3)).o(4)
up(a,2)   = lift(a,1).u(a,1)
up(a,3)   = lift(a,1).u(a,1)
up(a,4)    = lift(a,1).u(a,1)
up(a,5)    = lift(a,1).u(a,1)
up(a,6)    = lift(a,1).u(a,1)
up(b,4)    = lift(b,3).u(b,3)
up(b,5)    = lift(b,3).u(b,3)
up(b,6)    = lift(b,3).u(b,3)
```

In contrast with our former example, we have here there are two different causes of $open(6)$. Figure 51 shows the graphs corresponding to those causes. Each connected component corresponds to one alternative cause.

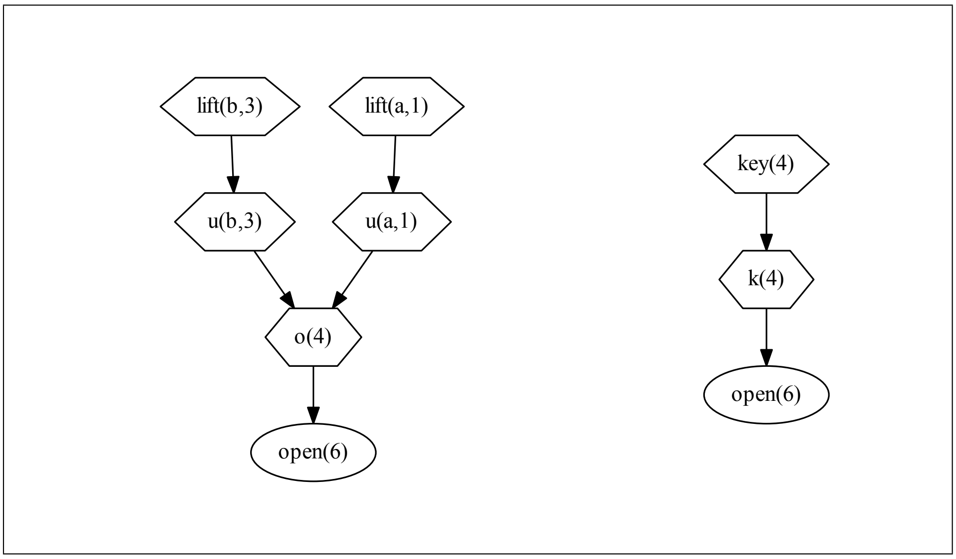

Figure 51: Causes of $open(6)$.

Figure 52 shows the flowchart of the application *cgraphs*, starting from the input causal program, and ending with the answer with the corresponding causal stable models. The parser and grounder modules are modifications of the tool *pyngo* [Konig, 2009] to cover the new syntax of causal programs: labels, causal literals of the form "$\#hascaused(A,B)$" and agent *#does* directives. Syntactically, labels have the same form of atoms and can contain variables. As usual variables appearing in the head or in the negative body of a rule are required to appear in some positive literal in the body. In addition, variables in labels must appear in some positive standard literal in the rule body. Causal literals are of the form

```
<cliteral> ::= [not] #hascaused(<cg-term>, <atom>)
             | [not] #hascaused(<variable>, <atom>)
```

Recall from Definition 3.12 that a cg-term is a term without sums (+). The test performed by a causal literal, can be stated by a cg-term or by a variable. For the latter, the term resulting from the rule grounding is considered to be a a label. For instance, the grounding of rule in line 8 consists of the following pair of rules:

```
p ::  prison(t,suzy)  :- #hascaused(suzy,  bomb(t)), agent(suzy).
p ::  prison(t,billy) :- #hascaused(billy , bomb(t)), agent(billy).
```

for $t \in \{4,5\}$. For $t \notin \{4,5\}$, the standard literal $bomb(t)$ does not hold, therefore no ground rules are produced for those values of t. Recall that the causes of *bomb* correspond to the term:

$$\Big(lift(1,suzy,1) \cdot l(1) * lift(3,suzy,2) \cdot l(2)\Big) \cdot u \cdot b \tag{107}$$

To inspect the influence of Suzy in the explosion of the *bomb*, we need to indicate the actions performed by Suzy. This is done in line 20:

```
suzy #does  lift(T,suzy,L).
```

This *action assignment* indicates that all labels that unify with $lift(T,suzy,L)$ are considered actions performed by Suzy. As a consequence, the causal literal "#hascaused(suzy, bomb(t))" hold for $t \in \{4,5\}$.

An action assignment is a set of rules such that each rule matches the follow grammar:

```
<action assignment> ::= <label>    #does <listOfLabels> .
                      | <variable> #does <listOfLabels> .

<listOfLabels>      ::= <label> [ , <listOfLabels> ]
```

All variables on the left hand side of #does must appear on the right. By using variables we may abbreviate the action assignments in lines 21-22 as:

```
A #does  lift(T,A,L).
```

That is, every label that unifies with $lift(T, A, L)$ is considered to be performed by the substitution corresponding to variable A.

In the rest of this chapter we detail how to obtain the causal stable models of a ground causal program. Recall from Chapter 4 that Theorem 4.4 shows that, for programs without non-standard causal literals, there is a one-to-one correspondence between the causal stable models of a program and their standard stable models. In such case, we may compute the causal stable models of a program by obtaining a standard stable model, for instance by means of some answer set solver, and use it to compute the reduct of the program. Then, we just need to compute the least model of the reduct to obtain a causal stable model.

Unfortunately, for evaluating the non-standard causal literals we need to know the causal value associated to its standard literal, and hence the above procedure does not work. Then, for this class of programs, we may compute a causal stable model by relying on its definition, that is, for all possible causal interpretation we compute the reduct of the program, its least model, and check that this corresponds to the interpretation used for computing the reduct.

For improving this naive approach, the following splitting Theorem and notation will be useful. By abuse of notation, for any interpretation I, we denote by I a set of rules of the form:

$$I \overset{\text{def}}{=} \{ A \leftarrow t \mid A \in At \text{ and } t = I(A) \}$$

Note the convenience of this notation: the least least model of the program I is the interpretation I.

Proposition 9.1. *Let I be a causal stable model of a program P. Then, the interpretation I is also the least model of the program I.* □

Proof. The proof can be found in Appendix A on page 288. □

Furthermore, given a set S of literals, we denote by $I_{|S}$ an interpretation such that $I_{|S}(A) = I(A)$ for all $A \in S$ and $I_{|S}(A) = 0$ for all $A \notin S$. We can then obtain the following result.

Theorem 9.1 (Splitting). *Let P_1 and P_2 be two programs such that no literal occurring in P_1 is a head atom of P_2. Let S be a set of atoms containing exactly the atoms occurring P_1 but no head atoms of P_2. An interpretation I is a causal stable model of $P_1 \cup P_2$ iff $I_{|S}$ is causal stable model of P_1 and I is the causal stable model of $I_{|S} \cup P_2$.* □

Proof. The proof can be found in Appendix A on page 289. □

We may make use of Theorem 9.1 for removing all negative non-standard literals from the input program, and obtaining program P_1 in Figure 52.

Proposition 9.2. *Let P be a causal program and R be a rule in P of the form:*

$$l_R: \quad A \leftarrow B_1, \ldots, B_m, \text{not } C_1, \ldots, \text{not } C_i, \ldots, \text{not } C_n$$

with C_i a non-standard causal literal. Let P_1 be the result of replacing R in P by

$$\begin{aligned} l_R: \quad A &\leftarrow B_1,\ldots,B_m,\text{not } C_1,\ldots,\text{not } aux,\ldots,\text{not } C_n \\ aux &\leftarrow C_i \end{aligned}$$

where aux is an auxiliary atom not appearing in P. Let I and I' be two causal interpretations such that:

- $I_1(aux) = I(C_i)$.
- $I = I_{1|S}$ *with* $S = Lit \backslash \{aux\}$

Then, I is a causal stable model of P if and only if I_1 is a causal stable model of P_1. □

Proof. Let $P' = \{aux \leftarrow C_i\}$. Note that, since $I(aux) = 0$, it follows that I_1 is the unique causal stable model of $I \cup P'$. Hence, since aux does not occur in P, from Theorem 9.1, it follows that I is a causal stable model of P if and only if I_1 is a causal stable model of $P \cup P'$. Finally note that, since $I_1(aux) = I(C_i)$, I_1 is a causal stable model of $P \cup P'$ if and only if I_1 is a causal stable model of P_1. □

Proposition 9.2 allows us to remove, one by one, all the negative causal literals of a program. In the light of this result, we do not need to guess a causal interpretation to compute the reduct of a program, but only a set of literals that are assumed to be true.

The next step is trying to remove each of the non-standard causal literals of the form $(\psi_i :: B_i)$ replacing it by an auxiliary atom aux and a pair of rules of the form:

$$\begin{aligned} aux &\leftarrow B_i, \textit{not } \overline{aux} \\ \overline{aux} &\leftarrow \textit{not } aux \end{aligned}$$

Note that, if B_i does not hold, then no causal literal containing it holds either. It is clear that this transformation loses the one-to-one correspondence. Instead, we only we have a complete, but not sound, result.

Proposition 9.3. *Let P_2 be a causal program and R be a rule in P_2 of the form:*

$$l_R : \quad A \leftarrow B_1, \dots, (\psi_i :: B_i), \dots, B_m, \text{not } C_1, \dots, \text{ not } C_n$$

Let P_3 be the result of replacing R in P_2 by

$$\begin{aligned} l_R : \quad A &\leftarrow B_1, \dots, aux, \dots, B_m, \text{not } C_1, \dots, \text{not } C_n \\ aux &\leftarrow B_i, \text{not } \overline{aux} \\ \overline{aux} &\leftarrow \text{not } aux \end{aligned}$$

where aux and $\overline{aux}$ auxiliary atoms not in P. If I_2 is a causal stable model of P_2, there is a causal stable model I_3 of P_3 such that

- $I_3(aux) = 1$ *iff* $I_2(\psi_i :: B_i) \neq 0$; $I_3(aux) = 0$ *otherwise.*
- $I_3(\overline{aux}) = 1$ *iff* $I_3(aux) \neq 0$; $I_2(\overline{aux}) = 0$ *otherwise.*
- $(I_2)^{cl} = (I_{3|S})^{cl}$ *with* $S = Lit \backslash \{aux, \overline{aux}\}$. □

Proof. The proof can be found in Appendix A on page 290. □

Corollary 9.1. *Let P_2 be a causal program. Let P_3 be the result of replacing all non-standard causal literals by auxiliary atoms as done in Proposition 9.3 and removing all the rule labels. If I_2 is a causal stable model of P_2, there is a stable model I_3 of P_3 such that*

- $I_3(aux) = 1$ *iff* $I_2(\psi_i :: B_i) \neq 0$; $I_3(aux) = 0$ *otherwise.*
- $I_3(\overline{aux}) = 1$ *iff* $I_3(aux) \neq 0$; $I_2(\overline{aux}) = 0$ *otherwise.*
- $(I_2)^{cl} = I_{3|S}$ *with* $S = Lit \backslash \{aux, \overline{aux}\}$. □

Proof. Let P_3' be the program resulting by removing all non-standard causal literals by auxiliary atoms as done in Proposition 9.3. Then P_3' is a labelled program (without non-standard causal literals) and there is an interpretation I_3' satisfying the three conditions of this statement. Hence, from Theorem 4.4, $I_3 \stackrel{\text{def}}{=} (I_3')^{cl}$ is a stable model of P_3 satisfying these three properties. □

Given a causal program P_2 without negative causal literals, we may obtain a standard program P_3 by replacing, one-by-one, all non-standard causal literals by auxiliary atoms as in Proposition 9.3, and then removing all its labels. If a set of literals S is a standard stable model of P_3, the least model I of the reduct of P_2^S is a candidate to be a causal stable model. To check whether I is a causal stable model of P_2 we only need to check whether $I(\psi_i :: B_i)$ holds if and only if aux_i is in S, where aux_i is the auxiliary atom that replaces $(\psi_i :: B_i)$. Corollary 9.1 shows that there does not exist more candidates. That is, this procedure is correct and complete for obtaining the causal stable model of a program P_2 without negative causal literals.

However, the evaluation of non-standard causal literals is blindly guessed, so the number of candidate models that the solver needs to compute is exponential in the number of those non-standard causal literals. For improving this procedure, we may rely on the causal well-founded model. Recall from Chapter 7 that deciding whether a causal literal is a consequence of a program under the well-founded semantics is feasible in polynomial time. Proposition 4.1 stated that a causal literal that holds with respect to $\texttt{lfp}(\Gamma^2_P)$, holds with respect to all causal stable models, and a causal literal that does not hold with respect to $\texttt{gfp}(\Gamma^2_P)$, does not hold with respect to any of them. Therefore, we may use the causal well-founded model for simplifying a program in the following way:

- Removing all rules with some positive causal literal that does not hold with respect to $\texttt{gfp}(\Gamma^2_P)$.
- Removing all rules with some negative causal literal that holds with respect to $\texttt{lfp}(\Gamma^2_P)$.
- Removing from the remaining rules those positive causal literals that hold with respect to $\texttt{lfp}(\Gamma^2_P)$.
- Removing from the remaining rules those negative causal literals that hold with respect to $\texttt{gfp}(\Gamma^2_P)$.

This procedure corresponds to obtaining program P_2 from program P_1 in the procedure described by Figure 52. This procedure keeps a one-to-one correspondence between the causal stable models of program P_1 and program P_2 in which the literals that hold in one of them hold in the other, and vice-versa. However, the causal stable models of both programs may differ in their associated causal values. In order to illustrate this fact, consider the following program whose causal well-founded model holds $\texttt{lfp}(\Gamma^2_P)(w) = a * b$.

```
a :: p.
b :: q.
     r :- p, q.
a :: s :- not t.
b :: t :- not s.
     r :- s.
     r :- t.

     w :- #hascaused(a*b,r).
```

```
#renaming a.
#renaming b.
```

Note that the causal literal "#hascaused(a*b,r)" holds with respect to the causal well-founded model, therefore we may remove it. This simplification leads us to the following program:

```
a :: p.
b :: q.
     r :- p, q.
a :: s :- not t.
b :: t :- not s.
     r :- s.
     r :- t.

     w.

#renaming a.
#renaming b.
```

whose causal stable models hold $I(w) = 1$. However, the causal stable models of the original program respectively satisfy $I_1(w) = a$ and $I_2(w) = b$. This example also illustrates that causal values associated to literals may not be the same even if, instead of removing the causal literal, this is replaced by its causal value. In our running example, this would consist in dealing with a rule of the form:

```
w :- a*b
```

which would lead to a causal stable model that satisfy $I_i(w) = a * b$ for $i \in \{1,2\}$.

Further improvements of this procedure may rely on Theorem 9.1 for splitting the program in smaller subprograms, whose causal stable models can be computed independently, and incorporating the causal information inside the solver in order to avoid blindly guessing the valuation of non-standard causal literals.

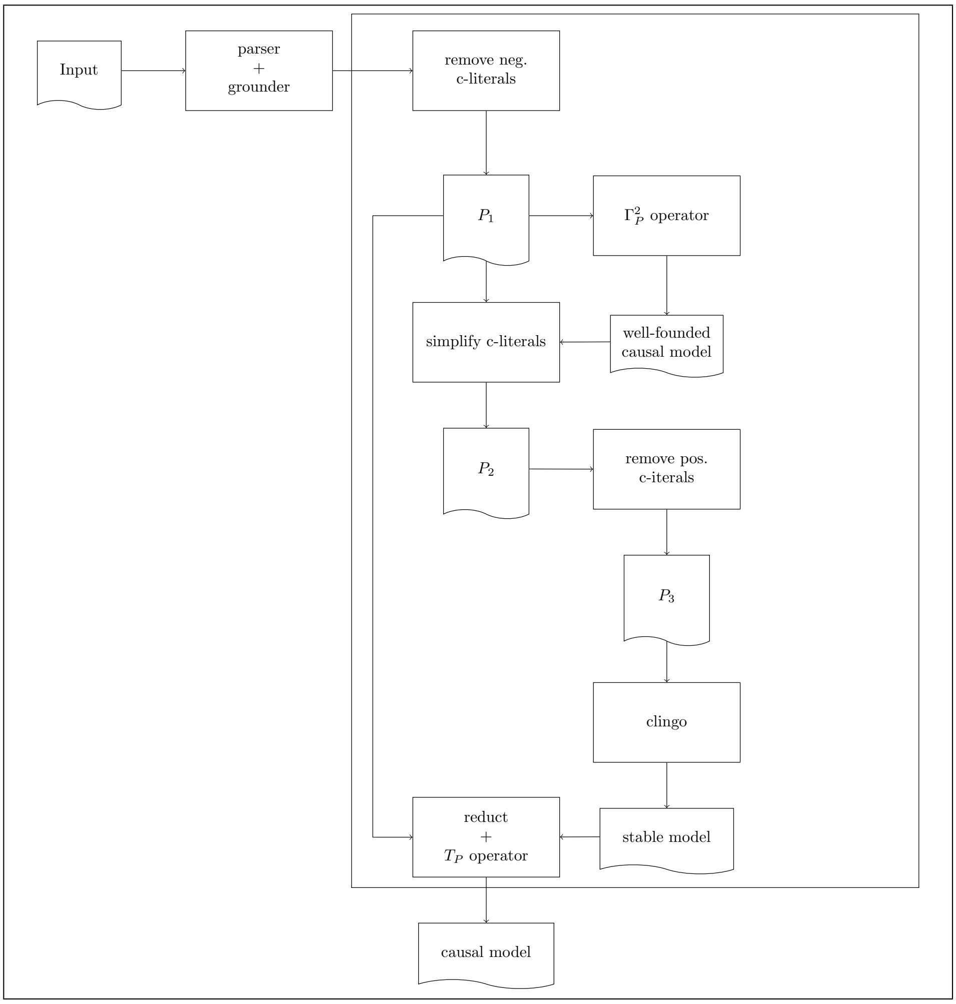

Figure 52: *cgraphs* flowchart.

10 CONCLUSIONS

In this dissertation, we have provided a logical semantics for representing and reasoning with causal explanations. In particular, the main contributions of this dissertation may be summarised as follows:

i) We have formally defined the concepts of causal graph and causal value which extend the Lewis' idea of causal chain distinguishing between joint and alternative causes. We have also defined the concepts of sufficient, necessary and contributory causes with respect to them. The concept of *sufficient cause* is closely related to a *non-redundant logical proof*: we have proved their one-to-one correspondence for programs in which every rule has a different label.

ii) We have studied the algebraic properties of causal values. In particular, we have shown that causal values can be manipulated by means of three algebraic operations $(\cdot)$, $(*)$ and $(+)$. Furthermore, we have shown that causal values are isomorphic to a free, completely distributive lattice generated by causal graphs, and so, the maximal causal terms without sums capture the notion of set of pairwise incomparable causal graphs.

iii) We have provided causal semantics for logic programs that are conservative extensions of the *least model*, the *stable model*, the *well-founded model* and the *answer set semantics*. We also have shown how causal information can be computed under those semantics. In particular, we have shown that causal information under the least model can be computed by an extension of the direct consequences operator for standard LP of van Emden and Kowalski [1976]. For the stable and the answer set semantics, causal information can be computed relying on the reduct of a program, similar as in standard LP [Gelfond and Lifschitz, 1988]. Finally, for the well-founded semantics, we may rely on an extension of the alternate fixpoint definition of Van Gelder [1989].

iv) We have explored how these semantics can be applied to KR problems for obtaining causal information. In particular, we have applied the causal answer set semantics for obtaining causal information when solving some of the traditional examples of the literature of Reasoning about Actions and Change. We also have shown how these semantics can be applied for representing some of the traditional examples of the literature on actual causation.

v) We have incorporated a new kind of causal literal that allows inspecting the causal information associated to standard ASP literals. In particular, we have used causal literals for defining a literal of the form *hascaused*(A, B) which holds when "A has been sufficient to cause B." Furthermore, we have shown how this kind of literal can be used for representing statements involving causality without falling in *problems of elaboration tolerance* [McCarthy, 1998].

vi) We have explored the computational cost of solving the decision problems associated to *entailment*, *sufficient* and *necessary causation*, and provided complete characterisations for them. In particular, we have shown that all these problems fit into the second level of the polynomial hierarchy, and that (under reasonable assumptions) the costs of both *entailment* and *sufficient causation* do not increase with respect to standard LP entailment.

vii) We have compared our work to justifications approaches for ASP and Halpern and Pearl's approach to actual causation. With respect to justifications approaches for ASP, we have shown a formal correspondence between *sufficient explanations* and *why-not provenance justifications* [Damásio et al., 2013] under the well-founded semantics. We have also informally shown the similarities and differences between our approach and approaches by Pontelli et al. [2009] and Schulz and Toni [2013, 2014].

viii) Finally, we have provided an implementation of a prototype that computes the sufficient causes of literals with respect to the well-founded and the stable model semantics. Furthermore, this tool incorporates the *hascaused*(A, B) predicate that allows, not only obtaining those causes, but also reasoning with them.

We next outline some of the aspects in which this work can be extended or completed in the future.

i) With respect to the semantics, there are two main issues that are worth to study. The first is replacing the reduct basic syntactic definition in favour of a logical treatment of default negation, as done for instance with the

Equilibrium Logic [Pearce, 1996] characterisation of stable models. The second consists in allowing rules to contain disjunctions in the head.

ii) With respect to the complexity assessment, from the definition of *contributory cause* it immediately follows the NP membership for the least and the well-founded semantics, and $\mathrm{NP}/\Pi_2^{\mathrm{P}}$ respectively for brave and cautions reasoning under the stable model semantics. Whether these characterisations are complete or not is still an open question.

iii) We have used causal literals to define the predicate $\mathit{hascaused}(A,B)$, and used the latter for representing statements that involve some causal relations. However, causal literals allow more complex queries about the causal information of ASP literals. How to exploit this information for KR is a major open question. With respect to causal literals themselves, their major limitation is the requirement of monotonicity. In particular, convex causal literals will allow reasoning with necessary and contributory causes.

iv) Going on with KR open questions, an interesting issue is how causal information can be extracted from programs with aggregates, so that we may be able to reason with the causal information of quantitative, and not only qualitative, problems.

v) Another interesting issue consists in defining causal stable models relying on CP-values. CP-values were mainly introduced for a formal unification of our work and why-not provenance justifications. However, the introduction of the negation operator ($\sim$) seems to be a step in the right direction for representing *short-circuit* and *switch* problems from the literature on actual causation.

vi) Finally, the performance of the tool for computing the causal information will be significantly improved by incorporating causal information into the solver, instead of using it as black box.

BIBLIOGRAPHY

Alfred V. Aho, Michae R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.

Mark D. Alicke. Culpable causation. *Journal of personality and social psychology*, 63(3):368, 1992.

Sergei N. Artëmov. Explicit provability and constructive semantics. *Bulletin of Symbolic Logic*, 7(1):1–36, 2001.

Chitta Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, 2003.

Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence*, 93:63–101, 1997.

Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

Krysia Broda, Dov Gabbay, Luis Lamb, and Alessandra Russo. *Compiled Labelled Deductive Systems: A Uniform Presentation of Non-Classical Logics*. Research Studies Press, 2004.

Pedro Cabalar. Causal logic programming. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 102–116. Springer, 2012.

Pedro Cabalar and Jorge Fandinno. An algebra of causal chains. *CoRR*, abs/1312.6134, 2013.

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

Pedro Cabalar, Jorge Fandinno, and Michael Fink. A complexity assessment for queries involving sufficient and necessary causes. In Eduardo Fermé and João Leite, editors, *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, pages 297–310. Springer, 2014b.

Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the Third Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

Fiery Cushman, Joshua Knobe, and Walter Sinnott-Armstrong. Moral appraisals affect doing/allowing judgments. *Cognition*, 108(1):281–289, 2008.

Carlos Viegas Damásio, Anastasia Analyti, and Grigoris Antoniou. Justifications for logic programming. In Pedro Cabalar and Tran Cao Son, editors, *Logic Programming and Nonmonotonic Reasoning, Twelfth International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *Lecture Notes in Computer Science*, pages 530–542. Springer, 2013.

Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3): 374–425, 2001.

Marc Denecker and Danny De Schreye. Justification semantics: A unifiying framework for the semantics of logic programs. In *Logic Programming and Non-monotonic Reasoning, 2nd International Workshop, LPNMR 1993, Lisbon, Portugal, June 1993*, pages 365–379. The MIT Press, 1993.

Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3-4):289–323, 1995.

Thomas Eiter and Thomas Lukasiewicz. Complexity results for structure-based causality. *Artificial Intelligence*, 142(1):53–89, 2002.

François Fages. A new fixpoint semantics for general logic programs compared with the well-founded and the stable model semantics. *New Generation Comput.*, 9(3-4):425–444, 1991.

Paolo Ferraris, Joohyung Lee, Yuliya Lierler, Vladimir Lifschitz, and Fangkai Yang. Representing first-order causal theories by logic programs. *Theory and Practice of Logic Programming TPLP*, 12(3):383–412, 2012.

Martin Gebser, Jörg Pührer, Torsten Schaub, and Hans Tompits. A meta-programming technique for debugging answer-set programs. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 448–453. AAAI Press, 2008.

Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.

Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3-4):365–386, 1991.

Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17(2-4):301–321, 1993.

Michael Gelfond and Yuanlin Zhang. Vicious circle principle and logic programs with aggregates. *Theory and Practice of Logic Programming TPLP*, 14 (4-5):587–601, 2014.

Enrico Giunchiglia and Vladimir Lifschitz. An action language based on causal explanation: Preliminary report. In Jack Mostow and Chuck Rich, editors, *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 623–630. AAAI Press / The MIT Press, 1998.

Joakim Gustafsson and Patrick Doherty. Embracing occlusion in specifying the indirect e ects of actions. In Doyle Aiello and Morgan Kaufmann Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, KR 1996, Cambridge, Massachusetts, USA, November 5-8*, pages 87–98. Morgan Kaufmann Publishers, 1996.

Ned Hall. Causation and the price of transitivity. *The Journal of Philosophy*, 97 (4):198–222, 2000.

Ned Hall. Two concepts of causation. In John Collins, Ned Hall, and L. A. Paul, editors, *Causation and counterfactuals*, pages 225–276. Cambridge, MA: MIT Press, 2004.

Ned Hall. Structural equations and causation. *Philosophical Studies*, 132(1):109–136, 2007.

Ned Hall and L. Paul. Causation and its counterexamples: a traveler's guide. Unpublished manuscript, 2003.

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

Joseph Y. Halpern. Appropriate causal models and stability of causation. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. AAAI Press, 2014.

Joseph Y. Halpern and Christopher Hitchcock. Actual causation and the art of modeling. *CoRR*, abs/1106.2652, 2011.

Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part I: Causes. *Proceedings of the Seventeenth Conference in Uncertainty in Artificial Intelligence, UAI 2001, University of Washington, Seattle, Washington, USA, August 2-5*, pages 194–202, 2001.

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

Steve Hanks and Drew V. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3):379–412, 1987.

Brian A. Haugh. Simple causal minimizations for temporal persistence and projection. In Kenneth D. Forbus and Howard E. Shrobe, editors, *Proceedings of the Sixth National Conference on Artificial Intelligence. Seattle, WA, July 1987.*, pages 218–223. Morgan Kaufmann, 1987.

Eric Hiddleston. Causal powers. *The British journal for the philosophy of science*, 56(1):27–59, 2005.

Christopher Hitchcock and Joshua Knobe. Cause and norm. *Journal of Philosophy*, 11:587–612, 2009.

Kevin D Hoover. The logic of causal inference: Econometrics and the conditional analysis of causation. *Economics and philosophy*, 6(02):207–234, 1990.

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

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

David S. Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. MIT Press, Cambridge, MA, USA, 1990. ISBN 0-444-88071-2.

Daniel Kahneman and Dale T Miller. Norm theory: Comparing reality to its alternatives. *Psychological review*, 93(2):136, 1986.

Daniel Kahneman and Amos Tversky. The simulation heuristic. In P.Slovic D. Kahneman and A. Tversky (Eds.), editors, *Judgement Under Incertainty: Heuristics and Biases*, pages 201–210. Cambridge/New York: Cambridge University Press, 1982.

Henry A. Kautz. The logic of persistence. In Tom Kehler, editor, *Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA, uSA, August 11-15, 1986. Volume 1: Science.*, pages 401–405. Morgan Kaufmann, 1986.

Michael Kifer and Ai Li. On the semantics of rule-based expert systems with uncertainty. In Marc Gyssens, Jan Paredaens, and Dirk Van Gucht, editors, *ICDT'88, 2nd International Conference on Database Theory, Bruges, Belgium, August 31 - September 2, 1988, Proceedings*, volume 326 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 1988. ISBN 3-540-50171-1.

Michael Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12(3-4):335–367, 1992.

Joshua Knobe and Ben Fraser. Causal judgment and moral judgment: Two experiments. *Moral psychology*, 2:441–8, 2008.

Phokion G. Kolaitis and Christos H. Papadimitriou. Why not negation by fixpoint? *Journal of Computer and System Sciences*, 43(1):125–144, 1991.

Arne Konig. pyngo, 2009.

Robert Kowalski and Marek Sergot. A logic-based calculus of events. In *Foundations of knowledge base management*, pages 23–55. Springer, 1989.

Joohyung Lee. Reformulating action language $\mathcal{C}$+ in answer set programming. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning: Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 405–421. Springer, 2012. ISBN 978-3-642-30742-3.

Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3): 499–562, 2006.

Leonid A. Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

David K. Lewis. Causation. *The journal of philosophy*, 70(17):556–567, 1973.

David K. Lewis. Philosophical papers, volume II, 1986.

David K. Lewis. Causation as influence. *The Journal of Philosophy*, 97(4):182–197, 2000.

Vladimir Lifschitz. Formal theories of action (preliminary report). In John P. McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence. Milan, Italy, August 1987*, pages 966–972. Morgan Kaufmann, 1987.

Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1-2):39–54, 2002.

Vladimir Lifschitz. Thirteen definitions of a stable model. In Andreas Blass, Nachum Dershowitz, and Wolfgang Reisig, editors, *Fields of Logic and Computation, Essays Dedicated to Yuri Gurevich on the Occasion of His Seventieth Birthday*, volume 6300 of *Lecture Notes in Computer Science*, pages 488–503. Springer, 2010. ISBN 978-3-642-15024-1.

Vladimir Lifschitz. The dramatic true story of the frame default. *Journal of Philosophical Logic*, 44(2):163–176, 2015.

Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Logic Programming, Proceedings of the Eleventh International Conference on Logic Programming, Santa Marherita Ligure, Italy, June 13-18, 1994*, pages 23–37. MIT Press, 1994. ISBN 0-262-72022-1.

Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Nonmonotonic Reasoning, Fifth International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings*, volume 1730 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 1999. ISBN 3-540-66749-0.

Fangzhen Lin. Embracing causality in specifying the indirect effects of actions. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, pages 1985–1993. Morgan Kaufmann, 1995.

Fangzhen Lin and Mikhail Soutchanski. Causal theories of actions revisited. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

John L. Mackie. Causes and conditions. *American Philosophical Quarterly*, 2(4): 245–264, 1965.

David R. Mandel, Denis J. Hilton, and Patrizia Catellani. *The psychology of counterfactual thinking*. Routledge, 1985.

V. Wiktor Marek and Miroslaw Truszczynski. Autoepistemic logic. *Journal of the ACM (JACM)*, 38(3):587–618, 1991.

Victor W. Marek and Miroslaw Truszczyńki. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Mirosław Truszczyński, and DavidS Warren, editors, *The Logic Programming Paradigm*, Artificial Intelligence, pages 375–398. Springer Berlin Heidelberg, 1999. ISBN 978-3-642-64249-4.

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

Norman McCain and Hudson Turner. Causal theories of action and change. In Benjamin Kuipers and Bonnie L. Webber, editors, *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island.*, pages 460–465. AAAI Press / The MIT Press, 1997. ISBN 0-262-51095-2.

Norman Clayton McCain. *Causality in commonsense reasoning about actions*. PhD thesis, University of Texas at Austin, 1997.

John McCarthy. Programs with common sense, 1959.

John McCarthy. Circumscription - A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39, 1980.

John McCarthy. Elaboration tolerance. In *Proceedings of the Fourth Symposium on Logical Formalizations of Commonsense Reasoning, Common Sense 98*, pages 198–217, London, UK, 1998.

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

Drew McDermott and Jon Doyle. Non-monotonic logic I. *Artificial intelligence*, 13(1):41–72, 1980.

John Stuart Mill. Systems of logic, 1843. Reprinted by University Press of the Pacific, Honolulu, 2002.

Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25 (3-4):241–273, 1999.

Ilkka Niemelä, Patrik Simons, and Tommi Syrjänen. Smodels: A system for Answer Set Programming. *CoRR*, cs.AI/0003033, 2000.

Christos H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

David Pearce. A new logical characterisation of stable models and answer sets. In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, editors, *Non-Monotonic Extensions of Logic Programming, NMELP 1996, Bad Honnef, Germany, September 5-6, 1996, Selected Papers*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70. Springer, 1996.

Judea Pearl. Embracing causality in default reasoning. *Artificial Intelligence*, 35 (2):259–271, 1988.

Judea Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, New York, NY, USA, 2000. ISBN 0-521-77362-8.

Luís Moniz Pereira, Joaquim Nunes Aparício, and José Júlio Alferes. Derivation procedures for extended stable models. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, pages 863–869. Morgan Kaufmann, 1991.

Enrico Pontelli, Tran Cao Son, and Omar El-Khatib. Justifications for logic programs under answer set semantics. *Theory and Practice of Logic Programming TPLP*, 9(1):1–56, 2009.

Raymond Reiter. A logic for default reasoning. *Artificial Intelligence*, 13(1-2): 81–132, 1980.

Raymond Reiter. Nonmonotonic reasoning. *Annual review of computer science*, 2 (1):147–186, 1987.

Neal J Roese. Counterfactual thinking. *Psychological bulletin*, 121(1):133, 1997.

Kenneth J Rothman. Causes. *American Journal of Epidemiology*, 104(6):587–592, 1976.

Kenneth J Rothman and Sander Greenland. *Modern epidemiology*. Philadelphia: Lippincott-Raven, 2 edition, 1998.

Erik Sandewall. Assessments of ramification methods that use static domain constraints. In Luigia Carlucci Aiello, Jon Doyle, and Stuart C. Shapiro, editors, *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning, KR 1996, Cambridge, Massachusetts, USA, November 5-8, 1996*, pages 99–110. Morgan Kaufmann, 1996.

Marcus Schaefer and Christopher Umans. Completeness in the polynomial-time hierarchy: A compendium. *SIGACT news*, 33(3):32–49, 2002.

John S. Schlipf. The expressive powers of the logic programming semantics. *Journal of Computer and System Sciences*, 51(1):64–86, 1995.

Claudia Schulz and Francesca Toni. Aba-based answer set justification. *Theory and Practice of Logic Programming TPLP*, 13(4-5 Online-Supplement), 2013.

Claudia Schulz and Francesca Toni. Justifying answer sets using argumentation. *CoRR*, abs/1411.5635, 2014.

Murray Shanahan. The ramification problem in the event calculus. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 1999, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 140–146. Morgan Kaufmann, 1999.

Gerd Stumme. Free distributive completions of partial complete lattices. *Order*, 14:179–189, 1997. ISSN 0167-8094.

Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

Michael Thielscher. Ramification and causality. *Artificial Intelligence*, 89(1-2): 317–364, 1997.

Hudson Turner. Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming*, 31(1-3):245–298, 1997.

Kristof Van Belleghem, Marc Denecker, and Daniele Theseider Dupré. A constructive approach to the ramification problem. In *Proceedings of ESSLLI*, pages 1–17. Citeseer, 1998.

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

Allen Van Gelder. The alternating fixpoint of logic programs with negation. In Avi Silberschatz, editor, *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 29-31, 1989, Philadelphia, Pennsylvania, USA*, pages 1–10. ACM Press, 1989.

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. Unfounded sets and well-founded semantics for general logic programs. In Chris Edmondson-Yurkanan and Mihalis Yannakakis, editors, *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, March 21-23, 1988, Austin, Texas, USA*, pages 221–230. ACM, 1988.

Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):620–650, 1991.

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

Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

Richard W Wright. Causation, responsibility, risk, probability, naked statistics, and proof: Pruning the bramble bush by clarifying the concepts. *The University of Iowa Law Review*, 73:1001, 1988.

A PROOFS

This appendix addresses the formal proofs of the results showed along the dissertation.

CHAPTER 3: SUFFICIENT CAUSES

Proof of Proposition 3.1

Lemma 1. *Given a completely labelled program P and proofs $\pi(A)$ and $\pi'(A)$, it holds that $graph(\pi) \subseteq graph(\pi')$ iff $cgraph(\pi) \subseteq cgraph(\pi')$.* $\square$

Proof. For the only if direction, just note that, by definition, $cgraph(\pi)$ is the transitive and reflexive closure of $graph(\pi)$. So that, assume that $cgraph(\pi) \subseteq cgraph(\pi')$ and suppose there is an edge in $graph(\pi) \backslash graph(\pi')$. Note that such edge must be in $cgraph(\pi) \subseteq cgraph(\pi')$. Note also that such edge must be of the form (l_R, l_A) or (l_B, l_R) for some rule R, and since P is completely labelled, $graph(\pi')$ must contain both edges (l_R, l_A) of (l_B, l_R) for every proof containing the rule l_R. $\square$

Lemma 2. *Given a completely labelled program P and proofs $\pi(A)$ and $\pi'(A)$, it holds that $graph(\pi) \subset graph(\pi')$ iff $cgraph(\pi) \subset cgraph(\pi')$.* $\square$

Proof. By Lemma 1, $graph(\pi) \subseteq graph(\pi')$ iff $cgraph(\pi) \subseteq cgraph(\pi')$ which is equivalent to $graph(\pi) \not\supseteq graph(\pi')$ iff $cgraph(\pi) \not\supseteq cgraph(\pi')$. Therefore, $graph(\pi) \subset graph(\pi')$ if and only if $graph(\pi) \subseteq graph(\pi')$ and $graph(\pi) \not\supseteq graph(\pi')$ if and only if $cgraph(\pi) \subseteq cgraph(\pi')$ and $cgraph(\pi) \not\supseteq cgraph(\pi')$ if and only if $cgraph(\pi) \subset cgraph(\pi')$. $\square$

Lemma 3. *Given a completely program P and a pair of proofs $\pi(A)$ and $\pi'(A)$, if $subproofs(\pi') \subseteq subproofs(\pi)$, then $graph(\pi') \subseteq graph(\pi)$.* □

Proof. Note that, by definition, the edges of $graph(\pi')$ are of the form (l_R, l_A) or (l_B, l_R) for some subproof π'' of π' of the form (50). Note also that $\pi'' \in subproofs(\pi') \subseteq subproofs(\pi)$ implies that those edges are in $graph(\pi)$ as well. Hence $graph(\pi') \subseteq graph(\pi')$. □

Lemma 4. *Given a completely program P and a pair of proofs $\pi(A)$ and $\pi'(A)$, if $subproofs(\pi') \subset subproofs(\pi)$, then $graph(\pi') \subset graph(\pi)$.* □

Proof. Assume as induction hypothesis that

$$subproofs(\pi') \subset subproofs(\pi'') \text{ implies } graph(\pi') \subset graph(\pi'')$$

for every proof π'' of the same atom than π which furthermore holds that $subproofs(\pi'') \subset subproofs(\pi)$.

If π has no subproofs then the antecedent holds vacuous. Otherwise, by hypothesis it holds that $subproofs(\pi') \subset subproofs(\pi)$, and this implies $subproofs(\pi') \subseteq subproofs(\pi)$ which, in its turn, implies, by Lemma 3, that $graph(\pi') \subseteq graph(\pi)$.

Suppose that $graph(\pi') = graph(\pi)$ and let π_s a sub-proof of π which is not a sub-proof of π' and let R and A be respectively the rule and the consequent of π_s. Then the edge $(l_R, l_A) \in graph(\pi) = graph(\pi')$, and hence, there exists a sub-proof π_t of π' with the same rule R and the same consequent A but with a different direct subproofs.

Note that $\{\pi_s, \pi_t\} \subseteq subproofs(\pi)$ and let π'' be the result of replacing in π each occurrence of π_s by π_t. Note that $subproofs(\pi'') = subproofs(\pi) \backslash \{\pi_s\}$, and then $subproofs(\pi') \subset subproofs(\pi'')$ because $\pi_s \notin subproofs(\pi')$. Hence, by induction hypothesis, $graph(\pi') \subset graph(\pi'')$,

Furthermore, by Lemma 3 again, $subproofs(\pi'') \subseteq subproofs(\pi)$ implies the fact $graph(\pi'') \subseteq graph(\pi)$. Hence, $graph(\pi') \subset graph(\pi'') \subseteq graph(\pi)$. □

Lemma 5. *Given a completely program P and a pair of proofs $\pi(A)$ and $\pi'(A)$, if $subproofs(\pi') \subset subproofs(\pi)$, then $graph(\pi') \subset graph(\pi)$.* □

Proof. By Lemma 4, $subproofs(\pi') \subset subproofs(\pi)$ implies the fact $graph(\pi') \subset graph(\pi)$ which, by Lemma 2, implies $cgraph(\pi') \subset cgraph(\pi)$. □

Lemma 6. *Given a completely program P, a non-redundant proof $\pi(A)$ and a proof $\pi'(A)$ s.t. $graph(\pi') \subseteq graph(\pi)$, then $subproofs(\pi') \subseteq subproofs(\pi)$.* □

Proof. Suppose there is a proof $\pi'' \in subproofs(\pi')$ but $\pi'' \notin subproofs(\pi)$. If π'' is of the form

$$\pi'' \quad = \quad \frac{\top}{A}\,(R),$$

then the edge $(l_R, l_A) \in graph(\pi'') \subseteq graph(\pi') \subseteq graph(\pi)$, and therefore, $\pi'' \in subproofs(\pi)$ which is a contradiction.

Assume as induction hypothesis that all subproofs of π'' but for itself are subproofs of π. Since the edge $(l_R, l_A) \in graph(\pi'') \subseteq graph(\pi') \subseteq graph(\pi)$, there is a subproof of π which is like π'' but with some different direct subproof $\pi_s(B)$ for some literal B in the body of R. Let $\pi_t(B)$ the direct subproof of π'' for the literal B. Since, by induction hypothesis, every subproof of π'' but for itself are a subproof of π, then there are two different subproof of π, for the literal B: $\pi_s(B)$ and $\pi_t(B)$. Let π''' be the result of replacing, in π, each occurrence of the subproof $\pi_s(B)$ by $\pi_t(B)$. Then $subproofs(\pi''') = subproofs(\pi) \backslash \{\pi_s(B)\}$, that is $subproofs(\pi''') \subset subproofs(\pi)$ which contradicts the assumption that π is non-redundant. □

Lemma 7. *Given a completely program P and a non-redundant proof $\pi(A)$, there not exists a proof $\pi'(A)$ such that $graph(\pi') \subset graph(\pi)$.* □

Proof. Note that, $graph(\pi') \subset graph(\pi)$ implies $graph(\pi') \subseteq graph(\pi)$, and then, by Lemma 6, it follows that $subproofs(\pi') \subseteq subproofs(\pi)$.

Furthermore, since $graph(\pi') \subset graph(\pi)$, there is an edge of the form (l_R, l_A) or (l_B, l_R) in $graph(\pi)$ which is not in $graph(\pi')$, and so, there is a proof π'', whose rule is R, which is a subproof of π but not of π', that is, $subproofs(\pi') \subset subproofs(\pi)$, that is π is redundant. □

Lemma 8. *Given a completely program P and a non-redundant proof $\pi(A)$, there not exists a proof $\pi'(A)$ such that $cgraph(\pi') \subset cgraph(\pi)$.* □

Proof. By Lemma 2, $cgraph(\pi') \subset cgraph(\pi)$ iff $graph(\pi') \subset graph(\pi)$ and, by Lemma 7, there not exists a proof $\pi'(A)$ s.t. $graph(\pi') \subset graph(\pi)$. □

Proof of Proposition 3.1. For the only if direction, by definition, $\pi(A)$ is redundant iff there exist some proof $\pi'(A)$ s.t. $subproofs(\pi') \subset subproofs(\pi)$, which, by Lemma 5, implies $cgraph(\pi') \subset cgraph(\pi)$, that is π is graph-redundant.

For the if direction, Suppose that $\pi(A)$ is non-redundant. By Lemma 8, there not exists a proof $\pi'(A)$ such that $cgraph(\pi') \subset cgraph(\pi)$ which is a contradiction with the assumption that $\pi(A)$ is graph-redundant, that is, there exist some proof $\pi'(A)$ s.t. $cgraph(\pi') \subset cgraph(\pi)$. □

Proof of Proposition 3.2

Proof of Proposition 3.2.

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

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

$$\begin{aligned} a \cdot b * b \cdot c * a \cdot c &= (a \cdot b * b \cdot c) * a \cdot c && \text{(associative } (*)) \\ &= (a \cdot b \cdot c) * a \cdot c && \text{(transitivity)} \\ &= a \cdot (b \cdot c) * a \cdot c && \text{(associative } (\cdot)) \\ &= a \cdot (b \cdot c * c) && \text{(distributivity)} \\ &= a \cdot (b \cdot c) && \text{(absorption der)} \\ &= a \cdot b \cdot c && \text{(associativity)} \\ &= a \cdot b * b \cdot c && \text{(transitivity)} \end{aligned}$$

Proof of Proposition 3.3.

- Product ($*$) is associative. That is, given three causal graphs G_1, G_2 and G_3 they hold that $G_1 * (G_2 * G_3) \ = \ (G_1 * G_2) * G_3$.

$$\begin{aligned}(G_1 * G_2) * G_3 &= \left((G_1 \cup G_2)^* \cup G_3\right)^* \\ &= \left(G_1 \cup G_2 \cup G_3\right)^* \\ &= \left(G_1 \cup (G_2 \cup G_3)^*\right)^* \\ &= \left(G_1 \cup (G_2 \cup G_3)^*\right)^* \\ &= G_1 * (G_2 * G_3)\end{aligned}$$

- Application ($\cdot$) is associative. That is, given three causal graphs G_1, G_2 and G_3 they hold that $G_1 \cdot (G_2 \cdot G_3) \ = \ (G_1 \cdot G_2) \cdot G_3$. By definition

$$\begin{aligned}(G_1 \cdot G_2) \cdot G_3 &= \left((G_1 \cup G_2 \cup G_{12})^* \cup G_3 \cup G_{12,3}\right)^* \\ &= \left(G_1 \cup G_2 \cup G_3 \cup G_{12} \cup G_{12,3}\right)^* \qquad (108)\end{aligned}$$

$$\begin{aligned}G_1 \cdot (G_2 \cdot G_3) &= \left(G_1 \cup \left(G_2 \cup G_3 \cup G_{23}\right)^* \cup G_{1,23}\right)^* \\ &= \left(G_1 \cup G_2 \cup G_3 \cup G_{1,23} \cup G_{23}\right)^* \qquad (109)\end{aligned}$$

where

$$\begin{aligned}G_{12} &\overset{\text{def}}{=} \{ \ (v_1,v_2) \mid v_1 \in G_1 \text{ and } v_2 \in G_2 \ \} \\ G_{12,3} &\overset{\text{def}}{=} \{ \ (v_{12},v_3) \mid v_{12} \in G_1 \cup G_2 \text{ and } v_3 \in G_3 \ \} \\ G_{23} &\overset{\text{def}}{=} \{ \ (v_2,v_3) \mid v_2 \in G_2 \text{ and } v_3 \in G_3 \ \} \\ G_{1,23} &\overset{\text{def}}{=} \{ \ (v_1,v_{23}) \mid v_1 \in G_1 \text{ and } v_{23} \in G_2 \cup G_3 \ \}\end{aligned}$$

From (108) and (109), it follows that $(G_1 \cdot G_2) \cdot G_3 = G_1 \cdot (G_2 \cdot G_3)$ holds if $G_{12} \cup G_{12,3} = G_{1,23} \cup G_{23}$ if $G_{12,3} \subseteq G_{1,23} \cup G_{23}$ and $G_{1,23} \subseteq G_{12} \cup G_{12,3}$. Note that $G_{12} \subseteq G_{1,23}$ and $G_{23} \subseteq G_{12,3}$.

We will show that $G_{12,3} \subseteq G_{1,23} \cup G_{23}$. Suppose an edge $(v_{12},v_3) \in G_{12,3}$ such that $(v_{12},v_3) \notin G_{23}$ and $(v_{12},v_3) \notin G_{1,23}$. Note that $v_{12} \in G_1 \cup G_2$ and $v_3 \in G_3$ implies that either $v_{12} \in G_1$ or $v_{12} \in G_2$. If $v_{12} \in G_1$, then $(v_{12},v_3) \in G_{1,23}$ which is a contradiction, and if, otherwise, $v_{12} \in G_2$, then $(v_{12},v_3) \in G_{23}$ which is also a contradiction. The case for $G_{1,23} \subseteq G_{12} \cup G_{12,3}$ is symmetric.

- The empty graph is the identity of product $(*)$. That is $G * \emptyset = G$. Just note that, by definition $G * \emptyset = (G \cup \emptyset)^* = G^*$. And since, G is a causal graphs, it is already close, that is $G = G^*$.
- The empty graph is the identity of application $(\cdot)$ That is $G \cdot \emptyset = G$. Mote that, by definition
$$G \cdot \emptyset = (G \cup \emptyset \cup \{ (v_1, v_2) \mid v_1 \in G \text{ and } v_2 \in \emptyset \})^* = G^* = G$$
- Product $(*)$ is commutative. That is $G * G' = G' * G$. Just note that by definition $G * G' = (G \cup G')^* = (G' \cup G)^* = G' * G$.
- Product $(*)$ is idempotent. That is $G * G = G$. Just note that by definition $G * G = (G \cup G)^* = (G)^* = G$.
- Application $(\cdot)$ is idempotent with respect to atomic causal graphs. Let $G = \{ (v, v) \}$. Then $G \cdot G = (G \cup G \cup \{ (v_1, v_2) \mid v_1 \in G \text{ and } v_2 \in G \})^*$ Thus $G \cdot G = (G \cup G \cup \{ (v, v) \})^* = G^* = G$.
- Product $(*)$ is monotonic. Assume that $G_1 \subseteq G_2$ and suppose also that $G_1 * G_3 \not\subseteq G_2 * G_3$ Pick $(v_1, v_n) \in (G_1 * G_3) \backslash (G_2 * G_3)$ By definition, it holds $(G_1 * G_3) = (G_1 \cup G_3)^*$, it follows that there are a sequence of edges $\{ (v_1, v_2), \ldots, (v_{n-1}, v_n) \} \subseteq G_1 \cup G_3$. Note that each edge (v_i, v_{i+1}) is either in G_1 or in G_3, and so, it is eider in $G_2 \supseteq G_1$ or G_2, that is $(v_i, v_{i+1}) \in G_2 \cup G_3$. Hence $\{ (v_1, v_2), \ldots, (v_{n-1}, v_n) \} \subseteq G_2 \cup G_3$, and therefore $(v_1, v_n) \in (G_2 \cup G_3)^* = G_2 * G_3$ which is a contradiction.
- Application $(\cdot)$ is monotonic. Assume that $G_1 \subseteq G_2$ and suppose that $G_1 \cdot G_3 \not\subseteq G_2 \cdot G_3$ Pick $(v_1, v_n) \in (G_1 \cdot G_3) \backslash (G_2 \cdot G_3)$ By definition, it holds $(G_1 \cdot G_3) = (G_1 \cup G_3 \cup G_{13})^*$, where
$$G_{13} = \{ (v_1, v_3) \mid v_1 \in G_1 \text{ and } v_3 \in G_3 \}$$
Then, there are a sequence of edges $\{ (v_1, v_2), \ldots, (v_{n-1}, v_n) \} \subseteq G_1 \cup G_3 \cup G_{13}$. If $(v_i, v_{i+1}) \in G_1 \cup G_3$, then, as above $(v_i, v_{i+1}) \in G_2 \cup G_3$. So that assume that $(v_i, v_{i+1}) \in G_{13}$. Then $v_i \in G_1$, and then $v_i \in G_2$, and $v_{i+1} \in G_3$, so that $(v_i, v_{i+1}) \in G_2 \cdot G_3$. Hence $\{ (v_1, v_2), \ldots, (v_{n-1}, v_n) \} \subseteq G_2 \cdot G_3$, and therefore $(v_1, v_n) \in (G_2 \cdot G_3)^* = G_2 \cdot G_3$ which is a contradiction. □

Proof of Proposition 3.4

Proof of Proposition 3.4.

- Application $(\cdot)$ distributes over products $(*)$. For every pair of non-empty sets of causal graphs S and S_2, it holds that

  $$(\textstyle\prod S) \cdot (\prod S_2) = \prod \{\ G_1 \cdot G_2 \mid G_1 \in S \text{ and } G_2 \in S_2\ \}.$$

  We define the following names

  $$\begin{aligned} G_r &\stackrel{\text{def}}{=} (\textstyle\prod S_1) \cdot (\prod S_2) \\ G_l &\stackrel{\text{def}}{=} \textstyle\prod \{\ G_1 \cdot G_2 \mid G_1 \in S_2 \text{ and } G_2 \in S_2\ \} \end{aligned}$$

  Then, by product definition, it follows that

  $$\begin{aligned} G_l &= \Big(\bigcup \{\ G_1 \mid G_1 \in S_1\ \} \cup \bigcup \{\ G_2 \mid G_2 \in S_2\ \} \cup G_l' \Big)^* \\ G_r &= \Big(\bigcup \{\ G_1 \cup G_2 \cup G_{12} \mid G_1 \in S_2 \text{ and } G_2 \in S_2\ \} \Big)^* \end{aligned}$$

  where

  $$\begin{aligned} G_l' &= \{\ (v_1,v_2) \mid v_1 \in \bigcup \{\ G_1 \mid G_1 \in S_1\ \} \text{ and } v_2 \in \bigcup \{\ G_2 \mid G_2 \in S_2\ \}\ \} \\ G_{12} &= \{\ (v_1,v_2) \mid v_1 \in G_1 \text{ and } v_2 \in G_2\ \} \end{aligned}$$

  Let $G_r' = \bigcup \{\ G_{12} \mid G_1 \in S \text{ and } G_2 \in S_2\ \}$. For every edge $(v_1,v_2) \in G_l'$ there are a pair of c-graphs $G_1 \in S_1$ and $G_2 \in S_2$ s.t. $v_1 \in G_1$ and $v_2 \in G_2$) and then $(v_1,v_2) \in G_{12}$ and consequently $(v_1,v_2) \in G_r'$.

  Moreover, for every edge $(v_1,v_2) \in G_r'$ there are a pair of c-graphs $G_1 \in S_1$ and $G_2 \in S_2$ s.t. $(v_1,v_2) \in G_{12}$ with $l_1 \in G_1$ and $v_2 \in G_2$. So that $(v_1,v_2) \in G_l'$. That is $G_l' = G_r'$. Then

  $$\begin{aligned} G_r &= \Big(\bigcup \{\ G_1 \mid G_1 \in S_1\ \} \cup \bigcup \{\ G_2 \mid G_2 \in S_2\ \} \cup G_r' \Big)^* \\ &= (G_l \backslash G_l' \cup G_r')^* = (G_l)^* = G_l \end{aligned}$$

  Consequently $G_l = G_r$.

- Application $(\cdot)$ holds the absorption equation. Note that, by definition, it is clear that $G_1 \cdot G_2 \cdot G_3 \supseteq G_2$ and then

  $$\begin{aligned} G_2 * G_1 \cdot G_2 \cdot G_3 &= (G_2 \cup G_1 \cdot G_2 \cdot G_3)^* \\ &= (G_1 \cdot G_2 \cdot G_3)^* = G_1 \cdot G_2 \cdot G_3 \end{aligned}$$

- Application $(\cdot)$ holds the transitive equation. That is, for any causal graphs G_1, $G_2 \neq \emptyset$ and G_3, it holds that $G_1 \cdot G_2 \cdot G_3 = G_1 \cdot G_2 * G_2 \cdot G_3$

It is clear that $G_1 \cdot G_2 \cdot G_3 \supseteq G_1 \cdot G_2$ and $G_1 \cdot G_2 \cdot G_3 \supseteq G_2 \cdot G_3$ and then $G_1 \cdot G_2 \cdot G_3 \supseteq G_1 \cdot G_2 * G_2 \cdot G_3$.

Let G_{12}, G_{23}, G_l and G_r be respectively $G_{12} = G_1 \cdot G_2$, $G_{23} = G_2 \cdot G_3$, $G_l = G_{12} \cdot G_3 = G_1 \cdot G_2 \cdot G_3$ and $G_r = G_{12} * G_{23}$.

Suppose there is an edge $(v_1, v_n) \in G_l \backslash G_r$. Then there is a sequence of edges $\{ (v_1, v_2), \ldots, (v_{n-1}, v_n) \} \subseteq G_{12} \cup G_3 \cup G_{12,3}$. For each (v_i, v_{i+1}), if $(v_i, v_{i+1}) \in G_{12} \subseteq G_r$ or $(v_i, v_{i+1}) \in G_3 \subseteq G_{23} \subseteq G_r$, then $(v_i, v_{i+1}) \in G_r$. If $(v_i, v_{i+1}) \in G_{12,3}$, then $v_1 \in G_{12}$ and $v_2 \in G_3$, and then one of the following holds

- If $v_i \in G_1$ and $v_{i+1} \in G_3$, then, since $G_2 \neq \emptyset$, there is $v \in G_3$, and therefore $(v_i, v) \in G_{12} \subseteq G_r$ and $(v, v_{i+1}) \in G_{23} \subseteq G_r$. Furthermore, since G_r is closed transitively, $(v_i, v_{i+1}) \in G_r$
- If $v_i \in G_2$ and $v_{i+1} \in G_3$, then $(v_i, v_{i+1}) \in G_{23} \subseteq G_r$

That is $(v_i, v_{i+1}) \in G_r$ for all i, and therefore, $(v_1, v_n) \in G_r$ which is a contradiction. That is, $G_l = G_1 \cdot G_2 \cdot G_3 = G_1 \cdot G_2 * G_2 \cdot G_3 = G_r$. □

Proof of Proposition 3.5

Proof of Proposition 3.5. Let us name as G the causal graphs associated to the expression $(cgraph(\pi_1) * \ldots * cgraph(\pi_m)) \cdot l_R \cdot l_A$.

By definition $cgraph(\pi)$ is the reflexive and transitive closure of $graph(\pi)$ and, in its turn, $graph(\pi)$ is given by

$$\{ graph(\pi_i) \mid 1 \leq i \leq m \} \cup \{ (l_{B_i}, l_R) \mid 1 \leq i \leq m \} \cup \{ (l_R, l_A) \}$$

It is easy to see that all edge in $graph(\pi_i)$ and also the edge (l_R, l_A) are in G. Similarly, the edges of the form (l_{B_i}, l_R) are in G because l_{B_i} belongs to $cgraph(\pi_i)$, and so, (l_{B_i}, l_R) to the result of apply the rule l_R. That is $graph(\pi) \subseteq G$, and hence also $cgraph(\pi) \subseteq G$.

The other way around. We proceed by structural induction assuming that $cgraph(\pi_i) \subseteq G$ for every direct sub-proof π_i and, furthermore, for every label $l \in cgraph(\pi_i)$ there is an edge $(l, l_{B_i}) \in cgraph(\pi_i)$ where B_i the consequent of π_i and l_{B_i} its associated label.

Suppose that $cgraph(\pi) \supset G$ and pick an edge G which is not an edge of $cgraph(\pi)$. By induction hypothesis, if the edge corresponds to a a sub-proof of π then it belongs to $cgraph(\pi_i)$ for some direct sub-proof π_i, and so, to G which is a contradiction. Furthermore, it is easy to see that the edge $(l_R, l_A) \in graph(\pi) \subseteq cgraph(\pi)$.

Then, such edge must be of the form (v_1, v_2) where $v_1 \in cgraph(\pi_i)$ for some π_i and $v_2 \in \{l_R, l_A\}$. By induction hypothesis, there is some edge $(v_1, l_{B_i}) \in graph(\pi_i) \subseteq graph(\pi)$. Note that $(l_{B_i}, l_R) \in graph(\pi)$ and, consequently, it follows that $(v_1, l_R) \in cgraph(\pi)$ because $cgraph(\pi)$ is transitively close. Similarly it follows that $(v_1, l_A) \in cgraph(\pi)$. □

Proof of Proposition 3.6

Proof of Proposition 3.6.

1. Product is the greatest lower bound of the sufficient $\leq$ relation. By definition $\prod_{G \in U} G = \left(\bigcup_{G \in U} G\right)^* \supseteq \bigcup_{G \in U} G \supseteq G$ for all $G \in U$. Note that, by definition $G \leq G'$ iff $G \supseteq G'$, so that $\prod_{G \in U} G \leq G$ for all $G \in U$. Furthermore, let G' the greatest lower bound of $\leq$, then $G' \supseteq G$ for all $G \in U$, and hence, $G' \supseteq \bigcup_{G \in U} G$ which implies $(G')^* \supseteq \left(\bigcup_{G \in U} G\right)^* = \prod_{G \in U} G$ Since G' is a causal graphs it is transitively and reflexively close, so that $G' = (G')^* = \prod_{G \in U} G$.
2. for every pair of causal graphs G and G' it holds that $G \leq G'$ if and only if $G * G' = G$. Just note that $G * G'$ is the greatest lower bound of $\leq$ and G is the a lower bound of both G and G' and any other lower bound have to be a lower bound of G, so that $G * G' = G$.
3. Both product and application are monotonic operations with respect to the sufficient $\leq$ relation. Note that, by Proposition 3.3 they are monotonic with respect to the subgraph relation and $G \leq G'$ iff $G \supseteq G'$, so that, they are monotonic with respect to the $\leq$ order relation.
4. for every pair of causal graphs G and G' it holds that $G \cdot G' \leq G * G'$. Note that by definition $G \cdot G' = (G \odot G')^*$ and $G * G' = (G \cup G')^*$ and $G \odot G' \supseteq G \cup G'$, so that, $G \cdot G' \supseteq G * G*$ and this, by definition, implies $G \cdot G' \leq G * G'$. □

Proof of Proposition 3.8

Proof of Proposition 3.8.

It is clear that $\downarrow$ is subjective since $\Downarrow\mathbf{C}_{Lb}$ is the image of $\downarrow$. Furthermore, ff $\downarrow G_1 = \downarrow G_2$, that is, $\{G' \mid G' \leq G_1\} = \{G' \mid G' \leq G_2\}$, then $G_2 \in \downarrow G_1$ and $G_1 \in \downarrow G_2$, and then $G_2 \leq G_1$ and $G_1 \leq G_2$, that is $G_1 = G_2$. Hence $\downarrow$ is bijective.

We show now that it preserves the product.

$$\downarrow \prod_{G \in U} G = \{G' \mid G' \leq G \text{ for all } G \in U\} = \bigcap_{G \in U} \{G' \mid G' \leq G\} = \prod_{G \in U} \downarrow G$$

Furthermore, since by definition, $G_1 \leq G_2$ iff $G_1 * G_2 = G_1$ and $\downarrow G_1 \leq \downarrow G_2$ iff $\downarrow G_1 \subseteq \downarrow G_2$ iff $\downarrow G_1 \cap \downarrow G_2 = \downarrow G_1$ iff $\downarrow G_1 * \downarrow G_2 = \downarrow G_1$, then $\downarrow$ also preserves the $\leq$ relation. □

Proof of Proposition 3.9

Proof of Proposition 3.9.

1. Application is associative. Let T, U and W be three causal values. By definition it follows that

$$\begin{aligned}(U \cdot T) \cdot W &= \downarrow \{\, G_U \cdot G_T \mid G_U \in U \text{ and } G_T \in T \,\} \cdot W \\ &= \downarrow \{\, G' \cdot G_W \mid G_U \in U,\ G_T \in T,\ G' \leq G_U \cdot G_T \text{ and } G_W \in W \,\} \\ &= \downarrow \{\, (G_U \cdot G_T) \cdot G_W \mid G_U \in U,\ G_T \in T \text{ and } G_W \in W \,\}\end{aligned}$$

   In the same way, it also follows that

$$U \cdot (T \cdot W) = \downarrow \{\, G_U \cdot (G_T \cdot G_W) \mid G_U \in U,\ G_T \in T \text{ and } G_W \in W \,\}$$

   Then it is enough to show that $(G_U \cdot G_T) \cdot G_W = G_U \cdot (G_T \cdot G_w) = G_U \cdot G_T \cdot G_W$ which holds due to Proposition 3.3. Then, it holds that $U \cdot (T \cdot W) = (U \cdot T) \cdot W = U \cdot T \cdot W$.

2. Addition distributive. By definition, it follows that

$$\begin{aligned}(U \cdot T) + (U \cdot W) &= (U \cdot T) \cup (U \cdot W) \\ &= \downarrow \{\, G_U \cdot G_T \;\big|\; G_U \in U \text{ and } G_T \in T \,\} \\ &\qquad \cup \downarrow \{\, G_U \cdot G_T \;\big|\; G_U \in U \text{ and } G_W \in W \,\} \\ &= \downarrow \{\, G_U \cdot G' \;\big|\; G_U \in U \text{ and } G' \in T \cup W \,\} \\ &= U \cdot (T \cup W) = U \cdot (T + W)\end{aligned}$$

   Furthermore $(U + T) \cdot W = (U \cdot W) + (T \cdot W)$ holds symmetrically.

3. Absorption. As we have seen above,

$$U \cdot T \cdot W = \downarrow\left\{\, G_U \cdot G_T \cdot G_W \;\middle|\; G_U \in U,\, G_T \in T \text{ and } G_W \in W \,\right\}$$

Furthermore, for every c-graph $G_T \in T$, it holds that $G_U \cdot G_T \cdot T_W \leq G_T$. Then, since T is an ideal, it follows that $G_U \cdot G_T \cdot T_W \in T$ and consequently $U \cdot T \cdot W \subseteq T$. Thus $U \cdot T \cdot W \cup T = T$. Similarly, it also holds that $U \cdot T \cdot W \cap T = U \cdot T \cdot W$. Then, by definition, these equalities can be rewritten as $U \cdot T \cdot W + T = T$ and $U \cdot T \cdot W * T = U \cdot T \cdot W$.

4. Identity and Annihilator follow directly from the definition of 1 and 0 respectively as $\mathbf{C}_{Lb}$ and $\emptyset$. $\square$

Proof of Proposition 3.10

Proof of Proposition 3.10.

Note that, by definition products ($*$) are ideal intersections ($\cap$) and $\leq$ stands for the subgraph relation $\subseteq$. Hence, the preservation of products and the $\leq$ relation directly follows from Proposition 3.8

It remains to show that $\downarrow$ preserves applications. Take any c-graphs G_1 and G_2, then

$$\begin{aligned} \downarrow G_1 \cdot \downarrow G_2 &= \Downarrow\{\, (G_1' \cdot G_2') \mid G_1' \in \downarrow G_1 \text{ and } G_2' \in \downarrow G_2 \,\} \\ &= \Downarrow\{\, (G_1' \cdot G_2') \mid G_1' \leq G_1 \text{ and } G_2' \leq G_2 \,\} \end{aligned}$$

Recall that $(\cdot)$ is monotonic with respect to '$\leq$', so that $G_1' \cdot G_2' \leq G_1 \cdot G_2$. Hence

$$\downarrow G_1 \cdot \downarrow G_2 \;=\; \Downarrow\{\, (G_1' \cdot G_2') \mid G_1' \leq G_1 \text{ and } G_2' \leq G_2 \,\} \;\subseteq\; \downarrow (G_1 \cdot G_2)$$

Furthermore, $G_1 \leq G_1$ and $G_2 \leq G_2$ implies $(G_1 \cdot G_2) \in (\downarrow G_1 \cdot \downarrow G_2)$. Note that every $G \in \downarrow(G_1 \cdot G_2)$ holds that $G \leq (G_1 \cdot G_2)$ and that $\downarrow G_1 \cdot \downarrow G_2$ is a down set, so that $G \in \downarrow G_1 \cdot \downarrow G_2$ too. Hence $\downarrow G_1 \cdot \downarrow G_2 \;=\; \downarrow(G_1 \cdot G_2)$. That is, $\downarrow$ also preserves applications. $\square$

Proof of Theorem 3.2

Proposition A.1. *Given a set of labels Lb, the* $\langle \mathbf{C}_{Lb}, *, \cdot \rangle$ *is isomorphic to the free algebra generated by atomic causal graphs* $\mathbf{A}_{Lb}$*, that is, for any set* $\mathbf{S}$ *and homomorphism* $\delta : \mathbf{A}_{Lb} \longrightarrow \mathbf{S}$*, there exists a homomorphism* $term : \mathbf{C}_{Lb} \longrightarrow \mathbf{S}$ *given by*

$$term(G) \mapsto \prod \{ \, \delta(v_1) \cdot \delta(v_2) \mid (v_1, v_2) \in G \, \} \tag{110}$$

such that $\delta(G) = term(G)$ *for all atomic causal graph* $G \in \mathbf{A}_{Lb}$. □

Proof. Recall that G^* represents the transitive and reflexive closure of G and, by definition, $\prod_{G \in U} = \big(\bigcup_{G \in U} \big)^*$. Furthermore, we write just v instead of $\delta(v)$. Let $F = \bigcup_{G \in U} G$. Then

$$\prod_{G \in U} term(G) = \prod_{G \in U} \prod_{(v_1, v_2) \in G} v_1 \cdot v_2 = \prod_{(v_1, v_2) \in F} v_1 \cdot v_2$$

Furthermore, from the equations in Figure 16 it follows

$$\begin{aligned} \prod_{(v_1, v_2) \in F} v_1 \cdot v_2 = \prod_{(v_1, v_2) \in F^*} l_1 \cdot l_2 &= term(F^*) \\ = term\Big(\Big(\bigcup_{G \in U} G \Big)^* \Big) &= term\Big(\prod_{G \in U} G \Big) \end{aligned}$$

That is, *term* preserves products. Let us see that it also preserves applications.

$$\begin{aligned} term(G_u) \cdot term(G_v) &= \Big(\prod_{(u_1, u_2) \in G_u} u_1 \cdot u_2 \Big) \cdot \Big(\prod_{(v_1, v_2) \in G_v} v_1 \cdot v_2 \Big) \\ &= \prod_{(u_1, u_2) \in G_u, \ (v_1, v_2) \in G_v} (u_1 \cdot u_2) \cdot (v_1 \cdot v_2) \end{aligned}$$

Note that

$$\begin{aligned} (u_1 \cdot u_2) \cdot (v_1 \cdot v_2) \ &= \ u_1 \cdot (u_2 \cdot v_1 \cdot v_2) \ = \ u_1 \cdot (u_2 \cdot v_1 * v_1 \cdot v_2) \\ &= (u_1 \cdot u_2 \cdot v_1) * (u_1 \cdot v_1 \cdot v_2) \\ &= (u_1 \cdot u_2 * u_2 \cdot v_1) * (u_1 \cdot v_1 * v_1 \cdot v_2) \\ &= (u_1 \cdot u_2 * u_2 \cdot v_1) * (u_1 \cdot v_1 * v_1 \cdot v_2) * (u_1 \cdot v_2 * u_2 \cdot v_2) \end{aligned}$$

reorganizing, it follows that

$$(u_1 \cdot u_2) \cdot (v_1 \cdot v_2) \ = \ (u_1 \cdot u_2 * v_1 \cdot v_2) * (u_1 \cdot v_1 * u_2 \cdot v_1 * u_1 \cdot v_2 * u_2 \cdot v_2)$$

and then

$$\begin{aligned}
term(G_u) \cdot term(G_v) &= \prod_{(u_1,u_2)\in G_u,\ (v_1,v_2)\in G_v} (u_1 \cdot u_2 * v_1 \cdot v_2) \\
&\quad * \prod_{(u_1,u_2)\in G_u,\ (v_1,v_2)\in G_v} (u_2 \cdot v_1 * u_1 \cdot v_1 * u_1 \cdot v_1 * u_1 \cdot v_2) \\
&= \prod_{(l_1,l_2)\in G_u \cup G_v} (l_1 \cdot l_2) * \prod_{u \in G_u,\ v \in G_v} (u \cdot v) \\
&= \prod_{(l_1,l_2)\in G_u \cdot G_v} (l_1 \cdot l_2) = term(G_1 \cdot G_2)
\end{aligned}$$

So that, *term* preserves applications, and hence, it is an homomorphism. □

Proposition A.1 formalize the intuition that every causal graph can be expressed as a combination of atomic causal graphs, products ($*$) and applications '$\cdot$'. Furthermore, it states that the set of causal graphs $\mathbf{C}_{Lb}$ is the most general set we can build with atomic causal graphs, product ($*$) and application ($\cdot$) in the sense that, any other set $\mathbf{S}$ generated by the atomic causal graphs together with these operations cannot have more distinct elements.

Proof of Theorem 3.2. Recall the convention of represent atomic causal graph of the form $\{(v,v)\}$ just as v. It is clear that $\delta : \mathbf{A}_{Lb} \longrightarrow Lb$ given by $\delta(\{(v,v)\}) \mapsto v$ is a bijection. Note that $\langle \mathbf{C}_{Lb}, *, \cdot \rangle$ is a the free algebra generated by $\mathbf{A}_{Lb}$ (Proposition A.1) and $\langle \mathbf{C}^t_{Lb}, *, \cdot \rangle$ is the free algebra generated by Lb. Hence they are isomorphic. Furthermore

$$term(G) \mapsto \prod \{\ v_1 \cdot v_2 \mid (v_1,v_2) \in G\ \} \tag{55}$$

corresponds to (110) with the above defined δ. So that, by Proposition A.1, the function $term : \mathbf{C}_{Lb} \longrightarrow \mathbf{C}^t_{Lb}$ is an homomorphism.

Furthermore, let f be a function that recursively maps each label l to the atomic causal graphs $\{\ (l,l)\ \}$ and compounded terms to their corresponding operations. It is easy to see that $f \circ term(G) = G$ for all G in $\mathbf{C}_{Lb}$. Then $term(G_1) = term(G_2)$ implies $f \circ term(G_1) = f \circ term(G_2)$ which, in its turn implies $G_1 = G_2$, that is, *term* is an injective function.

Finally note every term t can be rewritten as $x_1 * x_2 \ldots * x_n$ where each x_i only contains applications by successive application of the distributive equation. Note as well that each x_i can be rewritten as $l_1 \cdot l_2 * l_2 \cdot l_3 * \ldots * l_{n-1} \cdot l_n$ by application of the transitive equation if they are formed by more than two labels and as $l \cdot l$ by the label idempotence equation if they are formed by a single label l. Furthermore, each pair $l_i \cdot l_j * l_j \cdot l_k$ is equivalent to $l_i \cdot l_j * l_j \cdot l_k * l_i \cdot l_k$. That is, every term $t = term(G)$ for some causal graph G. So that, $term(G)$ is subjective function, and therefore a bijection too. □

Proof of Theorem 6.1

Lemma 9. *Given a completely labelled program P, $graph(\pi)$ is acyclic for any non-redundant proof π.* □

Proof. It is clear that, if π has not direct subproofs, then $graph(\pi) = \{(l_R, l_A)\}$, and so, it is acyclic. Assume as induction hypothesis, that for every sub-proof of π the statement holds, but $graph(\pi)$ has a cycle. Then one of the following holds.

- (l_R, l_A) and (l_A, l_R) are edges of $graph(\pi)$. Note that, the edge (l_R, l_A) implies that $head(R) = A$ and the edge (l_A, l_R) implies that A is in the body of R. Hence for every subproof π_R of π with rule R there is a subproof π_A of the atom A. Note that π_R is also a proof of the atom A. Let π' be the result of replacing each occurrence of π_R by π_A. Then π' is proof of the same atom than π and $subproofs(\pi') \subset subproofs(\pi)$ which contradicts the assumption that π is non-redundant.
- (l_B, l_R) and (l_R, l_B) are edges of $graph(\pi)$. Note that this case is symmetrical to the previous one.

Hence $graph(\pi)$ is acyclic. □

Note that $graph(\pi)$ is not necessary a transitive reduction as can be shown by the following program.

$$a \leftarrow b, c \qquad\qquad b \leftarrow c \qquad\qquad c$$

Proposition A.2. *Given a completely labelled program P, for any non-redundant proof π, $cgraph(\pi)$ is acyclic, that is, it does not contain any cycle but the reflexive ones.* □

Proof of Proposition A.2. By Lemma 9, it follows that $graph(\pi)$ is acyclic. Then its transitive closure does not contain cycles and its reflexive one only contains the reflexive cycles. Hence $cgraph(\pi)$ is acyclic. □

Definition A.1 (Proof heigh)**.** *Given a proof π its* height *is given by*

$$height(\pi) = 1 + \max\{\ height(\pi') \mid \pi' \text{ is a direct sub-proof of } \pi\ \}$$

If π has no direct subproofs, then $height(\pi) = 1$.

Lemma 10. *Let P be a positive program and $\pi = \pi(A)$ be a proof of A w.r.t. P with $height(\pi) = h$, Then $cgraph(\pi) \leq T_P^h(A)$.* □

Proof. In case that $h = 1$ the antecedent of $\pi(A)$ is empty, i.e.

$$\pi(A) = \frac{\top}{A}\ (R)$$

Then $cgraph(\pi) = l_R \cdot l_A$. Furthermore, since the fact $(l_R : A)$ is in the program P, it follows that $l_R \cdot l_A \in T_P^1(A)$.

In the remain cases, we proceed by structural induction assuming that for every natural number $h \leq n-1$, literal B_i and proof $\pi_i = \pi(B_i)$ of B_i w.r.t. P whose $height(\pi_i) = h$ it holds that $graph(\pi_i) \leq T_P^h(B_i)$ and we will show it in case that $h = n$.

Since $height(\pi) > 1$ it has a non empty antecedent, i.e.

$$\pi(p) = \frac{\pi_1(\psi_1 :: B_1), \ldots, \pi_m(\psi_m :: B_m)}{A}\ (R)$$

Note that $height(\pi_i) \leq n-1$ and then, by induction hypothesis, it holds that $graph(\pi_i) \leq T_P^{h-1}(B_i)$. Note also that, by since π_i is a proof of $(\psi_i :: B_i)$, then $\psi_i(graph(\pi_i)) = 1$ for all $1 \leq i \leq m$. Hence, by T_P definition,

$$\prod\{\ graph(\pi_i) \mid 1 \leq i \leq m\ \} \cdot l_R \cdot l_A \quad \leq \quad T_P^h(A)$$

Finally note that, by Proposition 3.5, this is just, $cgraph(\pi)$. □

Lemma 11. *Let P be a completely labelled positive program. For every literal A and $\leq$-maximal causal graph $G \leq T_P^\omega(A)$ there is a non-redundant proof $\pi = \pi(A)$ of A w.r.t. P such that $cgraph(\pi) = G$.*

Proof. For any causal graph $G \leq_{\max} T_P^k(A)$, there is a positive rule R of the form (92) with $n = 0$ and maximal graphs $G_1, \ldots, G_m$ such that $\psi_i(G_i) = 1$ and each G_i is maximal holding $G_i \leq T_P^{h-1}(B_i)$ and

$$G = (G_1 * \ldots * G_m) \cdot l_R \cdot l_A$$

We assume as induction hypothesis that for every literal B_i there is a non-redundant proof $\pi_i = \pi(B_i)$ of B_i w.r.t. P such that $graph(\pi_i) = G_i$. Let $\pi = \pi(A)$ defined as

$$\pi(p) = \frac{\pi_1, \ldots, \pi_m}{A} \ (R)$$

a proof of A w.r.t. P. Furthermore, by Proposition 3.5, $cgraph(\pi) = G$. Moreover, since G is $\leq$-maximal (alternatively $\subseteq$-minimal), by Proposition 3.1, it follows that π is non-redundant. □

Lemma 12. *Given a completely labelled, positive program P, a proof $\pi = \pi(A)$ is a non-redundant proof of A iff $cgraph(\pi)$ is a cause of A w.r.t. its least model.* □

Proof. Let I be the least causal model of the positive program P. For the only if direction. Let $\pi = \pi(A)$ be a non-redundant proof of A. Then, by Lemma 10, $cgraph(\pi) \leq T_P^h(A) \leq T_P^\omega(A)$. Recall that, by Theorem 4.1, the least model I of P is $T_P^\omega(A)$. Consequently $cgraph(\pi) \leq I(A)$. Moreover, if $cgraph(\pi)$ is not maximal, then there is G' such that $cgraph(\pi) < G'$ (alternatively $cgraph(\pi) \supset G'$) which, by Proposition 3.1, implies that π is redundant which is a contradiction with the assumption. Hence $cgraph(\pi)$ is maximal, and so, a sufficient cause of A.

For the if direction. If G be a sufficient cause of A, the, by Lemma 11 there is a non-redundant proof $\pi = \pi(A)$ such that $cgraph(\pi) = G$. □

Lemma 13. *Given a completely labelled program P, a proof $\pi = \pi(A)$ is a proof of A w.r.t. a causal stable model (resp. answer set) I of P iff π is a non-redundant proof of A w.r.t. the least model of P^I.* □

Proof. For the only if direction. Let $\pi = \pi(A)$ be a proof of A w.r.t. a causal stable model (resp. answer set) I. Since π is a proof w.r.t. I, every rule R appearing in π holds that $I \models body^-(R)$ and, therefore, R is in P^I and, consequently π is a proof of A w.r.t. P^I.

The other way around. Let $\pi = \pi(A)$ be a proof of A w.r.t. P^I. It is clear that every rule R appearing in P^I and so in π holds that $I \models body^-(R)$ and, therefore π is a proof of A w.r.t. P and I. □

Proof of Theorem 6.1. By Lemma 13 π is a proof of A w.r.t. P and I iff is a proof of A w.r.t. the least model of P^I. Furthermore, by Lemma 12, π is a proof of A w.r.t. the least model of P^I iff $cgraph(\pi)$ is a cause of A w.r.t. the least model of P^I, which since I is an stable model (resp. answer set) of P must be I itself. That is, π is a proof of A w.r.t. P and I iff $cgraph(\pi)$ is a cause of A w.r.t. interpretation I. □

Proof of Theorem 3.1

Lemma 14. *Let $\langle P, \leq, +, * \rangle$ partial lattice corresponding to the semilattice $\langle P, * \rangle$, and $(I_s)_{s \in S}$ a family of ideals of P. Let*

$$\begin{aligned} A_S &= \{\ I' \in \mathcal{I}_P \mid I_s \subseteq I' \text{ for some } s \in S\ \} \\ B_S &= \{\ I' \in \mathcal{I}_P \mid \bigcap_{s \in S} I_s \subseteq I'\ \} \end{aligned}$$

Then $A_S^{FI} = B_S$ □

Proof. We will show first that $A_S^{FI} = B_S$. Suppose there is $I' \in B_S$ such that $I' \notin A_S^{FI}$, then there is $F \in A_S^F$ such that $I' \cap F = \emptyset$, and furthermore, since $F \in A_S^F$, it holds that $I'' \cap F \neq \emptyset$ for all $I'' \in A_S$. Let G be the causal graph given by

$$G = \prod \{\ G' \mid G' \in I_s \cap F \text{ for all } s \in S\ \}$$

Note that $G' \in I_s \cap F$ implies $G \in F$ (filters are close under defined infimum) and $G \in I_s$ ($G \leq G'$ and ideals is down close under $\leq$). Furthermore, $G \in I_s \cap F$ for every I_s also implies $G \in \bigcap_{s \in S} I_s \subseteq I'$ because $I' \in B_S$. Hence $G \in I' \cap F$ which contradicts the fact that from the assumption it follows $I' \cap F = \emptyset$. Hence $B_S \subseteq A_S^{FI}$. Suppose now that there is $I' \in A_S^{FI}$ such that $I' \notin B_S$, then $I' \cap F \neq \emptyset$ for all $F \in A_S^F$ but $\bigcap_{s \in S} I_s \not\subseteq I'$. Let G be a causal graph in $\bigcap_{s \in S} I_s \backslash I'$ and $F = \uparrow G$. It is clear that $F \in \mathcal{F}_P$ and $F \cap I'' \neq \emptyset$ for all $I'' \in A_S$ because $G \in \bigcap_{s \in S} I_s \subseteq I_s \subseteq I''$. Hence $F \in A_S^F$. Furthermore $F \cap I' = \emptyset$, because $G \notin I'$, which is a contradiction with the assumption. Therefore $A_S^{FI} = B_S^{FI}$. □

Proposition A.3. *Given a partial lattice $\langle P, \leq, +, * \rangle$ corresponding to the semilattice $\langle P, * \rangle$, the map $\epsilon_I : B_P \longrightarrow \mathcal{I}_P$ given by $(\mathbf{I}^F, \mathbf{I}) \mapsto \bigcap \mathbf{I}$ is an isomorphism between the completely distributive lattices $\langle B_P, \leq, +, * \rangle$ and $\langle \mathcal{I}_P, \subseteq, \cup, \cap \rangle$.* □

The proof Proposition A.3 will rely on the following auxiliary Lemma:

Proof of Proposition A.3. We will show first that the function $f : \mathcal{I}_P \longrightarrow B_P$ given by $I \mapsto (\mathbf{I}^F, \mathbf{I})$ with $\mathbf{I} = \{ I' \in \mathcal{I}_P \mid I \subseteq I' \}$ is the inverse of ϵ_I. By the above Lemma 14, it follows that

$$\{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \text{ for some } s \in S \}^{FI} \ = \ \{ I' \in \mathcal{I}_P \mid \bigcap_{s \in S} I_s \subseteq I' \}$$

Furthermore, taking $(I_s)_{s \in S} = I$ follows

$$\{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \}^{FI} \ = \ \{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \}$$

Then the image of f is a subset of B_P. Furthermore

$$\epsilon_I \circ f(I) \ = \ \epsilon_I(\mathbf{I}^F, \mathbf{I}) \ = \ \bigcap \mathbf{I} \ = \ \bigcap \{ I' \in \mathcal{I}_P \mid I \subseteq I' \} \ = \ I$$

Moreover

$$f \circ \epsilon_I(\mathbf{I}^F, \mathbf{I}) = f\left(\bigcap \mathbf{I}\right) = (\mathbf{I}_2^F, \mathbf{I}_2)$$

where $\mathbf{I}_2 = \left\{ I' \in \mathcal{I}_P \;\middle|\; \bigcap \mathbf{I} \subseteq I' \right\} \supseteq \mathbf{I}$ because every $I' \in \mathbf{I}$ holds $\bigcap \mathbf{I} \subseteq I'$. Suppose there is $I' \in \mathbf{I}_2$ such that $I' \notin \mathbf{I}$. Since $\mathbf{I} = \mathbf{I}^{FI}$, then there is $F \in \mathbf{I}^F$ s.t.

- $I' \cap F = \emptyset$ and that
- $I'' \cap F \neq \emptyset$ for all $I'' \in \mathbf{I}$ because $F \in \mathbf{I}^F$, and
- $F \notin \mathbf{I}_2^F$ (otherwise it would be $I' \notin \mathbf{I}_2^{FI} = \mathbf{I}_2$).

But, since $F \notin \mathbf{I}_2^F$, there is some $I'' \in \mathbf{I}_2$ such that $F \cap I'' = \emptyset$. Furthermore, $I'' \in \mathbf{I}_2$ implies $\bigcap \mathbf{I} \subseteq I''$ which, in its turn implies $F \cap \bigcap \mathbf{I} = \emptyset$. However, $G = \prod \{ G' \mid G' \in I'' \cap F \text{ for some } I'' \in \mathbf{I} \}$ holds that $G \in I''$ for all $I'' \in \mathbf{I}$. Note that $G \leq G'$ for some $G' \in I''$ and all $I'' \in \mathbf{I}$ and ideals are down close. So that $G \in F \cap \bigcap \mathbf{I} = \emptyset$ which is a contradiction. Hence $\mathbf{I} = \mathbf{I}_2$ and therefore f is the inverse of ϵ_I, and so, we denote it by ϵ_I^{-1}. We will show now that

$$\sum_{s \in S} (\mathbf{I}_s^F, \mathbf{I}_s) \ = \ (\mathbf{I}^F, \mathbf{I}) \quad \text{iff} \quad \bigcup_{s \in S} I_s \ = \bigcap \mathbf{I}$$

Recall that, by definition,

$$\begin{aligned} \mathbf{I} \ &= \ \bigcap_{s \in S} \mathbf{I}_s \ = \ \bigcap_{s \in S} \{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \} \\ &= \ \{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \text{ for all } s \in S \} \\ &= \ \left\{ I' \in \mathcal{I}_P \;\middle|\; \bigcup_{s \in S} I_s \subseteq I' \right\} \end{aligned}$$

Note that also that $\epsilon_I^{-1} \circ \epsilon_I(\mathbf{I}_s^F, \mathbf{I}_s) = (\mathbf{I}_s^F, \mathbf{I}_s)$ implies $\mathbf{I} = \{ I' \in \mathcal{I}_P \mid \bigcap \mathbf{I} \subseteq I' \}$ and this holds if and only if $\bigcup_{s \in S} I_s = \bigcap \mathbf{I}$. That is, the function ϵ_I maps the $\sum$ operation into $\bigcap$. Next we show that ϵ_I also preserves the $\prod$ operation. Let now

$$\prod_{s \in S} (\mathbf{I}_s^F, \mathbf{I}_s) = (\mathbf{I}^F, \mathbf{I})$$

by definition

$$\mathbf{I} = \Big(\bigcup_{s \in S} \mathbf{I}_s \Big)^{FI} = \{ I' \in \mathcal{I}_P \mid I_s \subseteq I' \text{ for some } s \in S \}^{FI}$$

and then, by the above Lemma 14, it follows that

$$\mathbf{I} = \Big\{ I' \in \mathcal{I}_P \;\Big|\; \bigcap_{s \in S} I_s \subseteq I' \Big\}$$

which holds if and only if $\bigcap_{s \in S} I_s = \bigcap \mathbf{I}$. That is, the function ϵ_I maps the $\prod$ operation into $\bigcap$. Finally note that, as usual $(\mathbf{I}_1^F, \mathbf{I}_1) \leq (\mathbf{I}_2^F, \mathbf{I}_2)$ if and only if $(\mathbf{I}_1^F, \mathbf{I}_1) * (\mathbf{I}_2^F, \mathbf{I}_2) = (\mathbf{I}_1^F, \mathbf{I}_1)$ and $\bigcap \mathbf{I}_1 \subseteq \bigcap \mathbf{I}_2$ if and only if $\bigcap \mathbf{I}_1 \cap \bigcap \mathbf{I}_2 = \bigcap \mathbf{I}_1$. So that ϵ_I also maps $\leq$ into $\subseteq$. Hence ϵ_I is an bijective homomorphism between $\langle \mathbf{B}_P, \leq, \sum, \prod \rangle$ and $\langle \mathcal{I}_P, \subseteq, \cup, \cap \rangle$, that is an isomorphism. □

Proposition A.4. *The mapping $\downarrow : \mathbf{C}_{Lb} \longrightarrow \mathbf{V}_{Lb}$ is an injective homomorphism between structures $\langle \mathbf{C}_{Lb}, \leq, +, *, \cdot \rangle$ and $\langle \mathbf{V}_{Lb}, \leq, +, *, \cdot \rangle$ where the sum of causal graphs is the least upper bound of $\leq$ when it is defined.* □

Proof. By Proposition 3.10 it follows that $\downarrow$ is an isomorphism between structures $\langle \mathbf{C}_{Lb}, \leq, *, \cdot \rangle$ and $\langle \Downarrow\mathbf{C}_{Lb}, \leq, *, \cdot \rangle$. Since $\Downarrow\mathbf{C}_{Lb} \subseteq \mathbf{V}_{Lb}$ it follows that $\downarrow$ is injective, so that it just remain to show that $\downarrow$ preserves sums. Note that, if $\sum_{G \in U} G$ is defined, then $\sum_{G \in U} G = G_u$ with $G \leq G_u$ for all $G \in U$. Hence $\sum_{G \in U} {\downarrow} G = {\downarrow} G_u$ and then

$$\downarrow \sum_{G \in U} G = {\downarrow} G_u = \sum_{G \in U} {\downarrow} G$$

So that $\downarrow$ is an injective homomorphism. □

Proposition A.5. *Given a set of labels Lb, the structure $\langle \mathbf{V}_{Lb}, +, * \rangle$ is isomorphic to the free completely distributive (complete) lattice generated by the principal ideals $\Downarrow\mathbf{C}_{Lb}$. that is, for any set $\mathbf{S}$ and homomorphism $\delta : \mathbf{C}_{Lb} \longrightarrow \mathbf{S}$, there exists a homomorphism $term : \mathbf{V}_{Lb} \longrightarrow \mathbf{S}$ given by*

$$term(U) \mapsto \sum \{ \delta(G) \mid G \in U \} \tag{111}$$

such that $\delta(G) = term(G)$ for all causal graphs $G \in \mathbf{C}_{Lb}$. □

Proof. By Theorem 2.2, for each order-preserving map δ from $\langle \Downarrow \mathbf{C}_{Lb}, \leq \rangle$ to a completely distributive complete lattice $\mathbf{S}$, there exists a homomorphism $h : B_{\mathbf{C}_{Lb}} \longrightarrow \mathbf{S}$ such that $\delta = h \circ \epsilon_{\mathbf{C}_{Lb}}$. Furthermore, by Proposition A.3, the function $\epsilon_I : B_{\mathbf{C}_{Lb}} \longrightarrow \mathcal{I}_{\mathbf{C}_{Lb}}$ is an isomorphism between structures $\langle B_P, \leq, +, * \rangle$ and $\langle \mathcal{I}_{\mathbf{C}_{Lb}}, \subseteq, \cup, \cap \rangle$, and hence, the function $h \circ \epsilon_I^{-1} : \mathcal{I}_P \longrightarrow \mathbf{S}$ is an homomorphism. Note that by definition causal values are the ideals of $\mathbf{C}_{Lb}$ and are closed under defined suprema because it is only defined when it is the maximum, that is $\mathbf{V}_{Lb} = \mathcal{I}_P$.

Let $\mathbf{I}_1$ and $\mathbf{I}_2$ such that $\mathbf{I}_1 = \{ I \in \mathcal{I}_P \mid G \in I \}$ and $\mathbf{I}_2 = \{ I \in \mathcal{I}_P \mid {\downarrow} G \subseteq I \}$ for some causal graph G. Then $\epsilon_{\mathbf{C}_{Lb}}(G) = (\mathbf{I}_1^F, \mathbf{I}_1)$ and $\epsilon_I^{-1}({\downarrow} G) = (\mathbf{I}_2^F, \mathbf{I}_2)$ Note that $G \in I$ implies that ${\downarrow} G \subseteq I$ because ideals are down close and conversely is trivial because $G \in {\downarrow} G$, so that $\mathbf{I}_1 = \mathbf{I}_2$, and then, $\epsilon_{\mathbf{C}_{Lb}}(G) = \epsilon_I^{-1}({\downarrow} G)$. Hence $\delta(G) = h \circ \epsilon_I^{-1}({\downarrow} G)$ for all causal graph G. Then

$$\begin{aligned} term(U) &= \sum \{ \, \delta(G) \mid G \in U \, \} = \sum \{ \, h \circ \epsilon_I^{-1}({\downarrow} G) \mid G \in U \, \} \\ &= h \circ \epsilon_I^{-1} \Big(\sum \{ \, {\downarrow} G \mid G \in U \, \} \Big) = h \circ \epsilon_I^{-1}(U) \end{aligned}$$

for all causal value U, and hence $term$ is an homomorphism. □

Proof of Theorem 3.1. Note that Theorem 3.1 is just a less technical rephrasing of the statement of the above Proposition A.5. □

Proof of Theorem 3.4

Definition A.2 (Term function). *For every set $\mathbf{S}$ and homomorphism $\delta : \mathbf{A}_{Lb} \longrightarrow \mathbf{S}$ we denote by $term^v_\delta : \mathbf{V}_{Lb} \longrightarrow \mathbf{S}$ the function given by*

$$term^v_\delta(U) \mapsto \sum \{ \, term^g_\delta(G) \mid G \in U \, \}$$

where $term^g_\delta : \mathbf{C}_{Lb} \longrightarrow \mathbf{S}$ is the function given by (110) and $\mathbf{A}_{Lb}$ is the set of the principal ideals of atomic causal graphs. □

Proposition A.6. *Given a set of labels Lb, $\langle \mathbf{V}_{Lb}, \leq, +, *, \cdot \rangle$ is isomorphic to the free algebra generated by the principal ideals of atomic causal graphs $\mathbf{A}_{Lb}$, that is, $term^v_\delta$ is an homomorphism for any set $\mathbf{S}$ and homomorphism $\delta : \mathbf{A}_{Lb} \longrightarrow \mathbf{S}$.* □

Proof. Notice that, by Proposition A.5, the homomorphism $term^v_\delta$ preserves sums and products, so that

$$term^v_\delta(U) \;=\; term^v_\delta\Big(\bigcup_{G\in U} {\downarrow} G\Big) \;=\; term^v_\delta\Big(\sum_{G\in U} {\downarrow} G\Big) \;=\; \sum_{G\in U} term^v_\delta({\downarrow} G)$$

Furthermore

$$term^v_\delta({\downarrow} G) \;=\; \sum_{G'\leq G} term^c_\delta(G') \;=\; term^c_\delta(G)$$

so that, $term^v_\delta({\downarrow} G) = term^c_\delta(G)$. Recall that, from Proposition A.1, the function $term^c_\delta$, and so now also $term^v_\delta$, is an homomorphism between structures $\langle \mathbf{C}_{Lb}, *, \cdot\rangle$ and $\langle \mathbf{S}, *, \cdot\rangle$. Since $term^v_\delta$ preserves sums, it remains to show that is preserves $\leq$, but note that when sums or products are preserved, then $\leq$ is also preserved. $\square$

Proof of Theorem 3.4. Note that since $\langle \mathbf{V}_{Lb}, \leq, +, *, \cdot\rangle$ and $\langle \mathbf{V}^t_{Lb}, \leq, +, *, \cdot\rangle$ are structures of the same class (they have the same operations following the equations in Figures 13, 14 and 15), by Proposition A.6, it follows that $term^v_\delta$ is an homomorphism. Notice that when we take $\mathbf{S} = \mathbf{V}^t_{Lb}$ and δ as $\delta(G) \mapsto v$ such that v is the only vertex of G as in Theorem 3.2, it follows that the image of $term^c_\delta$ is $\mathbf{C}^t_{Lb}$ which is the subset of $\mathbf{V}^t_{Lb}$ formed by term without sums. Recall that, from Theorem 3.2, $term^c_\delta : \mathbf{C}_{Lb} \longrightarrow \mathbf{C}^t_{Lb}$ is an isomorphism. That is, every term without sums t hold that $(term^c_\delta)^{-1}(t) = G_t$ for some causal graphs G_t. Let f be a function that maps each term t without sums to the principal ideal ${\downarrow} G_t$ and sums to set unions. It is easy to see that $f \circ term^v_\delta(U) = U$ for all U in $\mathbf{V}_{Lb}$ which implies that $term^v_\delta$ is an injective function. Finally note that, by successive application of the distributive equations, every term t can be rewritten as $\sum_{G_i\in U} term^c_\delta(G_i)$ where each $term^c_\delta(G_i)$ is a term without sums that represents a causal graph G_i. Then $\Downarrow U$ is a causal value, and

$$term(\Downarrow U) \;=\; \sum_{G_i\in U}\sum_{G'\leq G} term^c_\delta(G') \;=\; \sum_{G_i\in U} term^c_\delta(G_i) \;=\; t$$

That is, every term $t = term^v_\delta(U)$ for some causal value U. So that, $term^v_\delta$ is subjective function, and therefore a bijection too. For convenience we avoid the subindex and superindices and denote by *term* both functions $term^v_\delta$ and $term^c_\delta$. $\square$

Proof of Proposition 3.11

Proof of Proposition 3.11. For properties of product $(*)$ and sum $(+)$ just note that by definition product, sum and sufficient relation respectively correspond to set intersection, union and subset relation.

In order to show that application $(\cdot)$ is continuous let $t, v_1 \leq v_2$ be causal values. Then for all $G \in t \cdot v_1$ there exists $G_t \in t$ and $G_{v_1} \in v_1$ such that $G \leq G_t \cdot G_{v_1}$. Furthermore, since $v_1 \leq v_2$, there exists $G_{v_2} \in v_2$ such that $G_{v_1} \leq G_{v_2}$ and, since application is monotonic for causal graphs (Proposition 3.6, it follows that $G \leq G_t \cdot G_{v_1} \leq G_t \cdot G_{v_2}$ and, therefore, $G \in t \cdot v_2$. In a similar manner it follows that $v_1 \cdot t \leq v_2 \cdot t$.

Let now G be a causal graph and U be a directed set of causal values. Note that $u \leq \sum_{u \in U} u$ for all $u \in U$. Then, since application is monotonic, it follows that ${\downarrow}\{G\} \cdot u \leq {\downarrow}\{G\} \cdot \sum_{u \in U} u$ for all $u \in U$. Thus,

$$\sum_{u \in U} {\downarrow}\{G\} \cdot u \quad \leq \quad {\downarrow}\{G\} \cdot \sum_{u \in U} u$$

The other way around. Pick $G' \in {\downarrow}\{G\} \cdot \sum_{u \in U} u$. Then there exists $G_t \in {\downarrow}\{G\}$ and $G_u \in \sum_{u \in U} u$ such that $G' \leq G_t \cdot G_u$. Furthermore, since $G_u \in \sum_{u \in U} u$, it follows that $G_u \in u$ for some $u \in U$. Hence $G' \in {\downarrow}\{G\} \cdot u$ for some $u \in U$ and, therefore $G' \in \sum_{u \in U} {\downarrow}\{G\} \cdot u$. Consequently

$$\sum_{u \in U} {\downarrow}\{G\} \cdot u \quad = \quad {\downarrow}\{G\} \cdot \sum_{u \in U} u$$

In a similar manner it follows that

$$\sum_{u \in U} u \cdot {\downarrow}\{G\} = \big(\sum_{u \in U} u\big) \cdot {\downarrow}\{G\}$$

Moreover, let T and U be two directed sets of causal values. Then

$$\big(\sum_{t \in T} t\big) \cdot \big(\sum_{u \in U} u\big) = \big(\sum_{t \in T} \sum_{G \in t} {\downarrow}\{G_t\}\big) \cdot \sum_{u \in U} \sum_{G_u \in U} {\downarrow}\{G_u\}\big)$$

and since application distributes over sums, it follows that

$$\begin{aligned}
(\sum_{t\in T} t)\cdot(\sum_{u\in U} u) &= \sum_{t\in T}\sum_{G_t\in t}(\downarrow\{G\}\cdot\sum_{u\in U}\sum_{G_u\in U}\downarrow\{G_u\}) \\
&= \sum_{t\in T}\sum_{G_t\in t}\sum_{u\in U}\sum_{G_u\in U}(\downarrow\{G_t\}\cdot\downarrow\{G_u\}) \\
&= \sum_{t\in T}\sum_{u\in U}\sum_{G_t\in t}\sum_{G_u\in U}(\downarrow\{G\}\cdot\downarrow\{G_u\}) \\
&= \sum_{t\in T}\sum_{u\in U}(\sum_{G_t\in t}\downarrow\{G\}\cdot\sum_{G_u\in U}\downarrow\{G_u\}) \\
&= \sum_{t\in T}\sum_{u\in U}(t\cdot u) \\
&= \sum_{(t,u)\in T\times U} t\cdot u
\end{aligned}$$

Finally to show that $u\cdot u'\leq u * u'$ note that, by definition

$$u\cdot u' \overset{\text{def}}{=} \Downarrow\{\, G\cdot G' \mid G\in u \text{ and } G'\in u' \,\}$$

and that $G\cdot G'\leq G$ and $G\leq G'$, so that $u\cdot u'\leq u$ and $u\cdot u'$. Since $u * u'$ is the greatest lower bound of $\leq$, it follows that $u\cdot u'\leq u * u'$. □

CHAPTER 4: CAUSAL SEMANTICS

Proof of Theorem 4.3

Lemma 15. *Let P and Q two positive labelled programs respectively over signatures $\langle Lit, Lb, \delta\rangle$ and $\langle Lit, Lb, \delta'\rangle$.*

- *If Q is the result of unlabelled rules with label l in P, that is $Q = P[l \mapsto 1]$, then $(T^k_P)^{cl} = (T^k_Q)^{cl}$.*
- *If $\delta'(A) = 1$ for some literal A and $\delta'(B) = \delta(B)$ for all literal $B \neq A$, then $(T^k_P)^{cl} = (T^k_Q)^{cl}$.* □

Proof. In case that $k = 0$, by definition, $T^k_P = T^k_Q = \mathbf{0}$ and $\mathbf{0}^{cl} = \mathbf{0}$. Otherwise the proof follows by induction assuming that the statement holds for the case $k-1$.

For every literal A, $(T^k_P)^{cl}(A) \neq 0$ iff $T^k_P(A) \neq 0$ iff there is some causal graph $G \leq T^k_P(A)$ iff there is a labelled rule of the form $l_R : A \leftarrow B_1, \ldots, B_m$ in P, there are causal graphs $G_1, \ldots, G_m$ such that $G_i \leq T^{k-1}_P(B_i)$ for all $1 \leq i \leq m$ and $G = (G_1 * \ldots * G_m) \cdot l_R \cdot \delta(A)$.

Note that, for $1 \leq i \leq m$, $G_i \leq T^{k-1}_P(B_i)$ iff $T^{k-1}_P(B_i) \neq 0$ iff $(T^{k-1}_P)^{cl}(B_i) \neq 0$. By induction hypothesis this holds iff $(T^{k-1}_Q)^{cl}(B_i) \neq 0$ iff $T^{k-1}_Q(B_i) \neq 0$ iff there is a causal graph $G'_i \leq T^{k-1}_Q(B_i)$.

Note also that there is a rule of the form $l_R : A \leftarrow B_1, \ldots, B_m$ in P iff there is a rule of the form $l'_R : A \leftarrow B_1, \ldots, B_m$ in Q where $l'_R = 1$ iff $l_R = l$ and $l'_R = l_R$ otherwise.

Hence, $(T^k_P)^{cl}(A) \neq 0$ iff $G' = (G'_1 * \ldots * G'_m) \cdot l'_R \cdot \delta(A) \leq T^k_Q(A)$ iff $T^k_Q(A) \neq 0$ iff $(T^k_Q)^{cl}(A) \neq 0$. Consequently $(T^k_P)^{cl} = (T^k_Q)^{cl}$.

In case that $k = \omega$, for $X \in \{P, Q\}$ $T^k_X(A) = \sum_{i \leq \omega} T^i_X(A)$, and therefore $T^k_P(A) = 0$ iff $T^i_P(A) = 0$ for some $i \leq \omega$ iff $T^i_Q(A) = 0$ iff $T^k_Q(A) = 0$. Hence $(T^k_P)^{cl} = (T^k_Q)^{cl}$.

It is easy to see that if the modification is done in $\delta(A)$ instead of l_R the same reasoning is applicable. □

Proof of Theorem 4.3. Let $P_0 = P$ and P_{i+1} the result of unlabelled some rule in P_i. From Lemma 15, it follows that

$$(T^\omega_P)^{cl} = (T^\omega_{P_0})^{cl} = (T^\omega_{P_1})^{cl} = \ldots = (T^\omega_{P_n})^{cl}$$

where n is the number of labelled rules in P. Let $Q_0 = P_n$ and Q_{i+1} a program of a signature $\langle Lit, Lb, \delta_{i+1} \rangle$ such that $\delta_{i+1}(A) = 1$ for some literal A such that $\delta_{i+1}(A) \neq 1$ and $\delta_{i+1}(B) = \delta_i(B)$ for all literal $B \neq A$. From Lemma 15 again, it follows that

$$(T_{Q_0}^{\omega})^{cl} = (T_{Q_1}^{\omega})^{cl} = \ldots = (T_{Q_m}^{\omega})^{cl}$$

where m is the number of literals in Lit such that $\delta(A) \neq 1$. That is, it holds that $(T_P^{\omega})^{cl} = (T_{Q_m}^{\omega})^{cl}$. Furthermore, from Theorem 4.1 these are respectively the least models I and J of P and Q_m. Note that, Q_m is a completely unlabelled program such that $\delta_m(A) = 1$ for all literal A. That is, Q_m is an standard program and J is the standard least model of Q_m which is the unlabelled version of P, and $J = J^{cl} = I^{cl}$. □

Proof of Theorem 4.4

Proof of Theorem 4.4. By definition I and I^{cl} assigns 0 to the same atoms, so that $P^I = P^{I^{cl}}$. Furthermore let Q (instead of P' for clarity) be the unlabelled version of P. Then $Q^{I^{cl}}$ is the unlabelled version of P^I.

1. Let I be a stable model of P and J be the least model of $Q^{I^{cl}}$. Then, I is the least model of P^I and, from Theorem 4.3, it follows that $I^{cl} = J$, that is I^{cl} is a stable model of Q.
2. Let I' is a stable model of Q and I be the least model of $P^{I'}$. Since I' is a stable model of Q, by definition it is the least model of $Q^{I'}$, furthermore, since $Q^{I'}$ is the unlabelled version of $P^{I'}$ it follows, from Theorem 4.3, that $I^{cl} = I'$. Note that $P^I = P^{I^{cl}} = P^{I'}$. Thus I is a stable model of P. □

Proof of Theorem 4.5

Proof of Theorem 4.5. By definition, if I is consistent causal answer set, it is an stable model. Then, by Theorem 4.4, I^{cl} is an stable model of P'. Furthermore, since I is consistent, $I(A) = 0$ or $I(\overline{A}) = 0$ for all literal A, so $I^{cl}(A) = 0$ or $I^{cl}(\overline{A}) = 0$ for all literal A, that is, I^{cl} is a answer set of P'. On the contrary, if I is an inconsistent answer set, there is an atom a such that $P^I \models a$ and $P^I \models \overline{a}$. Hence, by Corollary 4.1, $P'^I \models a$ and $P'^I \models \overline{a}$, and then I^{cl} is an inconsistent answer set of P'^I. Note that $P'^I = P'^{I^{cl}}$ and consequently I^{cl} is an inconsistent answer set of P'.

Note that, if either $I(A) = 0$ or $I(\overline{A}) = 0$ for all literal A, then $I^{cl}(A) = 0$ or $I^{cl}(\overline{A}) = 0$ for all literal A, and if $I(A) \neq 0$ for all literal A, then $I^{cl}(A) = 1$ for all literal A. Hence I is an answer set of P as well.

The other way around, if I' is a answer set of P', then it is an stable model of P' and, by Theorem 4.4, there is a unique causal stable model and a unique causal answer set I of P s.t. $I' = I^{cl}$. If otherwise I' is an inconsistent answer set of P', then there is an atom a such that $P'^{I'} \models a$ and $P'^{I'} \models \overline{a}$. Then, by Corollary 4.1, it follows that $P^{I'} \models a$ and $P^{I'} \models \overline{a}$. Therefore, $I = I'$ is an inconsistent answer set of P. □

Proof of Theorem 4.6

Proof of Theorem 4.6. From the only if direction, note that, by Definition 4.9, it follows that every proof π holds $I \models body^-(R)$ for all R in π, and hence, π is a proof w.r.t. P^I. Then, from Theorem 4.2, $graph(\pi)$ is a cause of A w.r.t. I and P^I, and since I is a stable model of P, then $graph(\pi)$ is a cause of A w.r.t. interpretation I and program P.

From the if direction, note that, since I is an stable model of P, then I is the least model of P^I. Hence, from Theorem 4.2, π is a proof of A w.r.t. P^I. Note that every rule R in P^I holds $I \models body^-(R)$, so that, π is also proof w.r.t. interpretation I and program P. □

Lemma 16. *Let P be a head labelled program and π be a non-redundant proof of some literal. If $cgraph(\pi)$ contains an edge (l_1, l_n), then there are subproofs π_1 and π_n of π such that*

- *π_1 is a sub-proof of π_n,*
- *either l_i is the label of the rule in π_i or l_i is the label of the literal in the conclusion π_i with $i \in \{1, n\}$.* □

Proof. Let us denote by $\pi \prec \pi'$ that π is a sub-proof of π'.

Assume that $(l_1, l_n) \in cgraph(\pi)$. Then there is a sequence of edges such that $\{(l_1, l_2), (l_2, l_3), \ldots, (l_{n-1}, l_n)\} \subseteq graph(\pi)$.

By definition of $graph(\pi)$ it follows, for $2 \leq i \leq n$, that there are proofs π_{A_i} and π_{B_i} such that π_{A_i} is a proof of A_i, π_{B_i} is a proof of B_i, π_{B_i} is a direct sub-proof of π_{A_i}, $\pi_{A_i} \prec \pi$ and either

1. l_i is the label of the rule in π_{A_i} and $l_{i+1} = \delta(A_i)$, or
2. $l_i = \delta(B_i)$ and l_{i+1} is the label of the rule in π_{A_i}

Note that for every edge in the for of (l_i, l_{i+1}) there is an edge of the form (l_{i+1}, l_{i+2}) for $1 \leq i \leq n-2$.

If (1), then $l_{i+1} = \delta(A_i)$, and then, since every literal A has an associated label $\delta(A)$ different from the label of any rule, it follows that $l_{i+1} = \delta(A_i)$ implies $l_{i+1} = \delta(B_{i+1})$ for $1 \leq i \leq n-2$. Furthermore, since different literals have different labels, it must hold that $A_i = B_{i+1}$.

Otherwise (2), l_{i+1} is the label of the rule in π_{A_i} and also of the rule in $\pi_{B_{i+1}}$. Since the program is head labelled — that is, rules with different heads has different labels — it follows that π_{A_i} and $\pi_{B_{i+1}}$ are proof of the same literal, that is $A_i = B_{i+1}$.

Then, since π is non-redundant, it follows that there are not different subproofs with the same consequent, and hence $A_i = B_{i+1}$ implies $\pi_{A_i} = \pi_{B_{i+1}}$. Hence, since $\pi_{B_{i+1}}$ is a direct sub-proof of $\pi_{A_{i+1}}$, it follows that $\pi_{A_i} \prec \pi_{A_{i+1}}$.

If l_1 is the label of the rule of π_{A_1}, then let $\pi_1 = \pi_{A_1}$. Otherwise, $l_1 = \delta(B_1)$, and let π_1 be the direct sub-proof of π_{A_1} with conclusion B_1.

Let also $\pi_n = \pi_{A_{n-1}}$. Then $\pi_1 \prec \pi_n$. Furthermore, either $l_i = \delta(A_i)$ with A_i the conclusion of π_i or l_i is the label in the rule of π_i with $i \in \{1, n\}$. □

Lemma 17. *Let l and l' be two labels. Let S be a set of causal graphs. Let also*

$$S' = \{ G[l \mapsto l'] \mid G \in S \}$$

Then $\prod S' = \prod S[l \mapsto l']$. □

Proof. Pick an edge (v'_1, v'_n) in $\prod S'$. Then there are edges and graphs such that $(v'_1, v'_2) \in G_1[l \mapsto l']$, ..., $(v'_{n-1}, v'_n) \in G_{n-1}[l \mapsto l']$ and $G_i \in S$. Hence, there are edges $(v_1, v_2) \in G_1$, ..., $(v_{n-1}, v_n) \in G_{n-1}$ such that $v'_i = v_i[l \mapsto l']$ for $1 \leq i \leq n$, and consequently $(v_1, v_n) \in \prod S$ and $(v'_1, v'_n) \in \prod S[l \mapsto l']$.

The other way around. Pick an edge (v'_1, v'_n) in $\prod S[l \mapsto 1]$. Similarly, there are edges $(v_1, v_2) \in G_1$, ..., $(v_{n-1}, v_n) \in G_{n-1}$ such that $v'_i = v_i[l \mapsto l']$ for $i \in \{1, n\}$. Hence, there are edges $(v'_1, v'_2) \in G_1[l \mapsto l']$, ..., $(v'_{n-1}, v'_n) \in G_{n-1}[l \mapsto l']$ such that $v'_i = v_i[l \mapsto l']$ for $1 \leq i \leq n$. Consequently $(v'_1, v'_n) \in \prod S'$. □

Proof of Theorem 4.7

Lemma 18. *Let P and Q two, positive programs such that Q is the result of replacing the label l by the label l' in all the in P, that is $Q = P[l \mapsto l']$. Then, it holds that $T_Q^{k-1} = T_P^{k-1}[l \mapsto l']$ for all $0 \leq k \leq \omega$.* □

Proof. In case that $k = 0$, and it follows that $T_Q^k(A) = 0$ and $T_P^k(A) = 0$. Hence, $0 = 0[l \mapsto t]$. Otherwise, the proof follows by induction assuming as induction hypothesis that both condition holds for the case $k-1$.

For any literal A and causal graph $G \leq_{\max} T_P^k(A)$ there are causal graphs $G_1, \ldots, G_m$ such that $G_i \leq_{\max} T_P^{k-1}(B_i)$ and $G = (G_1 * \ldots * G_m) \cdot l_R \cdot \delta(A)$ for some labelled rule $l_R : A \leftarrow B_1, \ldots B_m$.

Hence, if $l_R \neq l$, it follows that $G[l \mapsto l'] = (G_1 * \ldots * G_m)[l \mapsto 1] \cdot l_R \cdot \delta(A)$. Otherwise, $l_R = l$ and it follows that $G[l \mapsto 1] = (G_1 * \ldots * G_m)[l \mapsto l'] \cdot l'_R \cdot \delta(A)$ where $l'_R = l_R[l \mapsto l']$. By Lemma 17, it follows that

$$(G_1 * \ldots * G_m)[l \mapsto l'] \quad = \quad (G_1[l \mapsto l'] * \ldots * G_m[l \mapsto l'])$$

and, by induction hypothesis $G_i[l \mapsto l'] \leq T_Q^{k-1}$. Hence $G[l \mapsto l'] \leq T_Q^k$, and consequently $T_P^k[l \mapsto l'] \leq T_Q^k$.

The other way around. For any literal A and causal graph $G \leq_{\max} T_Q^k(A)$ there are causal graphs $G_1, \ldots, G_m$ such that $G_i \leq_{\max} T_Q^{k-1}(B_i)$ for $1 \leq i \leq m$ and $G = (G_1 * \ldots * G_m) \cdot l'_R \cdot \delta(A)$ for some labelled rule $l_{R,Q} : A \leftarrow B_1, \ldots B_m$. Since $T_Q^{k-1} = T_P^{k-1}[l \mapsto t]$, for each G_i there is a $G'_i \leq T_P^{k-1}(B_i)$ such that $G_i \leq G'_i[l \mapsto l']$. Let $G' = (G'_1 * \ldots * G'_m) \cdot l_R \cdot \delta(A)$. Then $G' \leq T_P^k(A)$. As above

$$G'[l \mapsto 1] \quad = \quad (G'_1[l \mapsto l'] * \ldots * G'_m[l \mapsto l']) \cdot l'_R \cdot \delta(A)$$

so that, $G \leq G'[l \mapsto l']$, and consequently $T_Q^k \leq T_P^k[l \mapsto l']$ and $T_Q^k = T_P^k[l \mapsto l']$. □

Lemma 19. *Let l and l' be two different labels. Let S be a set of causal graphs such that for every causal graph $G \in S$ and every edge $(v,l) \in G$ (resp. $(l,v) \in G$) with $v \neq l'$ there are an edges $(v,l') \in G$ (resp. $(l',v) \in G$). Then*

> *For every edge of the form $(v,l) \in \prod S$ (resp. $(l,v) \in \prod S$) with $v \neq l'$ there is an edge $(v,l') \in \prod S$ (resp. $(l',v) \in \prod S$).*

Proof. Pick $(v,l) \in \prod S$ such that $(v,l') \notin \prod S$. Since $(v,l) \in \prod S$ there are causal graphs $G_1,\ldots,G_{n-1}$ in S and edges $(v_i,v_{i+1}) \in G_i$ such that $v_1 = v$ and $v_n = l$. Note that, by hypothesis, $(v_{n-1},l') \in G_{n-1}$, and consequently $(v,l') \in \prod S$.

Similarly, pick $(l,v) \in \prod S$ such that $(l',v) \notin \prod S$. There are causal graphs $G_1,\ldots,G_{n-1}$ in S and edges $(v_i,v_{i+1}) \in G_i$ such that $v_1 = l$ and $v_n = v$. Note that, by hypothesis, $(l',v_2) \in G_1$, and consequently $(l',v) \in \prod S$. $\square$

Lemma 20. *Let P a positive, head labelled program with a unique atom labelling. Let $l \in Lb$ be a label not in the image of δ and k be an non-negative integer. Then*

> *For every literal A, causal graph $G \leq_{\max} T_P^k(A)$ and label v in P, if the edge $(v,l) \in G$ (resp. $(l,v) \in G$) with $v \neq l$, there is an edge $(v,l') \in G$ (resp. $(l',v) \in G$) where l' is the label assigned by δ to the head of all the rules labelled with l.* $\square$

Proof. In case that $k = 0$, $T_Q^k(A) = 0$, and therefore the statement holds vacuous. Otherwise, the proof follows by induction assuming as induction hypothesis that both condition holds for the case $k-1$.

For any literal A and every causal graph $G \leq_{\max} T_P^k(A)$ there are causal graphs $G_1,\ldots,G_m$ such that $G_i \leq_{\max} T_P^{k-1}(B_i)$ and $G = (G_1 * \ldots * G_m) \cdot l_R \cdot \delta(A)$ for some labelled rule $l_R : A \leftarrow B_1,\ldots B_m$.

If $l \notin G_i$ for all $1 \leq i \leq m$ and $l \neq l_R$, then $l \notin G$, so that the statement holds vacuous.

If $l \in G_i$ for some $1 \leq i \leq m$, by induction hypothesis, for each G_i and edge $(v,l) \in G_i$ (resp. (l,v)) there is an edge $(v,l') \in G_i$ (resp. $(l',v) \in G_i$). Hence, by Lemma 19, for every edge (v,l) (resp. (l,v) in $G_1 * \ldots * G_m$, there is an edge (v,l') (resp. (l',v)).

Suppose there is an edge $(v,l) \in G$ such that $(v,l') \notin G$. From the above statement, it follows that $(v,l) \notin G_1 * \ldots * G_m$. Furthermore l is not in the image of δ, so that $l = l_R$, and consequently $l' = \delta(A)$. But for every vertex $v \in G$ there is an edge $(v,l') \in G$ which is a contradiction with the assumption.

Suppose there is an edge $(l,v) \in G$ such that $(l',v) \notin G$. In the same way as above, $l = l_R$ and the only edge of the form $(l,v) \in G \backslash (G_1 * \ldots * G_n)$ are (l,l) and (l,l'). The edge (l',l) is exclude by hypothesis and the edge $(l',l') \in G$ because it is reflexive closed.

Suppose now that there exists some literal A' and graph $G' \leq_{\max} T_P^k(A')$ such that $(v,l) \in G'$ but $(v,l') \notin G'$. In the same way as above, $l = l_R$ and, since the program is head labelled, it follows that $A = A'$ and $l' = \delta(A)$. Hence $(v,l') \in G'$ following the same reasoning as above. The same applies to an edge of the form (l,v). □

Lemma 21. *Let l and l' be two different labels. Let S be a set of causal graphs such that for every causal graph $G \in S$ and every edge $(v,l) \in G$ (resp. (l,v)) with $v \neq l$ there are an edges $(v,l') \in G$ (resp. $(l',v) \in G$). Let also*

$$S' \;=\; \{\; G[l \mapsto 1] \;\big|\; G \in S \;\}$$

Then $\prod S' = \prod S[l \mapsto 1]$. □

Proof. Pick an edge $(v_1,v_n) \in \prod S'$. Then there are edges and graphs such that $(v_1,v_2) \in G_1[l \mapsto 1]$, …, $(v_{n-1},v_n) \in G_{n-1}[l \mapsto 1]$ and $G_i \in S$. Note that $G_i[l \mapsto 1] \subseteq G_i$. Then $(v_i,v_{i+1}) \in G_i$ for for $1 \leq i \leq n-1$, and then $(v_1,v_n) \in \prod S$, Note furthermore that, since $(v_i,v_{i+1}) \in G_i[l \mapsto 1]$ neither $v_1 = l$ nor $v_n = l$ holds, Consequently the edge $(v_i,v_{i+1}) \in \prod S[l \mapsto 1]$.

The other way around. Pick an edge $(v_1,v_n) \in \prod S[l \mapsto 1]$. Then there are edges and graphs $(v_1,v_2) \in G_1, \ldots, (v_{n-1},v_n) \in G_{n-1}$ such that $G_i \in S$. We may assume without lose of generality that $v_i \neq v_j$ for $1 \leq i < j \leq n+1$. Furthermore, since $(v_1,v_n) \in \prod S[l \mapsto 1]$, neither $v_1 = l$ nor $v_n = l$. Suppose that $(v_1,v_n) \notin \prod S'$. Then there is some $(v_i,v_{i+1}) \notin G_i[l \mapsto 1]$. That is, either $v_i = l$ or $v_{i+1} = l$.

If $v_i = l$, then $(l',v_{i+1}) \in G_i$, and then $(l',v_{i+1}) \in G_i[l \mapsto 1]$. Furthermore, the edge $(v_{i-1},v_i) \in G_{i-1}$, and then $(v_{i-1},l') \in G_{i-1}$. Hence $(v_{i-1},l') \in G_{i-1}[l \mapsto 1]$. Then $(v_{i-1},v_{i+1}) \in \prod S'$, and consequently $(v_1,v_n) \in \prod S'$ too. Otherwise, $v_{i+1} = l$, following the same reasoning it holds that $(v_i,v_{i+2}) \in \prod S'$, and consequently $(v_1,v_n) \in \prod S'$ too, which is a contradiction. □

Lemma 22. *Let P and Q two, positive head labelled programs over a signature $\langle Lit, Lb, \delta \rangle$ such that δ maps each literal to a different label and different of any label in P and Q, and such that Q is the result of unlabelled the rules with label l in P, that is $Q = P[l \mapsto 1]$. Then, it holds that $T_Q^{k-1} = T_P^{k-1}[l \mapsto 1]$.* □

Proof. In case that $k = 0$, and it follows that $T_Q^k(A) = 0$ and $T_P^k(A) = 0$. Hence, $0 = 0[l \mapsto t]$. Otherwise, the proof follows by induction assuming as induction hypothesis that both condition holds for the case $k-1$.

For any literal A and causal graph $G \leq_{\max} T_P^k(A)$ there are causal graphs $G_1, \ldots, G_m$ such that $G_i \leq_{\max} T_P^{k-1}(B_i)$ and $G = (G_1 * \ldots * G_m) \cdot l_R \cdot \delta(A)$ for some labelled rule $l_R : A \leftarrow B_1, \ldots B_m$.

Hence, if $l_R \neq l$, it follows that $G[l \mapsto 1] = (G_1 * \ldots * G_m)[l \mapsto 1] \cdot l_R \cdot \delta(A)$. Otherwise, $l_R = l$ and $G[l \mapsto 1] = (G_1 * \ldots * G_m)[l \mapsto 1] \cdot \delta(A)$. By Lemma 20, it follows that for every G_i and edge $(v,l) \in G$ (resp. $(l,v) \in G$) with $v \neq l$, there is an edge $(v,l') \in G$ (resp. $(l',v) \in G$) where $l' = \delta(A)$. Therefore, by Lemma 21, it follows that

$$(G_1 * \ldots * G_m)[l \mapsto 1] \quad = \quad (G_1[l \mapsto 1] * \ldots * G_m[l \mapsto 1])$$

and, by induction hypothesis $G_i[l \mapsto 1] \leq T_Q^{k-1}$. Hence $G[l \mapsto 1] \leq T_Q^k$, and consequently $T_P^k[l \mapsto 1] \leq T_Q^k$.

The other way around. For any literal A and causal graph $G \leq_{\max} T_Q^k(A)$ there are causal graphs $G_1, \ldots, G_m$ such that $G_i \leq_{\max} T_Q^{k-1}(B_i)$ for $1 \leq i \leq m$ and $G = (G_1 * \ldots * G_m) \cdot l_{R,Q} \cdot \delta(A)$ for some labelled rule $l_{R,Q} : A \leftarrow B_1, \ldots B_m$. Since $T_Q^{k-1} = T_P^{k-1}[l \mapsto 1]$, for each G_i there is a $G_i' \leq T_P^{k-1}(B_i)$ such that $G_i \leq G_i'[l \mapsto 1]$. Let $G' = (G_1' * \ldots * G_m') \cdot l_{R,P} \cdot \delta(A)$. Then $G' \leq T_P^k(A)$. Note that $l_{R,Q} = l_{R,P}[l \mapsto 1]$, that is, $l_{R,Q} = l_{R,P}$ if $l_{R,P} \neq l$ and 1 otherwise. As above

$$G'[l \mapsto 1] \quad = \quad (G_1'[l \mapsto 1] * \ldots * G_m'[l \mapsto 1]) \cdot l_{R,Q} \cdot \delta(A)$$

so that, $G \leq G'[l \mapsto 1]$, and consequently $T_Q^k \leq T_P^k[l \mapsto 1]$ and $T_Q^k = T_P^k[l \mapsto 1]$. □

Lemma 23. *Let P and Q two, positive head labelled programs over a signature $\langle Lit, Lb, \delta \rangle$ such that δ maps each literal to a different label and different of any label in P and Q, and such that Q is the result of replacing the label of l in P by $t \in Lb \cup \{1\}$, with t not in the image of δ, that is $Q = P[l \mapsto t]$. Let I and J be respectively the least models of P and Q. Then, it holds that $J = I[l \mapsto t]$.* □

Proof. Note that, from Theorem 4.1, I and J are respectively T_P^ω and T_Q^ω. After this observation, if $t \in Lb$, then the result follows directly from Lemma 18. Otherwise, $t = 1$, and then the result follows directly from Lemma 22. □

Proof of Theorem 4.7. Let $Q = P[l \mapsto t]$. If I is an stable model of P, then, by definition, I is the least model of P^I. Furthermore $Q^I = P^I[l \mapsto t]$, and therefore, by Lemma 23, $J = I[l \mapsto t]$ is the least model of Q^I. Note furthermore that $I(A) = 0$ iff $I[l \mapsto t] = 0$, and therefore $Q^I = Q^J$, so that J is a stable model of P.

Let $J = I'$. If J is an stable model of $Q = P[l \mapsto t]$, then J is the least model of Q^J and, by Lemma 22, the least model I of P^J holds $J = I[l \mapsto t]$ and $J(A) = 0$ iff $I(A) = 0$. Therefore $P^I = P^J$, and consequently I is a causal stable model of

P. Furthermore, since $J(A) = 0$ iff $I(A) = 0$, if J is an answer set, then I is an answer set.

Finally note that if I a answer set, it is an stable model and the statement holds. Otherwise $I(A) = 1$ for all atom A, and $I[l \mapsto t] = I$ will still be an inconsistent answer set of $P[l \mapsto t]$. □

Proof of Proposition 4.1

Proof of Proposition 4.1. Just note that if I is an stable model then $\Gamma_P(I) = I$ and consequently also $\Gamma^2_P(I) = I$. That is, I is a fixpoint of Γ^2_P and consequently it must be between the least and the greatest fixpoint $\mathtt{lfp}(\Gamma^2_P) \leq I \leq \mathtt{gfp}(\Gamma^2_P)$. □

Proof of Theorem 4.8

Proof of Theorem 4.8. Let I be an stable model of P such that G is a sufficient cause of A w.r.t. I and let P' be a completely labelled program with the same rules than P and let ϱ the replacing necessary to transform P' in P. That is, $P'[\varrho] = P$. From Theorem 4.7, there is a exactly one stable model I' of P' such that $I'[\varrho] = I$.

Furthermore, from Theorem 3.4, $I'(A)$ may be rewritten as $I'(A) = \sum_{G \in S} G$ for some set S of incomparable causal graphs. Moreover, from Theorem 4.6, it follows that every cause G of A w.r.t. I' holds $G = cgraph(\pi')$ for some non-redundant proof π of A. That is, we may rewrite the above term as $I'(A) = \sum_{\pi' \in \Pi} cgraph(\pi')'$ for some set of non-redundant proofs Π' w.r.t. P'. Hence $I(A) = I'(A)[\varrho] = \sum_{\pi' \in \Pi'} cgraph(\pi')[\varrho]$ and $G = cgraph(\pi')[\varrho]$ for some non-redundant proof π' of A w.r.t. P'. Note that, but for labels P and P' are the same program, so that a proof in non-redundant w.r.t. P' iff it is w.r.t. P. Consequently $G = cgraph(\pi)$ for some non-redundant proof $\pi = \pi'[\varrho]$ of A w.r.t. P.

Suppose that $cgraph(\pi)$ is not acyclic, that is, it contains a non-reflexive cycle formed by the $(l_1,l_2),(l_2,l_3),\ldots,(l_{n-1},l_n)$. Since $cgraph(\pi')$ is transitive close by definition, it follows that (l_1,l_n) and (l_n,l_1) are edges of $cgraph(\pi')$. Since, furthermore P' is head labelled, by Lemma 16 it follows that there are proofs π_1, π'_1, π_n, and π'_n, such that π_1 is a sub-proof of π_n, π'_n is a sub-proof of π'_1, l_1

is a label either in the rule or in the consequent of π_1 and π'_1 and l_n is a label either in the rule or in the consequent of π_n and π'_n.

Note that, since π_1 and π'_1 has the same label and P' is completely labelled it follows that $\pi_1 = \pi'_1$. Similarly, $\pi_n = \pi_n$. Hence π_1 is a sub-proof of π_n and vice-versa which is a contradiction. Consequently, $cgraph(\pi)$ is acyclic. □

Proof of Theorem 4.9

Proof of Theorem 4.9. Let $Q = P'$ in the sake of clarity. Recall that by definition $\Gamma_P(I)$ and $\Gamma_{P'}(J)$ are respectively the least model of P^I and Q^J. If $I = J^{cl}$, then $I(A) = 0$ iff $J(A) = 0$ for all literal A and consequently Q^J is the unlabelled version of P^I. Then, by Theorem 4.3, it follows that the least models $\Gamma_P(I)$ and $\Gamma_{P'}(J)$ respectively of P^I and Q^J hold that $\Gamma_{P'}(J) = \Gamma_P(I)^{cl}$. Hence, it also holds that $\Gamma^2_{P'}(J) = \Gamma^2_P(I)^{cl}$.

Then note that $\mathbf{0} = \mathbf{0}^{cl}$ and so $\Gamma^{2i}_{P'}(J) = \Gamma^{2i}_P(I)^{cl}$ for all non-negative integer $i \geq 0$. That is, $\texttt{lfp}(\Gamma^2_{P'}) = \texttt{lfp}(\Gamma^2_P)^{cl}$. Finally note that $\texttt{gfp}(\Gamma^2_{P'}) = \Gamma_{P'}(\texttt{lfp}(\Gamma^2_{P'})) = \Gamma_P(\texttt{lfp}(\Gamma^2_P)^{cl})^{cl} = \texttt{gfp}(\Gamma^2_P)^{cl}$.

Recall furthermore that the reduct of a program with respect to a causal interpretation I and with respect to its two-valued version I^{cl} is the same. Hence, $\Gamma_P(I) = \Gamma_P(I^{cl})$ holds for every interpretation I. Then,

$$\texttt{lfp}(\Gamma^2_P) \;=\; \Gamma_P(\texttt{gfp}(\Gamma^2_P)) \;=\; \Gamma_P(\texttt{gfp}(\Gamma^2_P)^{cl}) \;=\; \Gamma_P(\texttt{gfp}(\Gamma^2_{P'})$$

Similarly, it can be shown that $\texttt{gfp}(\Gamma^2_P) = \Gamma_P(\texttt{lfp}(\Gamma^2_{P'}))$. That is, we can obtain the causal well-founded model of a program just by apply the Γ^2_P one time to its standard well-founded model. □

Proof of Theorem 4.10

Proof of Theorem 4.10. Let $I = \texttt{gfp}(\Gamma^2_P)$. Then $\texttt{lfp}(\Gamma^2_P) = \Gamma_P(I)$ is the least model of P^I. If π is a proof of A with respect to $\texttt{gfp}(\Gamma^2_P)$ and P, then, by Definition 4.9, π is a proof of A with respect to P^I and, from Theorem 4.2, it follows that $cgraph(\pi)$ is a cause of A with respect to the least model P^I, that is $\texttt{lfp}(\Gamma^2_P)$. Hence, by Definition 4.15, $cgraph(\pi)$ is a cause of A under the well-founded semantics.

The other way around. If $cgraph(\pi)$ is a cause of A under the well-founded semantics, then $cgraph(\pi)$ is a cause of A with respect to the least model P^I and from Theorem 4.2, it follows that π is a proof of A with respect to P^I. Note that every rule R in P^I holds that $I \models body^-(R)$, and consequently π is a proof of A with respect to $I = \mathtt{gfp}(\Gamma_P^2)$ and P. □

CHAPTER 6: CAUSAL LITERALS

Proof of Proposition 6.1

Proof of Proposition 6.1. For the if direction. Assume there is some sufficient cause G of A w.r.t. I such that $\psi(G) = 1$. Since G is a sufficient cause of A w.r.t. I, it follows that $G \leq I(A)$ and, then, since $\psi(G) = 1$, it follows that $I(\psi :: A) \geq G$. That is, $I(\psi :: A) \neq 0$.

For the only if direction. Assume that $I \models (\psi :: A)$, then there is $G' \leq I(A)$ such that $\psi(G') = 1$. Let G be a causal graph such that $G' \leq G \leq_{\max} I(A)$. Then G is a sufficient cause of A w.r.t. I and, since ψ is monotonic and $G' \leq G$, it follows that $\psi(G) = 1$. □

Proof of Proposition 6.2

Proof of Proposition 6.2. Let S be a directed set of causal values, that is, for every pair of values $\{u_1, u_2\} \subseteq S$ there is a value $u_3 \in S$ such that $u_1 \leq u_3$ and $u_2 \leq u_3$. Let also u_S be the supremum of S, that is, $u_S \stackrel{\text{def}}{=} \sum_{u \in S} u$.

Pick $G' \leq f_\psi(u_S)$. Then $G' \leq u_S$ and $\psi(G') = 1$ and, hence, $G' \leq u$ for some $u \in S$. Hence $G' \leq f_\psi(u)$ and, therefore $G' \leq \sum_{u \in S} f(u)$ for all $G' \leq f_\psi(u_S)$, that is, $f_\psi(u_S) \leq \sum_{u \in S} f_\psi(u)$. The other way around, pick $G' \leq f_\psi(u)$ for some $u \in S$ and let u' any causal value such that $u \leq u'$. Since $G' \leq f_\psi(u)$ it follows that $\psi(G') = 1$ and $G' \leq u \leq u'$ and, therefore $G' \leq f_\psi(u')$, i.e. f_ψ is monotonic. Then $u_S \geq u$ implies $f_\psi(u_S) \geq \sum_{u \in S} f_\psi(u)$, and therefore, $f_\psi(u_S) = \sum_{u \in S} f_\psi(u)$. That is f_ψ is monotonic and continuous.

Finally note that by definition

$$f_\psi(I(A)) \stackrel{\text{def}}{=} \sum \{\, G \in \mathbf{C}_{Lb} \mid G \leq I(A) \text{ and } \psi(G) = 1 \,\}$$

which, by Definition 6.3, is equal to $I(\psi :: A)$. □

Proof of Proposition 6.4

Proof of Proposition 6.4. Note that, by Definition 6.7,

$$f_R\big(I(B_1),\ldots,I(B_m)\big) \ \leq\ I(A)$$

iff

$$\big(f_1(I(B_1)),\ldots,f_m(I(B_m))\big)\cdot l_R\cdot\delta(A) \ \leq\ I(A)$$

and, form Proposition 6.2, iff

$$\big(\ I(\psi_1 :: B_1) * \ldots * I(\psi_1 :: B_m)\ \big)\cdot l_R\cdot\delta(A) \ \leq\ I(A)$$

which, by Definition 4.2, holds iff $I \models R$. □

Proof of Proposition 6.5

Proof of Proposition 6.5. Note that, from Proposition 6.4, it follows that $I \models R$ iff $f_R(\ I(B_1),\ldots,I(B_m)\ ) \leq I(A)$ and $I \models_r R$ iff $f_R(\mu_1,\ldots,\mu_m) \leq I(A)$ for all $\mu_i \leq I(A)$.

Since $I(B_i) \leq I(B_i)$ for all lB_i, by taking $\mu_i = I(B_i)$, it is clear that that $I \models_r R$ implies $I \models R$.

The other way around, let μ_i any causal value such that $\mu_i \leq I(A_i)$ and assume that $I \models R$. Since f_R is monotonic (Proposition 6.3), it follows that

$$f_R(\mu_1,\ldots,\mu_m) \leq f_R\big(\ I(A_1),\ldots,I(A_m)\ \big)$$

and, since $I \models R$, it follows that $f_R\big(\ I(B_1),\ldots,I(B_m) \leq I(A)$. Hence $I \models_r R$. □

Proof of Proposition 6.6

Proof of Proposition 6.6. Note that $S_i \overset{\text{def}}{=} \{\mu_i \mid \mu_i \leq I(B_i)\}$ has the clearly upper bound $I(B_i)$, it is a directed set. Hence, since f_R is continuous, it follows that

$$\begin{aligned}\sum_{\mu_1 \in S_1, \ldots, \mu_m \in S_m} f_R(\mu_1, \ldots, \mu_m) &= f_R\Big(\sum_{\mu_1 \in S_1} \mu_1, \ldots, \sum_{\mu_m \in S_m} \mu_m\Big) \\ &= f_R(I(B_1), \ldots, I(B_m))\end{aligned}$$

and, thus, $T_P(I)(A)$ is equal to

$$\sum \{\, f_R(\mu_1, \ldots, \mu_m) \mid R \in P \,,\, head(R) = A \text{ and } \mu_1 \leq I(B_1), \ldots, \mu_m \leq I(B_m) \,\}$$

which in its turn is the definition of $R_P(I)(A)$. □

Proof of Proposition 6.7

Lemma 24. *Let P be a positive uniquely labelled causal program, A be a literal, $k \in \{0, \ldots, \omega\}$ be an ordinal and G be a causal graph. Let λ be the number of unlabelled rules in P. If $G \leq_{\max} T_P^k(A)$ and $h = height(G) + \lambda \leq k$ then $G \leq T_P^h(A)$.* □

Proof. In case that $h = 0$, G must be the empty graph. Hence, if $G \leq_{\max} T_P^k(A)$, there must be an unlabelled rule of the form $R = (A \leftarrow B_1, \ldots, B_m)$ such that $T_P^{k-1}(B_i) = i$ for $1 \leq i \leq m$. But, if there is some unlabelled rule, then $\lambda \geq 1$ and so $h \geq 1$, which is a contradiction with the assumption that $h = 0$.

In case that $h \geq 1$, we proceed by induction assuming as hypothesis that the lemma statement holds for any $h' < h$. If $G \leq_{\max} T_P^k(A)$, there must be an causal rule of the form $R = (l_R : A \leftarrow B_1, \ldots, B_m)$ such that $G_i \leq_{\max} T_P^{k-1}(B_i)$ for $1 \leq i \leq m$ and $G = G_R \cdot l_R \cdot \delta(l_R)$ with $G_R = G_1 * \ldots * G_m$.

If $m = 0$ then $G_R = 1$, and then $G = l_R \cdot \delta(A)$, and then $height(G) \leq 2$. Note that R is then a fact, and therefore $G \leq T_P^i(A)$ for all $i \geq 1$. Since $h \geq 1$, it follows that $G \leq T_P^h(A)$.

Otherwise, if $l_R = 1$, let $\lambda' = \lambda - 1$ and $h'_i = height(G_i) + \lambda'$ for $1 \leq i \leq m$ and $Q = P \backslash \{R\}$. Then $h = \max\{h'_1, \ldots, h'_m\} + 1$ and λ' the number of unlabelled rule in Q, and then, by induction hypothesis, $G_i \leq T_Q^{h_i}(B_i) \leq T_P^{h_i}(B_i)$. Consequently $G \leq T_P^h(A)$.

If otherwise $l_R \neq 1$. Let $h'_i = \mathit{height}(G_i) + \lambda$ for $1 \leq i \leq m$. Then, since any path in G_i is also a path G, it is clear that $\mathit{height}(G_i) \leq \mathit{height}(G)$, and therefore $h'_i \leq h$. Suppose that $h'_i = h$ for some G_i. Then there is a simple path $l_1, \dots, l_{h-\lambda}$ of rule labels of length $h - \lambda$ in G_i. Since $l_R \neq 1$, since $G = G_R \cdot l_R \cdot \delta(A)$, there is an edge $(l_{h-\lambda}, l_R) \in G$. That is $l_1, \dots, l_{h-\lambda}, l_R$ is a walk of length $h - \lambda + 1$ in G. Furthermore, since $l_R \notin G_i$ because the program is uniquely labelled, it follows that $l_j \neq l_R$ with $1 \leq j \leq h - \lambda$. Hence $l_1, \dots, l_{h-\lambda}, l_R$ is a simple path of length $h - \lambda + 1$ which contradicts the assumption that $\mathit{height}(G) = h - \lambda$.

Thus $h'_i < h$ for all G_i and then, by induction hypothesis, $G_i \leq T_P^{h'_i}(B_i)$. Hence $G \leq T_P^{h'}(A)$ where $h' = 1 + \max\{h'_1, \dots, h'_m\}$. Furthermore $h' \leq h$, and consequently $G \leq T_P^{h'}(A) \leq T_P^h(A)$. $\square$

In case that $k = \omega$, by definition $T_P^\omega(A) = \sum_{i<\omega} T_P^i(A)$. Thus, if $G \leq_{\max} T_P^\omega(A)$ and $\mathit{height}(G) = h$ then there is some $i < \omega$ s.t. $G \leq_{\max} T_P^i(A)$ and $h \leq i$, and as we already show, then $G \leq T_P^h(A)$. $\square$

Proof of Proposition 6.7. Let Q be a uniquely labelled program with the same rules than P but for the labels such that for every rule $R_i \in P$

- If R_i is unlabelled in P, then R_i is unlabelled in Q.
- Otherwise, R is labelled in Q with a label l_R different from any other rule in Q.

Then P is the result of replacing in Q each label l_R by the original label it has in P. Let ϱ such replacing, that is $P = Q[\varrho]$. Then, from Lemma 18, it follows that $T_P^k = T_Q^k[\varrho]$ for all $0 \leq k \leq \omega$. Furthermore, since Q is a uniquely labelled program, from Lemma 24, it follows that $G \leq T_Q^h(A) = T_P^h(A)$ holds whenever $G \leq_{\max} T_Q^k(A) = T_P^k(A)$. $\square$

CHAPTER 7: QUERIES AND COMPLEXITY

Proof of Proposition 7.1

Proof of Proposition 7.1.

1. If a vertex v is labelled with the operation $(*)$ or $(\cdot)$ and one of its children v_i is labelled with 0, then

$$\begin{aligned} term(\tilde{T},v) &= term(\tilde{T},v_1) * \ldots * term(\tilde{T},v_i) * \ldots * term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) * \ldots * \quad 0 \quad * \ldots * term(\tilde{T},v_1) \\ &= 0 \end{aligned}$$

   Hence, the label of the vertex v may be replaced by 0 and their children removed. Check whether the label of some children is 0 is clearly feasible in polynomial time.

2. If v is labelled with the operation $(+)$ and one of its children is labelled with 0, then

$$\begin{aligned} term(\tilde{T},v) &= term(\tilde{T},v_1) + \ldots + term(\tilde{T},v_i) + \ldots + term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) + \ldots + \quad 0 \quad + \ldots + term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) + \ldots \qquad + term(\tilde{T},v_1) \end{aligned}$$

   and consequently the child labelled with 0 may be removed.

3. If a vertex v is labelled with the operation $(*)$ and one of its children is labelled with 1, then

$$\begin{aligned} term(\tilde{T},v) &= term(\tilde{T},v_1) * \ldots * term(\tilde{T},v_i) * \ldots * term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) * \ldots * \quad 1 \quad * \ldots * term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) * \ldots \qquad * term(\tilde{T},v_1) \end{aligned}$$

   and the child may be removed.

4. If a vertex v labelled with the operation $(*)$ and its has no children, $term(\tilde{T},v) = \prod\emptyset = 1$, that is, it may be replaced by 1.

5. If a vertex v is labelled with the operation $(\cdot)$ and one of its children is labelled with 1, then either $term(\tilde{T},v) = 1 \cdot term(\tilde{T},v_r) = term(\tilde{T},v_r)$ or $term(\tilde{T},v) = term(\tilde{T},v_l) \cdot 1 = term(\tilde{T},v_l)$, that is, the last may be removed and v replaced to the other children.

6. If a vertex v is labelled with the operation $(+)$ and one of its children is labelled with 1, then

$$\begin{aligned} term(\tilde{T},v) &= term(\tilde{T},v_1) + \ldots + term(\tilde{T},v_i) + \ldots + term(\tilde{T},v_1) \\ &= term(\tilde{T},v_1) + \ldots + \quad 1 \quad + \ldots + term(\tilde{T},v_1) \\ &= \quad 1 \end{aligned}$$

and then the label of v may be replaced by 1 and all its children removed.

7. If the root v is labelled with 0, then it may be removed because $\sum \emptyset = 0$.

It is easy to see that all the these checks and their correspondent transformations are feasible in polynomial time. □

Proof of Proposition 7.3

Proof of Proposition 7.3. The fact that $\tilde{T}_P(\tilde{I}) = T_P(I)$ is immediately from the fact that operators $\sum$ and $\prod$ and $(\cdot)$ in Definition 4.3 are just mapped into they corresponding labelled vertex.

To see that $\tilde{T}_P(\tilde{I})$ is computable in polynomial time, note that $\sum S$ and $\prod S$ over a set of t-graphs S is feasible in polynomial time. In fact, it is only necessary a linear number of set unions. Compute $\tilde{T} \cdot l$ for some label l only takes constant number of set unions. □

Proof of Proposition 7.6

Proof of Proposition 7.6. The proof follows by structural induction assuming that it holds for every children of a vertex and showing that then it holds for such vertex.

1. If $f_V(v) = u \in Lb \cup \{1\}$ and $(w_u, w_l) \in G$ with $\psi_{Ac}(u) = w_u$ and $\psi_{Ac}(l) = w_l$.
Then $term(\tilde{T},v) = u$ and then $term(\tilde{T},v) \cdot l = u \cdot l$.
Note also that $u \cdot l \in \mathbf{C}_{Lb}$.
Hence $G' = u \cdot l$ is the only causal graph that holds $G' \leq_{\max} term(\tilde{T},v) \cdot l$.
Since $(w_u, w_l) \in G$, then $G \leq \psi_{Ac}(G') = w_u \cdot w_l$
Hence $caused(Ac, G, \tilde{T}, v, l) = G' = u \cdot l = \tilde{T}^l(\tilde{T}^u)$.

2. If $f_V(v) \in \{\cdot\}$ and $(w_u, w_l) \in G$ with $\{(v,v_l)^l, (v,v_r^u)^r\} \subseteq E(\tilde{T})$ and $f_V(v_r) = u$, $\psi_{Ac}(u) = w_u$ and $\psi_{Ac}(l) = w_l$.
By induction hypothesis,

$$\begin{aligned} caused(G,\tilde{T},v_l,u) \ =& \\ &\textstyle\sum\{G' \mid G \leq \psi_{Ac}(G') \text{ and } G' \leq term(\tilde{T},v_l)\cdot u\} \end{aligned} \tag{112}$$

and by application monotonicity

$$\begin{aligned} caused(G,\tilde{T},v_l,u)\cdot l \ = \ \textstyle\sum\{\ G' \mid G \leq \psi_{Ac}(G') \\ \text{and } G' \leq term(\tilde{T},v_l)\cdot u\ \}\cdot l \end{aligned}$$

Note that, by the above *caused* definition

$$caused(G,\tilde{T},v,l) = \tilde{T}^l(caused(G,\tilde{T},v_l,u)) = caused(G,\tilde{T},v_l,u)\cdot l$$

so that, (112) can be rewritten as

$$caused(G,\tilde{T},v,l) \ = \ \textstyle\sum\{\ G' \mid G \leq \psi_{Ac}(G') \text{ and } G' \leq term(\tilde{T},v_l)\cdot u\ \}\cdot l$$

which, since application distributes over sum can be rewritten as

$$caused(G,\tilde{T},v,l) \ = \ \textstyle\sum\{\ G'\cdot l \mid G \leq \psi_{Ac}(G') \text{ and } G' \leq term(\tilde{T},v_l)\cdot u\ \}$$

Furthermore, written down $term(\tilde{T},v_l)\cdot u$ as a sum of causal graphs, it follows that $term(\tilde{T},v_l)\cdot u = G_1\cdot u + \ldots + G_n\cdot u$.
We then assume, without of generality, that $G' \leq_{\max} term(\tilde{T},v_l)\cdot u$.
Then $G' = G_i\cdot u$ for some G_i, and then $G'\cdot l = G_i\cdot u\cdot l = G_i\cdot u * u\cdot l$.
Hence $G \leq \psi_{Ac}(G'\cdot l)$
iff $G \leq \psi_{Ac}(G_i\cdot u * u\cdot l)$
iff $G \leq \psi_{Ac}(G_i\cdot u)$ and $G \leq \psi_{Ac}(u\cdot l)$.
Note that, by assumption $(w_u, w_l) \in G$, so that, $G \leq \psi_{Ac}(u\cdot l)$, and then
$G \leq \psi_{Ac}(G_i\cdot u)$ and $G \leq \psi_{Ac}(u\cdot l)$
iff $G \leq \psi_{Ac}(G_i\cdot u)$
iff $G \leq \psi_{Ac}(G')$.
That is, $G \leq \psi_{Ac}(G'\cdot l)$ iff $G \leq \psi_{Ac}(G')$ and consequently

$$caused(G,\tilde{T},v,l) \ = \ \textstyle\sum\{\ G'\cdot l \mid G \leq \psi_{Ac}(G'\cdot l) \text{ and } G' \leq term(\tilde{T},v_l)\cdot u\ \}$$

In addition, by application monotonicity, it follows that
$G' \leq term(\tilde{T},v_l)\cdot u$ implies $G'\cdot l \leq term(\tilde{T},v_l)\cdot u\cdot l$. Then

$$\begin{aligned} caused(G,\tilde{T},v,l) \ \leq \ \textstyle\sum\{\ G'\cdot l \mid G \leq \psi_{Ac}(G'\cdot l) \\ \text{and } G'\cdot l \leq term(\tilde{T},v_l)\cdot u\cdot l\ \} \end{aligned}$$

Moreover, since $term(\tilde{T},v_l)\cdot u\cdot l \le term(\tilde{T},v_l)\cdot u$, it follows that $G'\cdot l \le term(\tilde{T},v_l)\cdot u\cdot l$ implies $G'\cdot l \le term(\tilde{T},v_l)\cdot u$.
and, since $G'\cdot l\cdot l = G'\cdot l$, then it follows that

$$\begin{aligned} caused(G,\tilde{T},v,l) \;=\; \textstyle\sum\{\; G'\cdot l \;\big|\; & G \le \psi_{Ac}(G'\cdot l) \\ & \text{and } G'\cdot l \le term(\tilde{T},v_l)\cdot u\cdot l \;\} \end{aligned}$$

Note also that, $term(\tilde{T},v_l)\cdot u\cdot l = term(\tilde{T},v)\cdot l$, and then

$$caused(G,\tilde{T},v,l) \;=\; \textstyle\sum\{\; G'\cdot l \;\big|\; G \le \psi_{Ac}(G'\cdot l) \text{ and } G'\cdot l \le term(\tilde{T},v)\cdot l \;\}$$

and rewritten $G'\cdot l$ as G'', it follows that

$$caused(G,\tilde{T},v,l) \;=\; \textstyle\sum\{\; G'' \;\big|\; G \le \psi_{Ac}(G'') \text{ and } G'' \le term(\tilde{T},v)\cdot l \;\}$$

Note that, $term(\tilde{T},v)\cdot l = G'_1\cdot l + \ldots + G'_n\cdot l$, so that, $G'' \le_{\max} term(\tilde{T},v)\cdot l$ implies $G'' = G'_i\cdot l$ for some G'_i.

3. If $f_V(v) \in \{+\}$, then by definition

$$caused(Ac,G,\tilde{T},v,l) \overset{\text{def}}{=} \tilde{T}^{+}\ \{\ caused(G,\tilde{T},v',l) \mid (v,v') \in E(\tilde{T})\ \}$$

that corresponds to

$$caused(Ac,G,\tilde{T},v,l) \overset{\text{def}}{=} \textstyle\sum\{\ caused(G,\tilde{T},v',l) \mid (v,v') \in E(\tilde{T})\ \}$$

Furthermore, by induction hypothesis

$$caused(G,\tilde{T},v',l) \;=\; \textstyle\sum\{G' \mid G \le \psi_{Ac}(G') \text{ and } G' \le term(\tilde{T},v')\cdot l\}$$

for all v' such that $(v,v') \in E(\tilde{T})$. Hence $caused(Ac,G,\tilde{T},v,l)$ is equal to

$$\sum_{(v,v')\in E(\tilde{T})} \textstyle\sum\{G' \mid G \le \psi_{Ac}(G') \text{ and } G' \le term(\tilde{T},v')\cdot l\}$$

which, in its turn, can be rewritten as

$$\sum\{\; G' \;\big|\; G \le \psi_{Ac}(G') \text{ and } G' \le \sum_{(v,v')\in E(\tilde{T})} term(\tilde{T},v')\cdot l \;\}$$

Note that $\sum_{(v,v')\in E(\tilde{T})} term(\tilde{T},v')\cdot l = term(\tilde{T},v)\cdot l$. Hence

$$\begin{aligned} caused(Ac,G,\tilde{T},v,l) \;=\; \textstyle\sum\{\; G' \;\big|\; & G \le \psi_{Ac}(G') \\ & \text{and } G' \le term(\tilde{T},v)\cdot l \;\} \end{aligned}$$

4. The case that $f_V(v) \in \{*\}$ is symmetrical to the case that $f_V(v) \in \{+\}$ until reach that $\mathit{caused}(Ac, G, \tilde{T}, v, l)$ is equal to

$$\prod_{(v,v') \in E(\tilde{T})} \sum \{G' \mid G \leq \psi_{Ac}(G') \text{ and } G' \leq \mathit{term}(\tilde{T}, v') \cdot l\}$$

Then, this equivalent to

$$\bigcap_{(v,v') \in E(\tilde{T})} \Downarrow\{G' \mid G \leq \psi_{Ac}(G')\} \cap \Downarrow\{G' \mid G' \leq \mathit{term}(\tilde{T}, v') \cdot l\}$$

which, in its turn, is equivalent to

$$\Downarrow\{G' \mid G \leq \psi_{Ac}(G')\} \cap \bigcap_{(v,v') \in E(\tilde{T})} \Downarrow\{G' \mid G' \leq \mathit{term}(\tilde{T}, v') \cdot l\}$$

Note that

$$\bigcap_{(v,v') \in E(\tilde{T})} \Downarrow\{G' \mid G' \leq \mathit{term}(\tilde{T}, v') \cdot l\} = \prod_{(v,v') \in E(\tilde{T})} \mathit{term}(\tilde{T}, v') \cdot l$$

and $\prod_{(v,v') \in E(\tilde{T})} \mathit{term}(\tilde{T}, v') \cdot l = \mathit{term}(\tilde{T}, v) \cdot l$. Then

$$\mathit{caused}(Ac, G, \tilde{T}, v, l) \;=\; \sum \{G' \mid G \leq \psi_{Ac}(G') \text{ and } G' \leq \mathit{term}(\tilde{T}, v) \cdot l\}$$

5. Otherwise, $f_V(v) = u \in Lb \cup \{1\}$ or $f_V(v) \in \{\cdot\}$ but $(w_u, w_l) \notin G$. Then there not exists G' such that $G \leq \psi_{Ac}(G')$ and $G' \leq \mathit{term}(\tilde{T}, v) \cdot l$ because every causal graph $G' \leq \mathit{term}(\tilde{T}, v) \cdot l$ must contain the edge (u, l) and therefore $(w_u, w_l) \in \psi_{Ac}(G')$ which would contradict that $G \leq \psi_{Ac}(G')$ ($G \supseteq \psi_{Ac}(G')$) since, by assumption, $(w_u, w_l) \notin G$.

Note furthermore that in order to compute $\mathit{caused}(\mathcal{A}, G, \tilde{T}, v, l)$, each vertex of the t-graph $\tilde{T}$ only needs to be visit one time, and each visit only implies check whether some elements belong to the sets $\mathcal{A}$ and G (causal graphs are represents by the set of their edges) and the application of the constructors. Hence this procedure is feasible in polynomial time. □

Proof of Proposition 7.5

Proof of Proposition 7.5. We recall that, from Proposition 7.4, deciding whether $G \leq \tilde{T}$ is feasible in polynomial time. Furthermore, by definition if $G \leq \tilde{T}$ and $G \not\leq_{\max} \tilde{T}$, then there is a causal graph G' such that $G < G' \leq \tilde{T}$.

Notice that since G is acyclic and finite, its transitive and reflexive reduction G^R is unique [Aho et al., 1972, Theorem 1]. Let $e_1, \ldots, e_m$ the edges of G^R and let $G_1, \ldots, G_m$ causal graphs such that each G_i is the transitive and reflexive closure of $G \backslash \{e_i\}$. It is clear that each $G_i \subseteq G$. Furthermore, if $G_i = G$ for some G_i, then G^R would not be the unique transitive and reflexive reduction of G which is a contradiction with the assumption. Hence $G_i \subset G$, that is $G_i > G$, for all G_i. Moreover, $G' \subset G$ implies $G' \subseteq G \backslash \{e_i\}$ for some e_i, and hence, $G' \subseteq G_i$, that is $G' \geq G_i$ for some G_i. In words, if there exists some G' such that $G < G' \leq \tilde{T}$ then there also exists some G_i such that $G_i \leq \tilde{T}$.

Hence, $G \leq_{\max} \tilde{T}$ iff $G \leq \tilde{T}$ and $G_i \not\leq \tilde{T}$ for all G_i. Therefore, $G \leq_{\max} \tilde{T}$ may be decided by $m + 1$ calls to the procedure to decide whether $G \leq \tilde{T}$, and since $m + 1$ is bounded by the size of G, then it is feasible in polynomial time. $\square$

CHAPTER 8: RELATED WORK

Proof of Theorem 8.1

Proof of Theorem 8.1. Note that the definition of CP direct consequences operator (Definition 8.6) is the same as the definition of the causal direct consequences operator (Definition 4.3). Furthermore, since every CP-term can be transformed in an equivalent in negation normal form, we may replace each atomic CP-term containing a negation by a new label. The result is a causal term and, therefore, by Theorem 4.1 it follows that

1. $\texttt{lfp}(\tilde{T}_P)$ is the least model of P, and
2. $\texttt{lfp}(\tilde{T}_P) = T_P^{\omega}(\tilde{\mathbf{0}})$.
3. If furthermore P is finite and has n rules, then $\texttt{lfp}(\tilde{T}_P) = \tilde{T}_P^{n}(\tilde{\mathbf{0}})$.

Finally, once replaced again the auxiliary labels by their corresponding atomic CP-terms are equivalences are preserved. □

Lemma 25. *Let t be a term. Then $\lambda^p(\sim t) = \neg\lambda^p(t)$.* □

Proof. We proceed by structural induction assuming that t is in negated normal form. In case that $t = a$ is atomic, it follows that $\lambda^p(\sim a) = \neg a = \neg\lambda^p(a)$. In case that $t = \sim a$ with a atomic, $\lambda^p(\sim t) = \lambda^p(\sim\sim a)$ and

$$\lambda^p(\sim\sim a) = a = \neg\neg a = \neg\lambda^p(\sim a) = \lambda^p(t)$$

In case that $t = \sim\sim a$, with a atomic, $\lambda^p(\sim t) = \lambda^p(\sim\sim\sim a)$ and

$$\lambda^p(\sim\sim\sim a) = \lambda^p(\sim a) = \neg a = \neg\lambda^p(\sim\sim a) = \neg\lambda^p(t)$$

In case that $t = u + v$. Then

$$\lambda^p(\sim t) = \lambda^p(\sim u * \sim v) = \lambda^p(\sim u) \wedge \lambda^p(\sim v)$$

By induction hypothesis $\lambda^p(\sim u) = \neg\lambda^p(u)$ and $\lambda^p(\sim v) = \neg\lambda^p(v)$. Then, it follows that $\lambda^p(\sim t) = \neg\lambda^p(u) \wedge \neg\lambda^p(v)$. Furthermore, it also holds that $\neg\lambda^p(t) = \neg(\lambda^p(u) \vee \lambda^p(v)) = \neg\lambda^p(u) \wedge \neg\lambda^p(v) = \lambda^p(\sim t)$.

In case that $t = u \otimes v$ with $\otimes \in \{\, *, \cdot \,\}$. Then $\lambda^p(\sim t) = \lambda^p(\sim u + \sim v) = \lambda^p(\sim u) \vee \lambda^p(\sim v)$ and by induction hypothesis $\lambda^p(\sim u) = \neg\lambda^p(u)$ and $\lambda^p(\sim v) = \neg\lambda^p(v)$. Consequently it holds that $\lambda^p(\sim t) = \neg\lambda^p(t)$. □

Lemma 26. *Let t be a term ϕ a provenance term. If $\phi \leq \lambda^p(t)$, then $\lambda^p({\sim}t) \leq \neg\phi$ and if $\lambda^p(t) \leq \phi$, then $\neg\phi \leq \lambda^p({\sim}t)$.* □

Proof. If $\phi \leq \lambda^p(t)$, then $\phi = \lambda^p(t) * \phi$ and then $\neg\phi = \neg\lambda^p(t) + \neg\phi$ and, by Lemma 25, it follows that $\neg\phi = \lambda^p({\sim}t) + \neg\phi$. Hence $\lambda^p({\sim}t) \leq \neg\phi$.

Furthermore if $\lambda^p(t) \leq \phi$, then $\phi = \lambda^p(t) + \phi$ and then $\neg\phi = \neg\lambda^p(t) * \neg\phi$ and, by Lemma 25, it follows that $\neg\phi = \lambda^p({\sim}t) * \neg\phi$. Hence $\neg\phi \leq \lambda^p({\sim}t)$. □

Lemma 27. *Let P be a labelled program, $\tilde{U}$ and V respectively be a CP and a causal interpretation such that $V \leq \lambda^c(\tilde{U})$. Let also $\tilde{I}$ and J respectively be a CP and a causal interpretation such that $J \leq \tilde{I}$. Then $T_{P^V}(J) \leq \tilde{T}_{P^{\tilde{U}}}(\tilde{I})$.* □

Proof. For all $E \leq T_{P^V}(J)(p)$ there is a rule in P^V of the form

$$r_i : p \leftarrow B_1, \ldots, B_m$$

that corresponds to a rule of the form (92) in P and furthermore $E \leq (E_{B_1} * \ldots * E_{B_m}) \cdot r_i$ with each $E_{B_j} \leq J(B_j)$ and $V(C_j) = 0$ for all B_j and C_j in $body(r_i)$. Hence there is a rule in $P^{\tilde{U}}$ of the form (92) and, by hypothesis, $E_{B_j} \leq \tilde{I}(B_j)$ for all B_j. Furthermore, clearly $V(C_j) = 0 \leq {\sim}\tilde{U}(C_j)$ for all C_j. Hence $E \leq (\tilde{I}(B_1) * \ldots * \tilde{I}(B_m) * {\sim}\tilde{U}(C_1) * \ldots * {\sim}\tilde{U}(C_m)) \cdot r_i \leq \tilde{T}_{P^{\tilde{U}}}(\tilde{I})$. □

Lemma 28. *Let P be a labelled logic program, $\tilde{U}$ and V respectively be a CP and a causal interpretation such that $V(p) \leq \lambda^c(\tilde{U})$. Let also $\tilde{I}$ and J respectively be the least model of programs $P^{\tilde{U}}$ and P^V. Then $J \leq \tilde{I}$.* □

Proof. Since $\tilde{I}$ and J are the least models respectively of programs $P^{\tilde{U}}$ and P^V it follows that $\tilde{I} = \tilde{T}^{\omega}_{P^{\tilde{U}}}$ and $J = T^{\omega}_{P^V}$. We proceed by induction on the number of iterations k. When $k = 0$, it follows that $T^0_{P^V}(p) = 0 \leq \tilde{T}^0_{P^{\tilde{U}}}(p)$. When $0 < k < \omega$, we assume as induction hypothesis the statement holds for the case $k-1$. Then, by Lemma 27, it follows for the case k.

Finally when $k = \omega$, for all $E \leq T^{\omega}_{P^V}(p))$ there is some i such that $E \leq T^i_{P^V}(p))$ and then, by induction hypothesis, $E \leq \lambda^c(T^i_{P^V}(p)) \leq \lambda^c(\tilde{T}^{\omega}_{P^{\tilde{U}}}(p))$. Therefore $J \leq \tilde{I}$. □

Lemma 29. *Let P be a labelled program, $\tilde{I}$ and J respectively be a CP and a causal interpretation such that $V(p) \leq \lambda^c(\tilde{U})$. Then $\Gamma(J) \leq \tilde{\Gamma}(\tilde{I})$.* □

Proof. By Lemma 27 it follows that $J' \leq \tilde{I}'$ with $\tilde{I}'$ and J' respectively the least models of $P^{\tilde{I}}$ and $P^{\tilde{J}}$ and by definition $\Gamma(J) = J'$ and $\tilde{\Gamma}(\tilde{I}) = \tilde{I}'$. Hence $\Gamma(J) \leq \tilde{\Gamma}(\tilde{I})$. □

Lemma 30. *Let P be a labelled program, $\tilde{U}$ and V respectively be a CP and a causal interpretation such that $V = \lambda^p(\tilde{U})$. Let also $\tilde{I}$ and J respectively be a CP and a CG interpretation such that $J \geq \lambda^c(\tilde{I})$. Then $T_{P^V}(J) = \lambda^c(\tilde{T}_{P^{\tilde{U}}}(\tilde{I}))$.* □

Proof. For all explanation $F \leq \tilde{T}_{P^{\tilde{U}}}(J)(p)$ there is a rule $r_i^{\tilde{U}}$in $P^{\tilde{U}}$ of the form (102) and so a rule of the form (92) in P and there are justifications $F_{B_j} \leq \tilde{I}(B_j)$ and $F_{C_j} \leq {\sim}\tilde{U}(C_j)$ for all B_j and C_j such that

$$F \leq (F_{B_1} * \ldots * F_{B_m} * F_{C_1} * \ldots * F_{C_n}) \cdot r_i$$

If F_{C_j} contains a negative label for some C_j, then $\lambda^c(F_{C_j}) = 0$ and consequently it follows that $\lambda^c(F) = 0 \leq T_{P^V}(J)(p)$. Thus, we assume that F_{C_j} does not contain negated labels, i.e. it only contains double negated labels, for all C_j. Therefore $\lambda^c(F_{C_j} = 1$ and there is not positive justification of C_j w.r.t. $\tilde{U}$ for all atom C_j and so $V(C_j) = 0$ for all C_j. Thus there is a rule in P^V in the form of

$$r_i : p \leftarrow B_1, \ldots, B_m$$

Note now that, by hypothesis, $\lambda^c(F_{B_j}) \leq T_{P^V}(B_j)$ for all B_j. Hence $\lambda^c(F) \leq T_{P^V}(p)$ for all atom p and so $T_{P^V}(J) \geq \lambda^c(\tilde{T}_{P^{\tilde{U}}}(I))$. Furthermore, by Lemma 27, it follows that $T_{P^V}(J) \leq \tilde{T}_{P^{\tilde{U}}}(I)$. Hence $T_{P^V}(J) = \lambda^c(\tilde{T}_{P^{\tilde{U}}}(I))$. □

Lemma 31. *Let P be a labelled logic program, $\tilde{U}$ and V respectively be a CP and a causal interpretation such that $V = \lambda^c(\tilde{U})$ for all atom p. Let also $\tilde{I}$ and J respectively be the least model of programs $P^{\tilde{U}}$ and P^V. Then $J = \lambda^c(\tilde{I})$.* □

Proof. Since $\tilde{I}$ and J are the least models respectively of programs $P^{\tilde{U}}$ and P^V it follows that $\tilde{I} = \tilde{T}^{\omega}_{P^{\tilde{U}}}$ and $J = T^{\omega}_{P^V}$. We proceed by induction on the number of iterations k. When $k = 0$, it follows that $T^0_{P^V}(p) = 0 \geq \lambda^c(0) = \lambda^c(\tilde{T}^0_{P^{\tilde{U}}}(p))$. When $0 < k < \omega$, we assume as induction hypothesis the statement holds for the case $k - 1$. Then, by Lemma 30, it follows for the case k.

Finally when $k = \omega$, for all $F \leq \tilde{T}^{\omega}_{P^{\tilde{U}}}(p))$ there is some i s.t. $F \leq \tilde{T}^i_{P^{\tilde{U}}}(p))$ and then, by induction hypothesis, $\lambda^c(F) \leq T^i_{P^V}(p)$ and, since furthermore it holds that $T^i_{P^V}(p)) \leq T^{\omega}_{P^V}(p)$, then $\lambda^c(F) \leq T^{\omega}_{P^V}(p)$. That is $J \geq \lambda^c(\tilde{I})$. Furthermore, by Lemma 27, it follows that $J \leq \tilde{I}$. Hence $J = \lambda^c(\tilde{I})$. □

Proof of Proposition 8.1

Lemma 32. *Let P be a labelled logic program, $\tilde{I}$ and J respectively be a CP and a causal interpretation such that $V = \lambda^c(\tilde{U})$. Then $\Gamma_P(J) \geq \lambda^c(\tilde{\Gamma}_P(\tilde{I}))$.* □

Proof. By Lemma 31 it follows that $J' = \tilde{I}'$ with $\tilde{I}'$ and J' respectively the least models of $P^{\tilde{I}}$ and $P^{\tilde{J}}$ and by definition $\Gamma(J) = J'$ and $\tilde{\Gamma}(\tilde{I}) = \tilde{I}'$. Hence $\Gamma(J) \geq \lambda^c(\tilde{\Gamma}(\tilde{I}))$. □

Proof of Proposition 8.1. Let $\mathfrak{P}'$ the result of replacing each label of the form *not* A by 1 in $\mathfrak{P}(P)$. Then, it follows that $\mathfrak{P}' = P \cup Q$ where every fact in Q is of the form $(0 : A)$. It is easy to see that every CP-interpretation $\mathfrak{I}$ is a model of $(0 : A)$ and, so that, the least models of $\mathfrak{P}(P^{\mathfrak{I}})$ and $P^{\mathfrak{I}}$ are the same for every CP-interpretation $\mathfrak{I}$. That is, $\mathfrak{G}^2_{\mathfrak{P}}(\mathfrak{I}) = \mathfrak{G}^2_P(\mathfrak{I})$ for every CP-interpretation $\mathfrak{I}$ and, consequently, the CP well-founded models of $\mathfrak{P}'$ and P are the same. □

Proof of Theorem 8.4

Lemma 33. *Let P be a labelled logic program, $\tilde{U}$ and $\mathfrak{U}$ respectively be a CP and a provenance interpretation such that $\mathfrak{U} \leq \lambda^p(\tilde{U})$. Let also $\tilde{I}$ and $\mathfrak{I}$ respectively be a CP and provenance interpretation such that $\lambda^p(I) \leq \mathfrak{I}$. Then, $\lambda^p(\tilde{T}_{P\tilde{U}}(I)) \leq \mathfrak{T}_{P\mathfrak{U}}(\mathfrak{I})$.* □

Proof. Suppose not and let the atom a and the CP justification E be the witnesses. Since $E \leq \tilde{T}_{P\tilde{U}}(I)(a)$, there is a rule $r_i \in P$ of the form (92) and

$$E \;=\; (E_{B_1} * \ldots * E_{B_1} * E_{C_1} * \ldots * E_{C_1}) \cdot r_i$$

where $E_{B_j} \leq \tilde{I}(B_j)$ and $E_{C_j} \leq {\sim}\tilde{U}(C_j)$. Note that, by hypothesis $\lambda^p(\tilde{I}) \leq \mathfrak{I}$ and $\mathfrak{U} \leq \lambda^p(\tilde{U})$. Since $\lambda^p(\tilde{I}) \leq \mathfrak{I}$ and $E_{B_j} \leq \tilde{I}(B_j)$, it follows that, $\lambda^p(E_{B_j}) \leq \mathfrak{I}(B_j)$. Furthermore, since $\mathfrak{U} \leq \lambda^p(\tilde{U})$, by Lemma 26, it follows that $\lambda^p({\sim}\tilde{U}) \leq \neg\mathfrak{U}$ and then, since $E_{C_j} \leq {\sim}\tilde{U}(C_j)$, it follows that $\lambda^p(E_{C_i}) \leq \neg\mathfrak{U}(C_j)$ for all C_j. Therefore $\lambda^p(E) \leq \mathfrak{T}_{P\mathfrak{U}}(\mathfrak{I})(a)$ which is a contradiction with the assumption. □

Lemma 34. *Let P be a labelled program, $\tilde{U}$ and $\mathfrak{U}$ respectively be a CP and a provenance interpretation such that $\lambda^p(\tilde{U}) \leq \mathfrak{U}$. Let also $\tilde{I}$ and $\mathfrak{I}$ respectively be a CP and provenance interpretation such that $\mathfrak{I} \leq \lambda^p(\tilde{I})$. Then, $\mathfrak{T}_{P\mathfrak{U}}(\mathfrak{I}) \leq \lambda^p(\tilde{T}_{P\tilde{U}}(\tilde{I}))$.* □

Proof. Suppose not and let the atom a and the provenance justification D be the witness. Since $D \leq \mathfrak{T}_{P^{\mathfrak{U}}}(\tilde{I})(a)$, there is some rule $r_i \in P^{\mathfrak{U}}$ of the form (92) and

$$D \;=\; D_{b_1} * \ldots * D_{b_m} * D_{c_1} * \ldots * D_{c_n} * r_i$$

where $D_{b_j} \leq \mathfrak{I}(b_j)$ for each b_j and $D_{c_j} \leq \neg\mathfrak{U}(c_j)$ for each c_j. By hypothesis, $\lambda^p(\tilde{U}) \leq \mathfrak{U}$ and $\mathfrak{I} \leq \lambda^p(\tilde{I})$. Since $D_{b_j} \leq \mathfrak{I}(b_j)$ and $\mathfrak{I} \leq \lambda^p(\tilde{I})$, it follows that $D_{b_j} \leq \lambda^p(\tilde{I})(b_j)$. Furthermore, since $\lambda^p(\tilde{U}) \leq \mathfrak{U}$, by Lemma 26, it follows that $\neg\mathfrak{U} \leq \lambda^p(\sim\tilde{U})$ and then, since $D_{c_j} \leq \neg\mathfrak{U}(c_j)$, it follows that $D_{c_j} \leq \lambda^p(\sim\tilde{U})(c_j)$. Therefore $E \overset{\text{def}}{=} (E_{B_1} * \ldots * E_{B_1} * E_{C_1} * \ldots * E_{C_1}) \cdot r_i \leq \tilde{T}_{P^{\tilde{U}}}(I)(a)$ such that $D_{B_j} \leq \lambda^p(E_{B_j})$ and $D_{C_j} \leq \lambda^p(E_{C_j})$. Then $D \leq \lambda^p(E)$ which contradicts the assumption. □

Lemma 35. *Let P be a labelled program, $\tilde{U}$ and $\mathfrak{U}$ respectively be a CP and a provenance interpretation such that $\lambda^p(\tilde{U}) = \mathfrak{U}$. Let also $\tilde{I}$ be CP interpretation. Then it holds that $\mathfrak{T}_{P^{\mathfrak{U}}}(\lambda^p(\tilde{I})) = \lambda^p(\tilde{T}_{P^{\tilde{U}}}(\tilde{I}))$.* □

Proof. The proof directly follows by combining the two directions from Lemmas 33 and 34. □

Lemma 36. *Let P be a labelled program, $\tilde{U}$ and $\mathfrak{U}$ respectively be a CP and a provenance interpretation such that $\lambda^p(\tilde{U}) = \mathfrak{U}$. Let also $\tilde{I}$ be the least model of the program $P^{\tilde{U}}$. Then $\lambda^p(\tilde{I})$ is the least model of $P^{\mathfrak{U}}$.* □

Proof. Since $\tilde{I}$ and $\mathfrak{I}$ respectively are the least fixpoints of $\tilde{T}_{P^{\tilde{U}}}$ and $\mathfrak{T}_{P^{\mathfrak{U}}}$, it follows, by Theorems 8.1 and 8.3, that $\tilde{I} = \tilde{T}^{\omega}_{P^{\tilde{U}}}$ and $\mathfrak{I} = \mathfrak{T}^{\omega}_{P^{\mathfrak{U}}}$. We proceed by induction on the number of iterations k. When $k = 0$, it follows that $\lambda^p(\tilde{T}^{0}_{P^{\tilde{U}}})(p) = \lambda^p(0)$ and $\mathfrak{T}^{0}_{P^{\mathfrak{U}}}(p) = 0$ and, since $\lambda^p(0) = \bot$, then it $\lambda^p(\tilde{T}^{0}_{P^{\tilde{U}}})(p) = \mathfrak{T}^{0}_{P^{\mathfrak{U}}}(p)$.

When $0 < k < \omega$, we assume as induction hypothesis that the statement holds for the case $k - 1$. Then, by Lemma 35, it follows for the case k. Finally when $k = \omega$, for all justification $E \leq \tilde{T}^{\omega}_{P^{\tilde{U}}}(p))$ (resp. for all $D \leq \mathfrak{T}^{\omega}_{P^{\mathfrak{U}}}(p))$) there is some $i < \omega$ s.t. $E \leq \tilde{T}^{i}_{P^{\tilde{U}}}(p))$ (resp. $D \leq \mathfrak{T}^{i}_{P^{\mathfrak{U}}}(p))$). By induction hypothesis, $\lambda^p(E) \leq \mathfrak{T}^{i}_{P^{\mathfrak{U}}}(p)$ (resp. $D \leq \lambda^p(\tilde{T}^{i}_{P^{\tilde{U}}}(p)))$). Therefore it holds that $\lambda^p(E) \leq \mathfrak{T}^{\omega}_{P^{\mathfrak{U}}}(p)$ (respectively $D \leq \lambda^p(\tilde{T}^{\omega}_{P^{\tilde{U}}}(p))$). That is $\lambda^p(\tilde{T}^{\omega}_{P^{\tilde{U}}}(p)) = \mathfrak{T}^{\omega}_{P^{\mathfrak{U}}}(p)$. □

Proof of Theorem 8.4. Let $\tilde{U}$ and $\mathfrak{U}$ respectively be a CP and a provenance interpretation such that $\lambda^p(\tilde{U}) = \mathfrak{U}$. Since P is positive, it follows that $P = P^{\tilde{U}} = P^{\mathfrak{U}}$.

That is, $\tilde{I}$ and $\mathfrak{I}$ are respectively the least CP-model and least provenance model of $P^{\tilde{U}}$ and $P^{\mathfrak{U}}$ and, consequently, by Lemma 36 it follows that $\mathfrak{I} = \lambda^p(\tilde{I})$. $\square$

Proof of Theorem 8.5

Lemma 37. *Let P be a labelled logic program and $\tilde{I}$ be a CP interpretation. Then $\mathfrak{G}_P(\lambda^p(\tilde{I})) = \lambda^p(\tilde{\Gamma}_{\mathfrak{P}}(\tilde{I}))$.* $\square$

Proof. By definition $\tilde{\Gamma}_P(\tilde{I})$ is the least models of $P^{\tilde{I}}$. Then, by Lemma 36, it follows that $\lambda^p(\tilde{\Gamma}_P(\tilde{I}))$ is the least model of $\mathfrak{P}^{\lambda^p(\tilde{I})}$ therefore $\mathfrak{G}_P(\lambda^p(\tilde{I})) = \lambda^p(\tilde{\Gamma}_P(\tilde{I}))$. $\square$

Lemma 38. *Let P be a labelled logic program and $\tilde{I}$ and $\tilde{J}$ be two fixpoints of the operator $\tilde{\Gamma}^2_P$ such that $\lambda^p(\tilde{J}) = \lambda^p(\tilde{I})$. Then $\tilde{I} = \tilde{J}$.* $\square$

Proof. Since $\lambda^p(\tilde{J}) = \lambda^p(\tilde{I})$, then $\neg\lambda^p(\tilde{J}) = \neg\lambda^p(\tilde{I})$ and by Lemma 26, $\lambda^p(\sim\tilde{J}) = \lambda^p(\sim\tilde{I})$, furthermore, this implies $\sim\tilde{J} = \sim\tilde{I}$. Hence $P^{\tilde{I}} = P^{\tilde{J}}$ and $\tilde{\Gamma}_P(\tilde{I}) = \tilde{\Gamma}_P(\tilde{J})$. Then it is clear that $\tilde{\Gamma}^2_P(\tilde{I}) = \tilde{\Gamma}^2_P(\tilde{J})$, i.e. $\tilde{I} = \tilde{J}$. $\square$

Lemma 39. *Let P be a labelled logic program and $\mathfrak{I}$ be a provenance interpretation. Then $\mathfrak{I}$ is a fixpoint of the operator $\mathfrak{G}^2_P$ if and only if there is a CP fixpoint $\tilde{I}$ of $\tilde{\Gamma}^2_P$ such that $\mathfrak{I} = \lambda^p(\tilde{I})$. Furthermore, given $\mathfrak{I}$, then $\tilde{I}$ is unique.* $\square$

Proof. By Lemma 37, for any interpretation $\tilde{J}$ such that $\tilde{J} = \lambda^p(\mathfrak{I})$ it holds that $\mathfrak{G}_P(\lambda^p(\tilde{J})) = \lambda^p(\tilde{\Gamma}_P(\tilde{J}))$. Then

$$\tilde{\Gamma}^2_P(\mathfrak{I}) = \lambda^p(\tilde{\Gamma}_P(\lambda^p(\tilde{\Gamma}_P(\tilde{J}))))$$

Since, for all $\tilde{J}$, it holds that $\tilde{\Gamma}_P(\lambda^p(\tilde{J})) = \tilde{\Gamma}_P(\tilde{J})$, then

$$\tilde{\Gamma}^2_P(\mathfrak{I}) = \lambda^p(\tilde{\Gamma}^2_P(\tilde{J}))$$

Hence, if $\mathfrak{I}$ is a fixpoint of $\tilde{\Gamma}^2_P$, i.e. $\tilde{\Gamma}^2_P(\mathfrak{I}) = \mathfrak{I}$

$$\mathfrak{I} = \lambda^p(\tilde{\Gamma}^2_P(\tilde{J}))$$

Therefore, for $\tilde{I} \overset{\text{def}}{=} \tilde{\Gamma}^{2\infty}_P(\tilde{J})$, it holds that

$$\mathfrak{I} = \lambda^p(\tilde{\Gamma}^{2\infty}_P(\tilde{J}))$$

Furthermore, let $\tilde{J}$ be fixpoint such that $\tilde{I} \neq \tilde{J}$ and $\lambda^p(\tilde{J}) = \mathfrak{I}$. By Lemma 38, it follows that $\tilde{I} = \tilde{J}$ which is a contradiction. So that $\tilde{I}$ is unique. The other way around. Since $\tilde{I} = \tilde{\Gamma}^2_P(\tilde{I})$, by Lemma 37, $\lambda^p(\tilde{I}) = \lambda^p(\tilde{\Gamma}^2_P(\tilde{I})) = \mathfrak{G}^2_P(\lambda^p(\tilde{I}))$. □

Proof of Theorem 8.5. By Lemma 39, there is a fixpoint $\tilde{I}$ of $\tilde{\Gamma}^2_P$ such that $\texttt{lfp}(\mathfrak{G}^2_P) = \lambda^p(\tilde{I})$. Then $\texttt{lfp}(\mathfrak{G}^2_P) \geq \lambda^p(\texttt{lfp}(\tilde{\Gamma}^2_P))$. By Lemma 39, it also follows that $\lambda^p(\texttt{lfp}(\tilde{\Gamma}^2_P))$ is a fixpoint of $\mathfrak{G}^2_P$. Thus, $\texttt{lfp}(\mathfrak{G}^2_P) \leq \lambda^p(\texttt{lfp}(\tilde{\Gamma}^2_{\mathfrak{P}}))$ and, consequently, $\texttt{lfp}(\mathfrak{G}^2_P) = \lambda^p(\texttt{lfp}(\tilde{\Gamma}^2_{\mathfrak{P}}))$. By following the same reasoning we conclude that $\texttt{gfp}(\mathfrak{G}^2_P) = \lambda^p(\texttt{gfp}(\tilde{\Gamma}^2_P))$. Note that by definition $\tilde{W}$ and $\mathfrak{W}$ are respectively equal to $\langle \texttt{lfp}(\tilde{\Gamma}^2_P), \texttt{gfp}(\tilde{\Gamma}^2_P) \rangle$ and $\langle \texttt{lfp}(\mathfrak{G}^2_P), \texttt{gfp}(\mathfrak{G}^2_P) \rangle$. Hence $\mathfrak{W} = \lambda^p(\tilde{W})$. □

Proof of Proposition 8.2

Proof of Proposition 8.2. Let $\mathfrak{I}$ be the CP well-founded model of P. From Lemma 8.1, it follows that $\tilde{I}$ is each occurrence of labels of the form *not* A by 1. Furthermore, from Theorem 8.2 it follows that $W = \lambda^c(\tilde{I})$. □

Proof of Theorem 8.6

Proof of Theorem 8.6. From the only if direction. If $D \leq \mathit{Why}_P(A)$, then, by Definition 8.15, it follows that $D \leq \texttt{lfp}(\mathfrak{G}^2_P)(A)$. Furthermore, by Theorem 8.5, it follows that $\texttt{lfp}(\mathfrak{G}^2_P)(A) = \lambda^p(\texttt{lfp}(\tilde{\Gamma}^2_P))(A)$. Hence, there is some CP-term with no sums E such that $D \leq \lambda^p(E)$ and $E \leq \texttt{lfp}(\tilde{\Gamma}^2_P))(A)$. Note that, λ^p only maps to 1 such labels corresponding to $\delta(B)$ for some literal B. Therefore D contains all the labels in E but for those corresponding to $\delta(B)$ for some literal B. Furthermore, if D does not contain negations, then E only may contains double negations, but not single negations. Let $G = \lambda^c(E)$. The G is a causal term with no sums and, since E does not contains single negations, it also holds that $G \neq 0$. Hence G is a causal graph. Furthermore, G does not contain any label not in E and, so that, D contains all the labels in G but for those corresponding to $\delta(B)$ for some literal B. Moreover, from Lemma 8.2, it follows that $\texttt{lfp}(\Gamma^2_P) = \lambda^c(\tilde{\Gamma}^2_P)$ and, therefore, $G = \lambda^c(E) \leq \lambda^c(\tilde{\Gamma}^2_P) = \texttt{lfp}(\Gamma^2_P)$.

For the if direction. Let $G \leq \texttt{lfp}(\Gamma^2_P)(A)$. Then, there is some CP-term with no sums E such that $G \leq \lambda^c(E)$ and $E \leq \texttt{lfp}(\tilde{\Gamma}^2_P)(A)$ and E contains all the labels in G. Let $D = \lambda^p(E)$. Then D is a conjunction containing all the labels in E — and so in G — but for those corresponding to $\delta(B)$ for some literal B and $D \leq \mathtt{lfp}(\mathfrak{G}^2_P)(A) = Why_P(A)$. □

Proof of Proposition A.7

Proposition A.7. *Let P be a labelled logic program with no two-valued (standard) well founded model (i.e. there is some undefined atom). Then there is not any real answer set why-not provenance justification for any atom.* □

Proof of Proposition A.7. Since p is undefined in the (standard) well-founded model of P, by Theorem 3 in Damásio et al. [2013], it follows that there is not positive justification of neither p nor *not* p.

Let $Why_P(A) = D_1 \vee \ldots \vee D_n$ and $Why_P(not\ A) = E_1 \vee \ldots \vee E_m$ such that each D_i and E_i are prime. Then for all D_i and E_j there are negative literals $\neg L_{C_i}$ and $\neg L_{E_i}$ such that $D_i \models \neg L_{C_i}$ and $E_i \models \neg L_{E_i}$.

Let $F' \overset{\text{def}}{=} L_{C_1} \wedge \ldots \wedge L_{C_n} \wedge L_{E_1} \wedge \ldots \wedge L_{E_m}$. Then $F' \models \neg D_i$ and $F \models \neg E_i$ for all D_i and E_i. Hence

$$\begin{array}{rcl} F' \models \neg Why_P(A) & = & \neg D_1 \wedge \ldots \wedge \neg D_n \\ F' \models \neg Why_P(A) & = & \neg E_1 \wedge \ldots \wedge \neg E_n \end{array}$$

Consequently $F' \models Why_P(\text{undef } A) \equiv \neg Why_P(A) \wedge Why_P(not\ A)$. Then let F be a prime implicant of $Why_P(\text{undef } A)$ such that $F' \models F$. By construction F is a positive conjunction such that $F \models Why_P(\text{undef } A)$. Note that F' is positive, and hence F must be too. Consequently $\neg Why_P(\text{undef } A) \models \neg F$. Note that, since F is a positive conjugation, $\neg F$ is a disjunction of negative literals, and consequently every answer set why-not provenance must contain at least one negative literal. □

CHAPTER 9: IMPLEMENTATION

Proof of Proposition 9.1

Proof of Proposition 9.1. If I is a causal stable model of P, then I is the least model of P^I and so $I = T^{\omega}_{P^I}$. Let $t = I(A)$ be a causal term and J be the least model of the program I. Then $t = u \cdot \delta(A)$ for any atom A. Then $J(A) = t \cdot \delta(A) = u \cdot \delta(A) \cdot \delta(A)$. Since $\delta(A) \in Lb \cup 1$, it follows that $\delta(A) \cdot \delta(A) = \delta(A)$. Hence $J(A) = u \cdot \delta(A) = t = I(A)$ for any atom A. That is, $J = I$. □

Proof of Theorem 9.1

Lemma 40. *Let P be a positive program, R be a rule and $Q = P \cup \{R\}$. Then $T^k_P \leq T^k_Q$ for all k.* □

Proof. In case that $k = 0$, $T^k_P = T^k_Q = 0$. Otherwise the proof follows by induction assuming that the statement holds for the case $k-1$. $T^k_Q(A)$ is given by

$$\sum \{ \ (\ T^{k-1}_Q(B_1) * \ldots * T^{k-1}_Q(B_m)\ ) \cdot l_R \cdot \delta(A) \mid R' \in Q \text{ and } head(R') = A \ \}$$

If $head(R) \neq A$, then just note that the rules with head A in P and Q are the same, so that, since sums, products and applications are monotonic, it follows that $T^{k-1}_P \leq T^{k-1}_Q$ implies $T^k_P \leq T^k_Q$. Otherwise, Q has a an extra rule, so that $T^k_P \leq T^k_Q$ also holds. The case ω is just a sum, so it follows from monotonicity. □

Lemma 41. *Let P_1 and P_2 be two positive programs such that no atom occurring in P_1 is a head atom of P_2. Then $T^{\omega}_{P_1 \cup P_2}(A) = T^{\omega}_{P_1}(A)$ for all atom A not in the head of any rule in P_2.* □

Proof. By monotonicity, it is clear that $T^{\omega}_{P_1 \cup P_2}(A) \geq T^{\omega}_{P_1}(A)$ or all literal A. In case that $k = 0$, it holds that $T^k_{I_1 \cup P_2} = T^k_{P_1 \cup P_2} = \mathbf{0}$. Otherwise, we assume as induction hypothesis that it holds for the case $k-1$. Suppose $T^k_{P_1 \cup P_2} \not\leq T^k_{P_1}$. Then there is a literal A and a rule R in $P_1 \cup P_2$ such that

$$t \ \overset{\text{def}}{=} \ (\ T^{k-1}_{P_1 \cup P_2}(B_1) * \ldots * T^{k-1}_{P_1 \cup P_2}(B_m)\ ) \cdot l_R \cdot \delta(A) \ \not\leq \ T^k_{P_1}(A)$$

Note that A is not in the head of any rule in P_2, so R must be in P_1, and by hypothesis, it follows that no literal in the body of R is in P_2. Hence, by induction hypothesis, $T^{k-1}_{P_1 \cup P_2}(B_i) \leq T^{k-1}_{P_1}(B_i)$ for all B_i. Hence, since R in P_1, it follows that, $t \leq T^k_{P_1}(A)$. Finally note that, in case that $k = \omega$, this is just the sum for $i \leq k$ and the sum operator is monotonic. □

Lemma 42. *Let P_1 and P_2 be two positive programs such that no literal occurring in P_1 is a head atom of P_2. Let I_1 be the least model of P_1. Then $T^{\omega}_{I_1 \cup P_2} = T^{\omega}_{P_1 \cup P_2}$, that is the least model of $I_1 \cup P_2$ is the same than of $P_1 \cup P_2$.* □

Proof. From Lemma 40, $T^{\omega}_{P_1} \leq T^{\omega}_{P_1 \cup P_2}$. Furthermore, $T^{\omega}_{I_1} = T^{\omega}_{P_1}$. So that $T^{\omega}_{I_1} \leq T^{\omega}_{P_1 \cup P_2}$. Again from Lemma 40, $T^{\omega}_{I_1 \cup P_2} \leq T^{\omega}_{P_1 \cup P_2 \cup P_2}$, that is $T^{\omega}_{I_1 \cup P_2} \leq T^{\omega}_{P_1 \cup P_2}$.

Furthermore, in case that $k = 0$, it holds that $T^k_{I_1 \cup P_2} = T^k_{P_1 \cup P_2} = \mathbf{0}$. Otherwise, we assume as induction hypothesis that it holds for the case $k-1$. Suppose $T^k_{I_1 \cup P_2} \not\geq T^k_{P_1 \cup P_2}$, then there is a literal A and a rule R in $P_1 \cup P_2$ such that

$$G \;=\; \big(\, T^{k-1}_{P_1 \cup P_2}(B_1) * \ldots * T^{k-1}_{P_1 \cup P_2}(B_m) \,\big) \cdot l_R \cdot \delta(A) \quad \not\leq \quad T^k_{I_1 \cup P_2}(A)$$

If R is in P_2, then it is also in $I_1 \cup P_2$, and since by induction hypothesis $T^{k-1}_{I_1 \cup P_2}(B_1) = T^{k-1}_{P_1 \cup P_2}(B_1)$ and all three operations are monotonic, the above inequality is a contradiction. Hence R must be in P_1, and by hypothesis no literal in R is a head in P_2, so that$T^k_{P_1 \cup P_2}(B_i) \leq I_1(B_i) \leq T^k_{I_1 \cup P_2}(B_i)$ for all B_i. Therefore, by induction hypothesis and operations monotonicity, the above is also a contradiction.

Otherwise, R is in P_1, and then no literal in the body of R is in the head on any rule in P_2. Hence, from Lemma 41, it follows $T^{k-1}_{P_1 \cup P_2}(B_i) \leq T^{k-1}_{P_1}(B_i) \leq I_1(B_i)$ for all B_i.

Then $G \leq T^k_{P_1}(A) \leq T_{P_1}(I_1)(A) = I_1(A) \leq T^k_{I_1 \cup P_2}(A)$. Hence $T^k_{I_1 \cup P_2} \geq T^k_{P_1 \cup P_2}$. In case that $k = \omega$, just note that is the sum for the cases $i \leq \omega$ and sums are monotonic. Consequently $T^k_{I_1 \cup P_2} = T^k_{P_1 \cup P_2}$. □

Proof of Theorem 9.1. Let I be a causal stable model of $J \cup P_2$ and $J = I_{|S}$ be a causal stable model of P_1. Then I is the least model of $(J \cup P_2)^I = J \cup P_2^I$ and J is the least model of P_1^J. From Lemma 42, the least models of programs $P_1^J \cup P_2^I$ and $J \cup P_2^I$ are the same. Hence I is the least model of $P_1^J \cup P_2^I$. Furthermore, $J(A) = I_{|S}(A) = I(A)$ for all $A \in S$, (that is for all literal A occurring in P_1). Hence $P_1^J = P_1^I$ and, then, I is the least model of $P_1^I \cup P_2^I = (P_1 \cup P_2)^I$ and, consequently, I is a causal stable model of $P_1 \cup P_2$.

The other way around. I is the least model of $(P_1 \cup P_2)^I = P_1^I \cup P_2^I$. Let J be the least model of P_1^I. Note that I satisfies all rules in P_1^I and, since for all literal $A \notin S$ there is no rule in P_1 with A in the head, it follows that $I_{|S}$ also satisfies all rules in P_1^I. That is $J \leq I_{|S}$. Moreover, from Lemma 41, it follows that $I(A) = J(A)$ for all atom A not in the head of any rule in P_2. Hence, if it were the case that $J(A) < I_{|S}(A)$ for some A, atom A would be in the head of some rule in P_2, so $A \notin S$ and, consequently, $I_{|S}(A) = 0$ which is a contradiction with the assumption that $J(A) < I_{|S}(A)$. Hence $J = I_{|S}$ is the least model of P_1^I. Note also that, since $J(A) = I_{|S}(A) = I(A)$ for every atom A occurring in P_1, then $P_1^I = P_1^J$. That is, $J = I_{|S}$ is a causal stable model of P_1. Furthermore, note that I is the least model of $(P_1 \cup P_2)^I = P_1^I \cup P_2^I = P_1^J \cup P_2^I$. From Lemma 41, the least models of programs $P_1^J \cup P_2^I$ and $J \cup P_2^I$ are the same, that is I is the least model of $J \cup P_2^I = I_{|S} \cup P_2^I$ and, consequently, a causal stable model of $I_{|S} \cup P_2$. □

Proof of Proposition 9.3

Proof of Proposition 9.3. Assume that I_2 is a causal stable model of P_2 and let $P' = \{aux \leftarrow B_i, not\ \overline{aux}\} \cup \{\overline{aux} \leftarrow not\ aux\}$ be a set of rules. Note that, $I_2 \cup P'$ has two causal stable models, $I_2 \cup \{aux \mapsto 1\}$ and $I_2 \cup \{\overline{aux} \mapsto 1\}$, if $I(B_i) \neq 0$, and it has a unique causal stable model containing $I_2 \cup \{aux \mapsto 1\}$ if $I(B_i) = 0$. Since $I_2(\psi_i :: B_i) \neq 0$ implies $I_2(B_i) \neq 0$, it holds that, in both cases, I_3 is one of the causal stable models of $I_2 \cup P'$. Furthermore, from Theorem 9.1, since I_2 is a causal stable model of P_2 and aux and $\overline{aux}$ do not occur in P_2, it follows that I_3 is a causal stable model of $P_2 \cup P'$.

Note that, P_3 and $P_2 \cup P'$ are equal but for the rule with label l_R. In case that $I_3(C_j) \neq 0$ for some C_j, then $(P_3)^{I_3} = (P_2 \cup P')^{I_3}$. Furthermore by definition, since I_3 is a causal stable model of $P_2 \cup P'$, it is the least model of $(P_3)^{I_3} = (P_2 \cup P')^{I_3}$, and consequently I_3 is a causal stable model of P_3.

Similarly, if $I(B_j) \neq 0$ for for some B_j with $j \neq i$, the least model of $(P_3)^{I_3}$ and $(P_2 \cup P')^{I_3}$ coincide, and therefore I_3 is a causal stable model of P_3.

Hence, in the following, we assume that $I_3(B_j) \neq 0$ for all B_j with $j \neq i$ and $I_3(C_j) = 0$ for all C_j.

Note that $I_3(B_i) = 0$ iff $I_2(B_i) = 0$, and then $I_3(B_i) = 0$ implies $I_2(\psi_i :: B_i) = 0$ and $I_3(aux) = 0$. Consequently the least model of $(P_3)^{I_3}$ and $(P_2 \cup P')^{I_3}$ coincide and I_3 is a causal stable model of P_3.

Otherwise, $I_3(B_i) \neq 0$ and either $I_2(\psi_i :: B_i) \neq 0$ or $I_2(\psi_i :: B_i) = 0$. If the last, $I_2(\psi_i :: B_i) = 0$, then, by definition, $I_3(aux) = 0$, and consequently the least model of $(P_3)^{I_3}$ and $(P_2 \cup P')^{I_3}$ coincide and I_3 is a causal stable model of program P_3.

Finally if $I_2(\psi_i :: B_i) \neq 0$, then, by definition $I_3(aux) = 1$ and the least model I'_3 of $(P_3)^{I_3}$ holds $(I'_3)^{cl} = (I_3)^{cl}$. Consequently $(P_3)^{I_3} = (P_3)^{I'_3}$ and I'_3 is a causal stable model of P_3. In this case just rename I'_3 as I_3 for the name used in the proposition statement.

That is, if I_2 is a causal stable model of P_2, there is a causal stable model I_3 of P_3 holding the proposition statement. $\square$

B IMPLEMENTATION OF EXAMPLES

This appendix revisits most of the examples used along this dissertation, showing for each of them the corresponding implementation in the input language of the program *cgraphs*.

We start with Program 3.1 representing Example 1.4.

```
b    :: bomb  :- open.
o    :: open  :- up(a), up(b).
k    :: open  :- key.
u(L) :: up(L) :- lift(L).

:: lift(a).
:: lift(b).
:: key.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
bomb key lift(a) lift(b) open up(a) up(b)

bomb    = (lift(b).u(b)*lift(a).u(a)).o.b + key.k.b
key     = key
lift(a) = lift(a)
lift(b) = lift(b)
open    = key.k + (lift(b).u(b)*lift(a).u(a)).o
up(a)   = lift(a).u(a)
up(b)   = lift(b).u(b)
```

The causal term assigned to *bomb* is equivalent, after apply distributive to the following causal term:

$$lift(a) \cdot u(a) \cdot o \cdot b \; * \; lift(b) \cdot u(b) \cdot o \cdot b \; + \; key \cdot k \cdot b \tag{49}$$

presented when this example was discussed. In the implementation, we have preferred not fully expand tg-terms in products of causal chains in the sake of compactness.

Next, Program 3.2 representing Example 3.1.

```
a :: alarm      :- sw3, current(d).
b :: current(b) :- sw1.
b :: current(b) :- sw4, current(c).
c :: current(c) :- sw2.
c :: current(c) :- sw4, current(b).
d :: current(d) :- current(b).
d :: current(d) :- current(c).

:: sw1.
:: sw2.
:: sw3.
:: sw4.
```

Similarly, the following listing shows the unique corresponding stable model:

```
Answer 1:
alarm current(b) current(c) current(d) sw1 sw2 sw3 sw4

alarm      = (sw3*sw1.b.d).a + (sw3*sw2.c.d).a
current(b) = sw1.b + (sw2.c*sw4).b
current(c) = (sw4*sw1.b).c + sw2.c
current(d) = sw1.b.d + sw2.c.d
sw1        = sw1
sw2        = sw2
sw3        = sw3
sw4        = sw4
```

Program 3.3 representing Example 3.2.

```
lock(a).
lock(b).

b    :: bomb  :- open.
o    :: open  :- up(a), up(b).
l(L) :: up(L) :- wireless, lock(L).

s :: wireless.
y :: wireless.
```

The following listing shows the corresponding unique stable model:

```
Answer 1:
bomb lock(a) lock(b) open up(a) up(b) wireless

bomb     = (y.l(a)*y.l(b)).o.b + (s.l(b)*s.l(a)).o.b
open     = (s.l(b)*s.l(a)).o + (y.l(b)*y.l(a)).o
up(a)    = s.l(a) + y.l(a)
up(b)      = s.l(b) + y.l(b)
wireless = y + s
```

The evaluation of *bomb* corresponds to the causal graphs G_1 and G_2 depicted in Figure 9. The program *cgraphs* computes the causes corresponding to a program with a unique atom labelling, from which removes the labels introduced by that mapping. The evaluation of *bomb* corresponds to causal graphs the sum of G_1' and G_2' depicted in Figure 12. After removing the vertices in the image of δ and contract the adjacent vertices from G_1' and G_2' we obtain causal graphs G_1 and G_2 depicted in Figure 9.

Program 4.1 representing Example 2.1.

```
t1  ::  turn(1)  :-  turn(2),  coupled.
t2  ::  turn(2)  :-  turn(1),  coupled.

::  coupled.
::  turn(1).
```

In Program 4.1 we use the labels *c* and *s* instead of *coupled* and *spinning*(1) due to space reasons. Here, we label facts by an homograph label as usual. The following listing shows the unique stable model of this program:

```
Answer  1:
coupled  turn(1)  turn(2)

coupled  =  coupled
turn(1)  =  turn(1)
turn(2)  =  (coupled*turn(1)).t2
```

Program 4.3 representing Example 1.7.

```
1  f  ::  fire    :-  match,  oxygen.
2         oxygen  :-  not  noxygen.
3
4  :-  oxygen,  noxygen.
5
6  ::  match.
7  ::  oxygen.
```

In line 2, we use *noxygen* for representing the strong negation of *oxygen*, that is $\overline{oxygen}$. The restriction in line 4 states that *oxygen* cannot be true and false at the same time. The following listing shows the corresponding unique stable model:

```
Answer 1:
fire match oxygen

fire  = match.f
match = match
```

On page 98, where we have discussed this program, we have previously analysed the following variation, where the rule in line 2 has been removed.

```
1  f :: fire   :- match, oxygen.
2
3
4  :- oxygen, noxygen.
5
6  :: match.
7  :: oxygen.
```

The following listing shows the corresponding unique stable model, where we easily can see that *oxygen* and *match* are symmetrically treated as part of the cause of *fire*.

```
Answer 1:
fire match oxygen

fire   = (oxygen*match).f
match  = match
oxygen = oxygen
```

Program 4.4 was another variation representing Example 1.7. The following listening contains this program in the input language of *cgraphs*.

```
1  f  :: fire :- match, not ab.
2  ab :: ab   :- noxygen.
3
4  :- oxygen, noxygen.
5
6  :: match.
7  :: oxygen.
```

The following listing shows the unique stable model of this program that coincides with the above stable model of Program 4.3.

```
Answer 1:
fire match oxygen
fire    = match.f
match   = match
oxygen  = oxygen
```

Program 4.6.

```
a :: p :- not q.
b :: q :- not p.
```

The following listing shows the two stable models of this program.

```
Answer 1:
p

p = a

Answer 2:
q

q = b
```

Each of the two causal stable models showed in above coincidences in the truth with one standard stable model.

Program 4.7.

```
f :: fire   :- match, oxygen.
d :: oxygen :- not noxygen.

:- oxygen, noxygen.

:: match.
:: oxygen.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
fire match oxygen

fire    = (d*match).f + (oxygen*match).f
match   = match
oxygen  = oxygen + d
```

Program 4.10.

```
a :: p.

q :- not r.
r :- not q.
p :- q.
p :- r.
```

The following listing shows the well-founded model of this program by means of the least and greatest fixpoints of the Γ_P^2 operator.

```
Least Fixpoint
p

p = a

Greatest Fixpoint:
p q r
```

The following listing shows the unique stable model of this program.

```
Answer 1:
p q

Answer 2:
p r
```

Recall that, true literals whose causal term is not show is 1, that is, $I(p) = 1$ in both stable models, while $W(p) = a$ in the well-founded causal mode.

The following listing shows the program representing the Yale Shooting scenario of Example 5.1.

```
time(1..3).

d(S) :: dead(S+1)    :- shoot(S), loaded(S).
o(S) :: loaded(S+1) :- load(S).

dead(S+1)    :- dead(S),    not ndead(S+1),    time(S).
loaded(S+1) :- loaded(S), not nloaded(S+1), time(S).

:: load(1).
:: shoot(3).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
dead(4) load(1) loaded(2) loaded(3) loaded(4) shoot(3) time(1)
time(2) time(3)

dead(4)    = (shoot(3)*load(1).o(1)).d(3)
load(1)    = load(1)
loaded(2)  = load(1).o(1)
loaded(3)  = load(1).o(1)
loaded(4)  = load(1).o(1)
shoot(3)   = shoot(3)
```

In the following, we show the three different representation of Example 5.2. We start for the representation that corresponds to *loaded* and *dead* following the symmetrical behaviour.

```
time(1..4).

d(S)    :: dead(S+1)      :- loaded(A,S), shoot(A,S).
o(A,S)  :: loaded(A, S+1) :- load  (A,S).

dead(S+1)      :- dead(S),      not ndead(S+1),      time(S).
loaded(A,S+1) :- loaded(A,S), not nloaded(A,S+1), time(S).


:: load(suzy,1).
:: shoot(suzy,3).

:: load(billy,2).
:: shoot(billy,4).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
dead(4) dead(5) load(billy,2) load(suzy,1) loaded(billy,3)
loaded(billy,4) loaded(billy,5) loaded(suzy,2) loaded(suzy,3)
loaded(suzy,4) loaded(suzy,5) shoot(billy,4) shoot(suzy,3)
time(1) time(2) time(3) time(4)

dead(4)          = (shoot(suzy,3)*load(suzy,1).o(suzy,1)).d(3)
dead(5)          = (shoot(billy,4)*load(billy,2).o(billy,2)).d(4)
                 + (shoot(suzy,3)*load(suzy,1).o(suzy,1)).d(3)
load(billy,2)    = load(billy,2)
load(suzy,1)     = load(suzy,1)
loaded(billy,3)  = load(billy,2).o(billy,2)
loaded(billy,4)  = load(billy,2).o(billy,2)
loaded(billy,5)  = load(billy,2).o(billy,2)
loaded(suzy,2)   = load(suzy,1).o(suzy,1)
loaded(suzy,3)   = load(suzy,1).o(suzy,1)
loaded(suzy,4)   = load(suzy,1).o(suzy,1)
loaded(suzy,5)   = load(suzy,1).o(suzy,1)
shoot(billy,4)   = shoot(billy,4)
shoot(suzy,3)    = shoot(suzy,3)
```

It is easy to see that, under the symmetrical behaviour, both Suzy and Billy are equally alternative causes of *dead*(5). On the other hand, the following listing shows the same example under the inertial preference behaviour.

```
time(1..4).

d(S)    :: dead(S+1)      :- shoot(A,S), loaded(A,S), not dead(S).
o(A,S)  :: loaded(A, S+1) :- load  (A,S), not loaded(A,S).

dead(S+1)       :- dead(S),      not ndead(S+1),     time(S).
loaded(A,S+1)   :- loaded(A,S),  not nloaded(A,S+1), time(S).

:: load(suzy,1).
:: shoot(suzy,3).

:: load(billy,2).
:: shoot(billy,4).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
dead(4) dead(5) load(billy,2) load(suzy,1) loaded(billy,3)
loaded(billy,4) loaded(billy,5) loaded(suzy,2) loaded(suzy,3)
loaded(suzy,4) loaded(suzy,5) shoot(billy,4) shoot(suzy,3)
time(1) time(2) time(3) time(4)

dead(4)          = (shoot(suzy,3)*load(suzy,1).o(suzy,1)).d(3)
dead(5)          = (shoot(suzy,3)*load(suzy,1).o(suzy,1)).d(3)
load(billy,2)    = load(billy,2)
load(suzy,1)     = load(suzy,1)
loaded(billy,3)  = load(billy,2).o(billy,2)
loaded(billy,4)  = load(billy,2).o(billy,2)
loaded(billy,5)  = load(billy,2).o(billy,2)
loaded(suzy,2)   = load(suzy,1).o(suzy,1)
loaded(suzy,3)   = load(suzy,1).o(suzy,1)
loaded(suzy,4)   = load(suzy,1).o(suzy,1)
loaded(suzy,5)   = load(suzy,1).o(suzy,1)
shoot(billy,4)   = shoot(billy,4)
shoot(suzy,3)    = shoot(suzy,3)
```

Here, we can see that, under the inertial preference behaviour, only Suzy, who shoots early, is considered to be a cause of *dead*(5). Similarly, the following listing shows this example under the causal preference behaviour.

```
time(1..4).

d(S)      :: dead(S+1)       :- shoot(A,S), loaded(A,S).
o(A,S)    :: loaded(A, S+1) :- load (A,S).

noinertial(dead(S+1))       :- shoot(A,S), loaded(A,S).
noinertial(loaded(A, S+1)) :- load (A,S).

loaded(A,S) :- caused(loaded(A,S)).
dead(S)     :- caused(dead(S)).



loaded(A,S+1) :- loaded(A,S), time(S),
                     not noinertial( loaded(A,S+1)).

dead(S+1)     :- dead(S), time(S),
                     not noinertial(  dead(S+1)).

:: load(suzy,1).
:: shoot(suzy,3).

:: load(billy,2).
:: shoot(billy,4).

#hide noinertial(_).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
dead(4) dead(5) load(billy,2) load(suzy,1) loaded(billy,3)
loaded(billy,4) loaded(billy,5) loaded(suzy,2) loaded(suzy,3)
loaded(suzy,4) loaded(suzy,5) shoot(billy,4) shoot(suzy,3)
time(1) time(2) time(3) time(4)

dead(4)          = (shoot(suzy,3)*load(suzy,1).o(suzy,1)).d(3)
dead(5)          = (shoot(billy,4)*load(billy,2).o(billy,2)).d(4)
load(billy,2)    = load(billy,2)
load(suzy,1)     = load(suzy,1)
loaded(billy,3)  = load(billy,2).o(billy,2)
loaded(billy,4)  = load(billy,2).o(billy,2)
loaded(billy,5)  = load(billy,2).o(billy,2)
loaded(suzy,2)   = load(suzy,1).o(suzy,1)
loaded(suzy,3)   = load(suzy,1).o(suzy,1)
loaded(suzy,4)   = load(suzy,1).o(suzy,1)
loaded(suzy,5)   = load(suzy,1).o(suzy,1)
shoot(billy,4)   = shoot(billy,4)
shoot(suzy,3)    = shoot(suzy,3)
```

In this case, just Billy is considered to be the cause of *dead*(5).

The following listings shows the program corresponding to the suitcase scenario of Example 1.4. First, we show the representation under the inertial preference behaviour.

```
time(1..6).

o(S)     :: open(S)   :- up(a,S), up(b,S), not open(S-1).
k(S)     :: open(S+1) :- key(S), not open(S).
u(L,S)   :: up(L,S+1) :- lift(L,S), not up(L,S).

open(S+1)    :- open(S), not nopen(S+1), time(S).

up(L,S+1)    :- up(L,S), not nup(L,S+1), time(S).

lift(a,1) :: lift(a,1).
lift(b,3) :: lift(b,3).

key(4)    :: key(4).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
key(4) lift(a,1) lift(b,3) open(4) open(5) open(6) open(7)
time(1) time(2) time(3) time(4) time(5) time(6) up(a,2)
up(a,3) up(a,4) up(a,5) up(a,6) up(a,7) up(b,4) up(b,5)
up(b,6) up(b,7)

key(4)     = key(4)
lift(a,1)  = lift(a,1)
lift(b,3)  = lift(b,3)
open(4)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(5)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(6)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(7)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
up(a,2)    = lift(a,1).u(a,1)
up(a,3)    = lift(a,1).u(a,1)
up(a,4)    = lift(a,1).u(a,1)
up(a,5)    = lift(a,1).u(a,1)
up(a,6)    = lift(a,1).u(a,1)
up(a,7)    = lift(a,1).u(a,1)
up(b,4)    = lift(b,3).u(b,3)
up(b,5)    = lift(b,3).u(b,3)
up(b,6)    = lift(b,3).u(b,3)
up(b,7)    = lift(b,3).u(b,3)
```

And we also introduce the same scenario under the causal preference behaviour.

```
time(1..6).

o(S)     :: open(S)   :- up(a,S), up(b,S).
k(S)     :: open(S+1) :- key(S).
u(L,S)   :: up(L,S+1) :- lift(L,S).

noinertial(open(S))    :- up(a,S), up(b,S).
noinertial(open(S+1)) :- key(S).
noinertial(up(L,S+1)) :- lift(L,S).

open(S+1) :- open(S), not noinertial(open(S+1)), time(S).

up(L,S+1) :- up(L,S), not noinertial(up(L,S+1)), time(S).

lift(a,1) :: lift(a,1).
lift(b,3) :: lift(b,3).

key(4)    :: key(4).

#hide noinertial(_).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
key(4) lift(a,1) lift(b,3) open(4) open(5) open(6) open(7)
time(1) time(2) time(3) time(4) time(5) time(6) up(a,2)
up(a,3) up(a,4) up(a,5) up(a,6) up(a,7) up(b,4) up(b,5)
up(b,6) up(b,7)

key(4)     = key(4)
lift(a,1) = lift(a,1)
lift(b,3) = lift(b,3)
open(4)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(4)
open(5)    = key(4).k(4)
           + (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(5)
open(6)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(6)
open(7)    = (lift(b,3).u(b,3)*lift(a,1).u(a,1)).o(7)
up(a,2)    = lift(a,1).u(a,1)
up(a,3)    = lift(a,1).u(a,1)
up(a,4)    = lift(a,1).u(a,1)
up(a,5)    = lift(a,1).u(a,1)
up(a,6)    = lift(a,1).u(a,1)
up(a,7)     =  lift(a,1).u(a,1)
up(b,4)     =  lift(b,3).u(b,3)
up(b,5)     =  lift(b,3).u(b,3)
up(b,6)     =  lift(b,3).u(b,3)
up(b,7)     =  lift(b,3).u(b,3)
```

Note that under causal preference behaviour, the causes of *open* changes from situation 4 to situation 5 after the key is turned.

The following listing shows the program representing the gears wheel scenario of Example 5.4.

```
time(0..6).

m(W,S)    ::  motor(W,S+1) :-  start(W,S).
nm(W,S)   ::  nmotor(W,S+1) :-  stop(W,S).
p(S)      ::  coupled(S+1) :-  couple(S).
p(S)      ::  ncoupled(S+1) :-  uncouple(S).


noinertial(motor(W,S+1)) :-  start(W,S).
noinertial(motor(W,S+1)) :-  stop(W,S).
noinertial(coupled(S+1)) :-  couple(S).
noinertial(coupled(S+1)) :-  uncouple(S).


r(W,S)    ::   turn(W,S) :-  motor(W,S), not ab(turn,W,W,S).
t(a,S)    ::   turn(a,S) :-  turn(b,S), coupled(S),
                             not turn(a,S-1), not ab(turn,b,a,S).
t(b,S)    ::   turn(b,S) :-   turn(a,S), coupled(S),
                             not turn(b,S-1), not ab(turn,a,b,S).
nt(a,S)   ::  nturn(a,S) :-  nturn(b,S), coupled(S),
                             not nturn(b,S-1), not ab(turn,a,b,S).
nt(b,S)   ::  nturn(b,S) :-  nturn(a,S), coupled(S),
                            not nturn(a,S-1), not ab(turn,b,a,S).


noinertial(turn(W,S))   :-  motor(W,S), not ab(turn,W,W,S).
noinertial(turn(a,S))   :-  turn(b,S), coupled(S), not turn(a,S-1),
                                                   not ab(turn,b,a,S).
noinertial(turn(b,S))   :-   turn(a,S), coupled(S),not turn(b,S-1),
                                                   not ab(turn,a,b,S).
noinertial(turn(a,S)) :-  nturn(b,S), coupled(S), not nturn(b,S-1),
                                                   not ab(turn,a,b,S).
noinertial(turn(b,S)) :-  nturn(a,S), coupled(S), not nturn(a,S-1),
                                                   not ab(turn,b,a,S).
```

```
 coupled(S+1) :-  coupled(S), not ncoupled(S+1), time(S).
ncoupled(S+1) :- ncoupled(S), not  coupled(S+1), time(S).

 motor(W,S+1) :-  motor(W,S), not nmotor(W,S+1), time(S).
nmotor(W,S+1) :- nmotor(W,S), not  motor(W,S+1), time(S).

 turn(W,S+1)  :-  turn(W,S), not noinertial(turn(W,S+1)), time(S).
nturn(W,S+1)  :- nturn(W,S), not noinertial(turn(W,S+1)), time(S).


:: nmotor(a,0).
:: nmotor(b,0).

:: nturn(a,0).
:: nturn(b,0).

:: ncoupled(0).

:: start(a,2).
:: couple(3).
:: uncouple(5).

:- turn(W,S), nturn(W,S).

#hide noinertial(_).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
couple(3) coupled(4) coupled(5) motor(a,3) motor(a,4)
motor(a,5) motor(a,6) motor(a,7) ncoupled(0) ncoupled(1)
ncoupled(2) ncoupled(3) ncoupled(6) ncoupled(7) nmotor(a,0)
nmotor(a,1) nmotor(a,2) nmotor(b,0) nmotor(b,1) nmotor(b,2)
nmotor(b,3) nmotor(b,4) nmotor(b,5) nmotor(b,6) nmotor(b,7)
nturn(a,0) nturn(a,1) nturn(a,2) nturn(b,0) nturn(b,1)
nturn(b,2) nturn(b,3) start(a,2) time(0) time(1) time(2)
time(3) time(4) time(5) time(6) turn(a,3) turn(a,4) turn(a,5)
turn(a,6) turn(a,7) turn(b,4) turn(b,5) turn(b,6) turn(b,7)
uncouple(5)

couple(3)    = couple(3)
coupled(4)   = couple(3).p(3)
coupled(5)   = couple(3).p(3)
motor(a,3)   = start(a,2).m(a,2)
motor(a,4)   = start(a,2).m(a,2)
motor(a,5)   = start(a,2).m(a,2)
motor(a,6)   = start(a,2).m(a,2)
motor(a,7)   = start(a,2).m(a,2)
ncoupled(0)  = ncoupled(0)
ncoupled(1)   = ncoupled(0)
ncoupled(2)   = ncoupled(0)
ncoupled(3)   = ncoupled(0)
ncoupled(6)   = uncouple(5).p(5)
ncoupled(7)   = uncouple(5).p(5)
nmotor(a,0)   = nmotor(a,0)
nmotor(a,1)   = nmotor(a,0)
nmotor(a,2)   = nmotor(a,0)
nmotor(b,0)   = nmotor(b,0)
nmotor(b,1)   = nmotor(b,0)
nmotor(b,2)   = nmotor(b,0)
nmotor(b,3)   = nmotor(b,0)
nmotor(b,4)   = nmotor(b,0)
nmotor(b,5)   = nmotor(b,0)
nmotor(b,6)   = nmotor(b,0)
nmotor(b,7)   = nmotor(b,0)
nturn(a,0)    = nturn(a,0)
nturn(a,1)    = nturn(a,0)
nturn(a,2)    = nturn(a,0)
nturn(b,0)    = nturn(b,0)
nturn(b,1)    = nturn(b,0)
nturn(b,2)    = nturn(b,0)
nturn(b,3)    = nturn(b,0)
start(a,2)    = start(a,2)
turn(a,3)     = start(a,2).m(a,2).r(a,3)
turn(a,4)     = start(a,2).m(a,2).r(a,4)
turn(a,5)     = start(a,2).m(a,2).r(a,5)
turn(a,6)     = start(a,2).m(a,2).r(a,6)
turn(a,7)     = start(a,2).m(a,2).r(a,7)
turn(b,4)     = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
turn(b,5)     = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
turn(b,6)     = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
turn(b,7)     = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
uncouple(5)   = uncouple(5)
```

The next program, adds the possibility of braking the wheels. Furthermore, the wheel *b* is braked at situation 6.

```
time(0..6).

m(W,S)     :: motor(W,S+1)  :- start(W,S).
nm(W,S)    :: nmotor(W,S+1) :- stop(W,S).
p(S)       :: coupled(S+1)  :- couple(S).
p(S)       :: ncoupled(S+1) :- uncouple(S).



noinertial(motor(W,S+1)) :- start(W,S).
noinertial(motor(W,S+1)) :- stop(W,S).
noinertial(coupled(S+1)) :- couple(S).
noinertial(coupled(S+1)) :- uncouple(S).


r(W,S)   ::   turn(W,S) :- motor(W,S), not ab(turn,W,W,S).
t(a,S)   ::   turn(a,S) :- turn(b,S), coupled(S),
                             not turn(a,S-1), not ab(turn,b,a,S).
t(b,S)   ::   turn(b,S) :-  turn(a,S), coupled(S),
                             not turn(b,S-1), not ab(turn,a,b,S).
nt(a,S) :: nturn(a,S) :- nturn(b,S), coupled(S),
                             not nturn(b,S-1), not ab(turn,a,b,S).
nt(b,S) :: nturn(b,S) :- nturn(a,S), coupled(S),
                            not nturn(a,S-1), not ab(turn,b,a,S).


noinertial(turn(W,S))   :- motor(W,S), not ab(turn,W,W,S).
noinertial(turn(a,S))   :- turn(b,S), coupled(S), not turn(a,S-1),
                                                  not ab(turn,b,a,S).
noinertial(turn(b,S))   :-  turn(a,S), coupled(S),not turn(b,S-1),
                                                  not ab(turn,a,b,S).
noinertial(turn(a,S)) :- nturn(b,S), coupled(S), not nturn(b,S-1),
                                                  not ab(turn,a,b,S).
noinertial(turn(b,S)) :- nturn(a,S), coupled(S), not nturn(a,S-1),
                                                  not ab(turn,b,a,S).


 coupled(S+1) :-  coupled(S), not ncoupled(S+1), time(S).
ncoupled(S+1) :- ncoupled(S), not  coupled(S+1), time(S).

 motor(W,S+1) :-  motor(W,S), not nmotor(W,S+1), time(S).
nmotor(W,S+1) :- nmotor(W,S), not  motor(W,S+1), time(S).

 turn(W,S+1)  :-  turn(W,S), not noinertial(turn(W,S+1)), time(S).
nturn(W,S+1)  :- nturn(W,S), not noinertial(turn(W,S+1)), time(S).


:: nmotor(a,0).
:: nmotor(b,0).

:: nturn(a,0).
:: nturn(b,0).

:: ncoupled(0).

:: start(a,2).
:: couple(3).
:: uncouple(5).

:- turn(W,S), nturn(W,S).
```

```
#hide noinertial(_).


b(W,S)     ::  braked(W,S+1)  :-  brake(W,S).
nb(W,S)    ::  nbraked(W,S+1) :-  unbrake(W,S).
nr(W)      ::  nturn(W,S)     :-  braked(W,S), not ab(turn,W,W,S).

noinertial(turn(W,S)) :- braked(W,S), not ab(turn,W,W,S).

 braked(W,S+1) :-  braked(W,S), not noinertial(braked(W,S+1)),
                                                    time(S).
nbraked(W,S+1) :- nbraked(W,S), not noinertial(braked(W,S+1)),
                                                    time(S).

brake(b,6).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
brake(b,6) braked(b,7) couple(3) coupled(4) coupled(5)
motor(a,3) motor(a,4) motor(a,5) motor(a,6) motor(a,7)
ncoupled(0) ncoupled(1) ncoupled(2) ncoupled(3) ncoupled(6)
ncoupled(7) nmotor(a,0) nmotor(a,1) nmotor(a,2) nmotor(b,0)
nmotor(b,1) nmotor(b,2) nmotor(b,3) nmotor(b,4) nmotor(b,5)
nmotor(b,6) nmotor(b,7) nturn(a,0) nturn(a,1) nturn(a,2)
nturn(b,0) nturn(b,1) nturn(b,2) nturn(b,3) nturn(b,7)
start(a,2) time(0) time(1) time(2) time(3) time(4) time(5)
time(6) turn(a,3) turn(a,4) turn(a,5) turn(a,6) turn(a,7)
turn(b,4) turn(b,5) turn(b,6) uncouple(5)

braked(b,7)  = b(b,6)
couple(3)    = couple(3)
coupled(4)   = couple(3).p(3)
coupled(5)   = couple(3).p(3)
motor(a,3)   = start(a,2).m(a,2)
motor(a,4)   = start(a,2).m(a,2)
motor(a,5)   = start(a,2).m(a,2)
motor(a,6)   = start(a,2).m(a,2)
motor(a,7)   = start(a,2).m(a,2)
ncoupled(0)  = ncoupled(0)
ncoupled(1)  = ncoupled(0)
ncoupled(2)  = ncoupled(0)
ncoupled(3)  = ncoupled(0)
ncoupled(6)  = uncouple(5).p(5)
ncoupled(7)  = uncouple(5).p(5)
nmotor(a,0)  = nmotor(a,0)
nmotor(a,1)  = nmotor(a,0)
nmotor(a,2)  = nmotor(a,0)
nmotor(b,0)  = nmotor(b,0)
nmotor(b,1)  = nmotor(b,0)
nmotor(b,2)  = nmotor(b,0)
nmotor(b,3)  = nmotor(b,0)
nmotor(b,4)  = nmotor(b,0)
nmotor(b,5)  = nmotor(b,0)
nmotor(b,6)  = nmotor(b,0)
nmotor(b,7)  = nmotor(b,0)
nturn(a,0)   = nturn(a,0)
nturn(a,1)   = nturn(a,0)
nturn(a,2)   = nturn(a,0)
nturn(b,0)   = nturn(b,0)
nturn(b,1)   = nturn(b,0)
nturn(b,2)   = nturn(b,0)
nturn(b,3)   = nturn(b,0)
nturn(b,7)   = b(b,6).nr(b)
start(a,2)   = start(a,2)
turn(a,3)    = start(a,2).m(a,2).r(a,3)
turn(a,4)    = start(a,2).m(a,2).r(a,4)
turn(a,5)    = start(a,2).m(a,2).r(a,5)
turn(a,6)    = start(a,2).m(a,2).r(a,6)
turn(a,7)    = start(a,2).m(a,2).r(a,7)
turn(b,4)    = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
turn(b,5)    = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
turn(b,6)    = (couple(3).p(3)*start(a,2).m(a,2).r(a,4)).t(b,4)
uncouple(5)  = uncouple(5)
```

If we brake wheel *b* at situation 4 instead of at situation 6, we obtain no model. At situation 4 wheel *b* must be spinning because it is coupled to wheel *a* which is spinning, and it must be stopped because it is braked.

Next, the World Block Scenario of Example 5.3.

```
time(0..4).
block(B) :- on(B,L,0).

m(B,L,S)   :: on(B,L,S+1) :- move(B,L,S), not on(B,L,S).

noinertial(on(B,L,S+1)) :- move(B,L,S), not on(B,L,S).

on(B,L,S+1) :- on(B,L,S), not noinertial(on,B,S+1), time(S).

:- on(B1,B,S), on(B2,B), B1 != B2, block(B).
:- move(B1,B2,S), move(B2,L,S).

on(bA,table,0).
on(bB,table,0).
on(bC,table,0).
```

```
:: move(bB,bA,0).
:: move(bC,bB,2).

#hide noinertial(_).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
block(bA) block(bB) block(bC) move(bB,bA,0) move(bC,bB,2)
on(bA,table,0) on(bA,table,1) on(bA,table,2) on(bA,table,3)
on(bA,table,4) on(bA,table,5) on(bB,bA,1) on(bB,bA,2)
on(bB,bA,3) on(bB,bA,4) on(bB,bA,5) on(bB,table,0)
on(bB,table,1) on(bB,table,2) on(bB,table,3) on(bB,table,4)
on(bB,table,5) on(bC,bB,3) on(bC,bB,4) on(bC,bB,5)
on(bC,table,0) on(bC,table,1) on(bC,table,2) on(bC,table,3)
on(bC,table,4) on(bC,table,5) time(0) time(1) time(2) time(3)
time(4)

move(bB,bA,0) = move(bB,bA,0)
move(bC,bB,2) = move(bC,bB,2)
on(bB,bA,1)   = move(bB,bA,0).m(bB,bA,0)
on(bB,bA,2)   = move(bB,bA,0).m(bB,bA,0)
on(bB,bA,3)   = move(bB,bA,0).m(bB,bA,0)
on(bB,bA,4)   = move(bB,bA,0).m(bB,bA,0)
on(bB,bA,5)   = move(bB,bA,0).m(bB,bA,0)
on(bC,bB,3)   = move(bC,bB,2).m(bC,bB,2)
on(bC,bB,4)   = move(bC,bB,2).m(bC,bB,2)
on(bC,bB,5)   = move(bC,bB,2).m(bC,bB,2)
```

In the following, we will show the implementation of those programs considered in Chapter 8. We start by Program 8.1, which is a variation of Program 4.4:

```
f  :: fire :- match, not ab.
ab :: ab   :- noxygen.

:- oxygen, noxygen.

:: match.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
fire match

fire  = match.f
match = match
```

The following listing shows Program 7.1.

```
len(1..2).

m(I) :: p(I+1) :- p(I), q(I), len(I).
n(I) :: q(I+1) :- p(I), q(I), len(I).

a :: p(1).
b :: p(1).

c :: q(1).
d :: q(1).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
len(1) len(2) p(1) p(2) p(3) q(1) q(2) q(3)

p(1) = b + a
p(2) = (d*a).m(1) + (c*a).m(1) + (d*b).m(1) + (c*b).m(1)
p(3) = ((d*a).n(1)*(d*a).m(1)).m(2)
     + ((d*b).n(1)*(d*b).m(1)).m(2)
     + ((c*a).n(1)*(c*a).m(1)).m(2)
     + ((c*b).n(1)*(c*b).m(1)).m(2)
q(1) = c + d
q(2) = (d*b).n(1) + (c*a).n(1) + (d*a).n(1) + (c*b).n(1)
q(3) = ((c*a).n(1)*(c*a).m(1)).n(2)
     + ((c*b).n(1)*(c*b).m(1)).n(2)
     + ((d*a).n(1)*(d*a).m(1)).n(2)
     + ((d*b).n(1)*(d*b).m(1)).n(2)
```

Recall that the interesting feature of this example was that the number of causes of a literal $p(n)$ was $2^{2^{n-1}}$. Note however, that *cgraphs* computes the causes making use of unique atom labelling. In such case, there are a constant number 4 of causes of $p(n)$. The following program shows that even with a unique atom labelling, there are examples where the number of causes is exponential in the size of the program.

```
len(1..2).

m(I) :: p(I+1) :- p(I), q(I), len(I).
p(I) :: p(I+1) :- p(I), q(I), len(I).
n(I) :: q(I+1) :- p(I), q(I), len(I).
q(I) :: q(I+1) :- p(I), q(I), len(I).

a :: p(1).
b :: p(1).
```

```
c :: q(1).
d :: q(1).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
len(1) len(2) p(1) p(2) p(3) q(1) q(2) q(3)

p(1) = b + a
p(2) = (c*a).m(1)
     + (d*a).m(1)
     + (c*a).p(1)
     + (c*b).p(1)
     + (d*b).m(1)
     + (d*a).p(1)
     + (d*b).p(1)
     + (c*b).m(1)
p(3) = ((d*a).n(1)*(d*a).p(1)).p(2)
     + ((d*a).q(1)*(d*a).p(1)).p(2)
     + ((c*a).n(1)*(c*a).m(1)).m(2)
     + ((d*b).q(1)*(d*b).p(1)).m(2)
     + ((c*b).q(1)*(c*b).m(1)).m(2)
     + ((d*a).q(1)*(d*a).m(1)).m(2)
     + ((c*b).q(1)*(c*b).p(1)).p(2)
     + ((d*a).n(1)*(d*a).m(1)).p(2)
     + ((d*b).n(1)*(d*b).p(1)).p(2)
     + ((c*b).q(1)*(c*b).p(1)).m(2)
     + ((d*b).n(1)*(d*b).m(1)).p(2)
     + ((d*b).n(1)*(d*b).p(1)).m(2)
     + ((d*a).q(1)*(d*a).m(1)).p(2)
     + ((c*b).n(1)*(c*b).p(1)).p(2)
     + ((d*b).q(1)*(d*b).p(1)).p(2)
     + ((c*b).n(1)*(c*b).p(1)).m(2)
     + ((d*a).q(1)*(d*a).p(1)).m(2)
     + ((c*a).n(1)*(c*a).m(1)).p(2)
     + ((c*a).q(1)*(c*a).p(1)).p(2)
     + ((c*b).n(1)*(c*b).m(1)).p(2)
     + ((c*b).n(1)*(c*b).m(1)).m(2)
     + ((c*a).n(1)*(c*a).p(1)).p(2)
     + ((d*a).n(1)*(d*a).m(1)).m(2)
     + ((d*b).n(1)*(d*b).m(1)).m(2)
     + ((d*a).n(1)*(d*a).p(1)).m(2)
     + ((c*a).q(1)*(c*a).m(1)).m(2)
     + ((d*b).q(1)*(d*b).m(1)).p(2)
     + ((d*b).q(1)*(d*b).m(1)).m(2)
     + ((c*a).q(1)*(c*a).p(1)).m(2)
     + ((c*a).n(1)*(c*a).p(1)).m(2)
     + ((c*b).q(1)*(c*b).m(1)).p(2)
```

```
      + ((c*a).q(1)*(c*a).m(1)).p(2)
q(1)  = c + d
q(2)  = (d*b).n(1)
      + (c*a).q(1)
      + (d*a).n(1)
      + (c*b).q(1)
      + (d*b).q(1)
      + (d*a).q(1)
      + (c*a).n(1)
      + (c*b).n(1)
q(3)  = ((d*b).n(1)*(d*b).m(1)).n(2)
      + ((c*b).q(1)*(c*b).p(1)).n(2)
      + ((c*b).q(1)*(c*b).m(1)).n(2)
      + ((c*b).n(1)*(c*b).m(1)).q(2)
      + ((d*b).q(1)*(d*b).m(1)).n(2)
      + ((c*a).q(1)*(c*a).m(1)).q(2)
      + ((c*b).n(1)*(c*b).p(1)).n(2)
      + ((d*a).n(1)*(d*a).m(1)).n(2)
      + ((c*b).q(1)*(c*b).p(1)).q(2)
      + ((c*a).q(1)*(c*a).p(1)).q(2)
      + ((c*b).n(1)*(c*b).m(1)).n(2)
      + ((d*b).q(1)*(d*b).m(1)).q(2)
      + ((c*a).n(1)*(c*a).m(1)).n(2)
      + ((c*a).q(1)*(c*a).p(1)).n(2)
      + ((c*a).n(1)*(c*a).p(1)).q(2)
      + ((d*b).q(1)*(d*b).p(1)).n(2)
      + ((d*a).n(1)*(d*a).p(1)).q(2)
      + ((d*a).n(1)*(d*a).m(1)).q(2)
      + ((d*a).q(1)*(d*a).p(1)).n(2)
      + ((d*b).n(1)*(d*b).p(1)).q(2)
      + ((d*a).q(1)*(d*a).m(1)).n(2)
      + ((c*a).q(1)*(c*a).m(1)).n(2)
      + ((c*b).q(1)*(c*b).m(1)).q(2)
      + ((d*a).q(1)*(d*a).p(1)).q(2)
      + ((d*a).q(1)*(d*a).m(1)).q(2)
      + ((c*a).n(1)*(c*a).m(1)).q(2)
      + ((d*b).n(1)*(d*b).m(1)).q(2)
      + ((c*a).n(1)*(c*a).p(1)).n(2)
      + ((d*b).q(1)*(d*b).p(1)).q(2)
      + ((c*b).n(1)*(c*b).p(1)).q(2)
      + ((d*a).n(1)*(d*a).p(1)).n(2)
      + ((d*b).n(1)*(d*b).p(1)).n(2)
```

Program 8.2 representing the load/unload process of the gun of the Yale Shooting scenario.

```
time(1..3).

o(S)  ::  loaded(S+1) :- load(S),   time(S).
u(S)  :: nloaded(S+1) :- unload(S), time(S).

loaded(S+1) :- loaded(S), not nloaded(S+1), time(S).

:: load(1).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
load(1) loaded(2) loaded(3) loaded(4) time(1) time(2) time(3)

load(1)    = load(1)
loaded(2)  = load(1).o(1)
loaded(3)  = load(1).o(1)
loaded(4)  = load(1).o(1)
```

Program 8.4.

```
r1  :: a :- c, not b.
r2  :: b :- not a.
c   :: c.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
a c

a = c.r1
c = c

Answer 2:
b c

b = r2
c = c
```

Program 8.5.

```
r1  :: a :- c, not b.
r2  :: b :- not a.
c   :: c.

r3  :: d :- b, not d.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
a c

a = c.r1
c = c
```

Program 8.6.

```
a :: a :- f, not b.
f :: f :- e.
b :: b :- e, not a.
d :: d :- c, e.
e :: e.
c :: c :- c, f.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
a e f

a = e.f.a
e = e
f = e.f

Answer 2:
b e f

b = e.b
e = e
f = e.f
```

Next, we show the implementation of Program 8.7 representing the decisions support system used by Dr. Smith in Example 8.1.

```
m :: tightOnMoney            :- student, not richParents.
p :: caresAboutPracticality :- likesSports.
r :: correctiveLens          :- shortSighted, not laserSurgery.
s :: laserSurgery            :- shortSighted, not tightOnMoney,
                                               not correctiveLens.
g :: glasses          :- correctiveLens, not caresAboutPracticality,
                                         not contactLens.
c :: contactLens      :- correctiveLens, not afraidToTouchEyes,
                                         not longSighted,
                                         not glasses.
i :: intraocularLens :- correctiveLens, not glasses, not contactLens.
```

```
:: shortSighted.
:: afraidToTouchEyes.
:: student.
:: likesSports.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
afraidToTouchEyes caresAboutPracticality correctiveLens
intraocularLens likesSports shortSighted student tightOnMoney

afraidToTouchEyes       = afraidToTouchEyes
caresAboutPracticality  = likesSports.p
correctiveLens          = shortSighted.r
intraocularLens         = shortSighted.r.i
likesSports             = likesSports
shortSighted            = shortSighted
student                 = student
tightOnMoney            = student.m
```

Next, Program 8.10 representing Example 1.5 where Suzy and Billy throw rocks to a bottle.

```
time(1..3).

:: shattered(T+1)   :-  throw(X,T), not shattered(T).
            shattered(T+1) :-  shattered(T), time(T).

:: throw(suzy,2).
:: throw(billy,4).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
shattered(3) shattered(4) throw(billy,4) throw(suzy,2) time(1)
time(2) time(3)

shattered(3)   = throw(suzy,2).shattered(3)
shattered(4)   = throw(suzy,2).shattered(3)
throw(billy,4) = throw(billy,4)
throw(suzy,2)  = throw(suzy,2)
```

Next, the variation of this program where John throws before Suzy.

```
time(1..3).

:: shattered(T+1)    :-  throw(X,T), not shattered(T).
               shattered(T+1) :-  shattered(T), time(T).

:: throw(suzy,2).
:: throw(billy,4).

:: throw(john,0).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
shattered(1) shattered(2) shattered(3) shattered(4)
throw(billy,4) throw(john,0) throw(suzy,2) time(1) time(2)
time(3)

shattered(1)    = throw(john,0).shattered(1)
shattered(2)    = throw(john,0).shattered(1)
shattered(3)    = throw(john,0).shattered(1)
shattered(4)    = throw(john,0).shattered(1)
throw(billy,4)  = throw(billy,4)
throw(john,0)   = throw(john,0)
throw(suzy,2)   = throw(suzy,2)
```

Program 8.11 representing the desert traveller scenario of Example 1.8.

```
:: death :- shoot, poison.
:: death :- shoot.
:: shoot.
:: poison.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
death poison shoot

death  = shoot.death
poison = poison
shoot  = shoot
```

Program 8.16 representing the same scenario, but including the intermediate variables *dehydration* and *intake*.

```
:: death        :- dehydration.
:: death        :- intake.
:: dehydration  :- shoot.
:: intake       :- nshoot, poison.
:: shoot.
:: poison.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
death dehydration poison shoot

death       = shoot.dehydration.death
dehydration = shoot.dehydration
poison      = poison
shoot       = shoot
```

Program 8.18 representing Example 8.2.

```
:: death :- shoot(suzy), load(john).
:: death :- shoot_load(billy).

:: shoot(suzy).
:: load(john).
:: shoot_load(billy).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
death load(john) shoot(suzy) shoot_load(billy)

death             = shoot_load(billy).death
                  + (shoot(suzy)*load(john)).death
load(john)        = load(john)
shoot(suzy)       = shoot(suzy)
shoot_load(billy) = shoot_load(billy)
```

Next, the program representing the variation of Example 8.3 in which Suzy does not shoot.

```
death             :: death :- shoot(suzy), load(john).
death             :: death :- shoot_load(billy).


load(john)        :: load(john).
shoot_load(billy) :: shoot_load(billy).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
death load(john) shoot_load(billy)

death            = shoot_load(billy).death
load(john)       = load(john)
shoot_load(billy) = shoot_load(billy)
```

Program 8.21 representing Example 8.4.

```
:: survive :- antidote.
:: survive :- not poison.

:: antidote.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
antidote survive

antidote = antidote
survive  = survive
```

And the variation in which the victim is poisoned.

```
:: survive :- antidote.
:: survive :- not poison.

:: antidote.
:: poison.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
antidote poison survive

antidote = antidote
poison   = poison
survive  = antidote.survive
```

Next, Program 8.22 representing Example 8.5.

```
agent(billy).
agent(putin).

    death        :- not ndeath, not promise.
    ndeath       :- not  death, not npromise.

     water       :- water(X).
     promise     :- promise(X).
    npromise     :- not promise.

::  ndeath       :- water(X).
::   death       :- nwater(X), not npromise(X), not water.

    npromise(X)  :- not promise(X), agent(X).

:: promise(billy).
:: nwater(billy).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
agent(billy) agent(putin) death npromise(putin) nwater(billy)
promise promise(billy)

death            = nwater(billy).death
nwater(billy)    = nwater(billy)
promise          = promise(billy)
promise(billy)   = promise(billy)
```

The variation in which Putin is considered, but does not promise to water the plant.

```
agent(billy).
agent(putin).

    death        :- not ndeath, not  promise.
    ndeath       :- not  death, not npromise.

     water       :- water(X).
     promise     :- promise(X).
    npromise     :- not promise.

::  ndeath       :- water(X).
::   death       :- nwater(X), not npromise(X), not nwater.

    npromise(X)  :- not promise(X), agent(X).
```

```
:: promise(billy).
:: nwater(billy).

:: nwater(putin).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
agent(billy) agent(putin) death npromise(putin) nwater(billy)
nwater(putin) promise promise(billy)

death           = nwater(billy).death
nwater(billy)   = nwater(billy)
nwater(putin)   = nwater(putin)
promise         = promise(billy)
promise(billy)  = promise(billy)
```

And the variation in which Putin promises to water the plant.

```
agent(billy).
agent(putin).

    death       :- not ndeath, not  promise.
    ndeath      :- not  death, not npromise.

     water      :- water(X).
     promise    :- promise(X).
    npromise    :- not promise.

::  ndeath      :- water(X).
::   death      :- nwater(X), not npromise(X), not nwater.

    npromise(X) :- not promise(X), agent(X).

:: promise(billy).
:: nwater(billy).

:: promise(putin).
:: nwater(putin).
```

The following listing shows the unique stable model of this program.

```
Answer 1:
agent(billy) agent(putin) death nwater(billy) nwater(putin)
promise promise(billy) promise(putin)

death            = nwater(putin).death + nwater(billy).death
nwater(billy)    = nwater(billy)
nwater(putin)    = nwater(putin)
promise          = promise(putin) + promise(billy)
promise(billy)   = promise(billy)
promise(putin)   = promise(putin)
```

Finally, Program 8.24 representing Example 8.6.

```
:: arrive :- left.
:: arrive :- right.
:: left   :- train, switch.
:: right  :- train, not switch.


:: train.
:: switch.
```

The following listing shows the unique stable model of this program.

```
Answer 1:
arrive left switch train

arrive = (switch*train).left.arrive
left   = (switch*train).left
switch = switch
train  = train
```

C | RESUMEN

Causalidad es un concepto presente en todo tipo de escenarios cotidianos y firmemente asentado en el razonamiento de sentido común. De hecho, ha aparecido en diferentes culturas, distantes tanto geográfica como temporalmente, y es uno de los objetivos centrales de muchos estudios en las ciencias físicas, conductuales, sociales y biológicas. El hecho de que la intuición sobre relaciones causa-efecto no sólo afecte al sentido común, sino que también se encuentra implícitamente presente en la ciencia o en el razonamiento formal, demuestran la importancia de la obtención de una formalización del concepto de causalidad. Sin embargo, su formalización ha sido un asunto difícil de alcanzar que genera desacuerdo entre expertos de diferentes campos. Pearl [2000] ilustra la importancia de ésta clase de relaciones poniendo como ejemplo la segunda ley de la mecánica de Newton, y como esta describe la manera en que una fuerza aplicada a un objeto *cambiará* su estado de movimiento:

> *"El cambio de movimiento es proporcional a la fuerza motriz impresa y ocurre según la línea recta a lo largo de la cual aquella fuerza se imprime."*

Esta ley es capturada por la bien conocida ecuación matemática:

$$a = \frac{f}{m} \tag{113}$$

La manera en que (113) está escrita contiene implícita una relación causa-efecto entre la fuerza f y la aceleración a que no se refleja en la semántica de la ecuación. De hecho, de acuerdo con las leyes del álgebra, esta ecuación se puede reescribir de forma equivalente como $f = m \cdot a$ o como $m = f/a$. Sin embargo, decimos que "la relación f/a nos ayuda a *determinar* la masa m" o que "la masa calculada m *explica* por qué una fuerza f dada ha provocado la aceleración observada," pero no que "esta fuerza f ha *causado* la masa m." Del mismo modo, la ecuación $f = m \cdot a$ nos ayuda a *planear* lo que tenemos que hacer para imprimir una determinada aceleración a un objeto de masa dada, pero esto no significa que las aceleraciones causen fuerzas. Generalmente, estas consideraciones causales son tenidas en cuenta cuando los físicos usan (113), pero, según explica Pearl, "tales distinciones no están soportadas por las ecuaciones de la física." Pearl [1988] también ha señalado que algo similar sucede cuando estamos formalizando una amplia clase de conocimiento abstracto en un formalismo lógico. Para ilustrar este hecho, considere el siguiente escenario introducido por Lin [1995]:

Ejemplo C.1 (La maleta)**.** *Una maleta tiene dos cerraduras y un muelle que abre la maleta cuando ambas cerraduras se elevan.* □

Una representación directa de este escenario podría ser la implicación:

$$up(a) \land up(b) \;\supset\; open \tag{114}$$

indicando que cuando ambas cerraduras están en posición up, la maleta se abre. Una teoría de lógica clásica que consista en la conjunción de la implicación anterior más el hecho de que ambas cerraduras están en posición up:

$$up(a) \;\land\; up(b)$$

nos conduce a la conclusión de que la maleta se abrió. Una lectura *de izquierda a derecha* de la *implicación material* puede llevarnos a reconocer erróneamente una relación causal en (114). Sin embargo, como sucede con (113), la implicación (114) también la podemos escribir de manera equivalente como:

$$up(a) \land \neg open \;\supset\; \neg up(b) \tag{115}$$

o

$$up(b) \land \neg open \;\supset\; \neg up(a) \tag{116}$$

En estos casos, debemos cambiar la intuición de estas expresiones: podemos *explicar* que una cerradura está up y la maleta está cerrada porque la otra cerradura no está up, pero *no consideramos que* $up(a) \land \neg open$ haya causado $\neg up(b)$.

Además, en algunos casos, explicar las causas que han llevado a que algún evento haya occurrido puede ser tan importante como la predicción de que vaya a suceder. Por ejemplo, considere la siguiente variación del escenario de la maleta.

Ejemplo C.2 (Ex. C.1 continuación)**.** *La maleta está conectado a un mecanismo que provoca la explosión de una bomba cuando esta se abre.* □

Podemos representar este nuevo escenario añadiendo la siguiente implicación:

$$open \supset bomb \tag{117}$$

Solemos estar interesados en asignar la responsabilidad de algún evento ocurrido a algún conjunto de hechos, especialmente en aquellos casos en que dicho evento tiene consecuencias importantes como puede ser el caso de la explosión de una bomba. Cuando se requiere una explicación de la bomba, podemos usar la fórmula (117) de nuevo en su lectura de izquierda a derecha para determinar que elevar ambas cerraduras explica la explosión de la bomba. Por otra parte, si se requiere una explicación detallada, podemos construir una *prueba deductiva*, como la mostrada en la Figura 53.

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{up(a)} \qquad \dfrac{\top}{up(b)}}{up(a) \wedge up(b)}}{open}\;(114)}{bomb}\;(117)$$

Figure 53: Prueba del átomo *bomb* correspondiente con el Ejemplo C.2.

Posiblemente un juez no estará satisfecho simplemente con una explicación física de la explosión de una bomba, sino que estará más preocupado por la aplicación de una ley que establece lo siguiente:

Ejemplo C.3 (Ex. C.2 continuación)**.** *Aquel que causare la explosión de una bomba será castigado con pena de prisión.* □

La formalización de esta frase es un *problema desafiante* para la *Representación del Conocimiento* (KR del inglés *Knowledge Representation*) porque habla de "las causas de una explosión" sin describir explícitamente *las posibles maneras en las que, eventualmente, la explosión puede ser causada*. Si buscamos una *solución tolerante a la elaboración*, la formalización de esta ley *no* debe variar cuando una nueva forma de causar la explosión sea incluida en la teoría. Un formalismo es *tolerante a la elaboración* en la medida en que se puede tener en cuenta nuevos fenómenos o cambios en las circunstancias modificando un conjunto de hechos expresados en dicho formalismo [McCarthy, 1998]. Con el fin de obtener una solución que cumpla este criterio, necesitaríamos una especie de predicado modal "hascaused(A,B)" donde A y B puedan ser formulas. Luego podemos codificar la ley del Ejemplo C.3 como:

$$hascaused(up(a), bomb) \supset prison \tag{118}$$

Es decir, si $up(a)$ es una causa de *bomb*, entonces podemos concluir que el agente que realizó $up(a)$ deberá ir a la cárcel. Obviamente, el problema viene cuando tratamos de dar un sentido al predicado "hascaused," ya que su verdad depende de la formalización del resto de la teoría. Por ejemplo, podemos derivar fácilmente hechos de este predicado que se concluyan de los efectos directos añadiendo las fórmulas:

$$up(a) \wedge up(b) \supset hascaused(up(a), open) \tag{119}$$
$$up(a) \wedge up(b) \supset hascaused(up(b), open) \tag{120}$$

También podemos añadir una fórmula de la forma:

$$open \supset hascaused(open, bomb) \tag{121}$$

de manera que concluyamos una causa de *bomb* para los efectos indirectos. Desafortunadamente, este método no tiene en cuenta el comportamiento transitivo que siguen las relaciones de causa-efecto: los hechos $up(a)$ and $up(b)$ *eventualmente* también causan *bomb*. Por supuesto, también podríamos añadir las siguientes implicaciones:

$$up(a) \wedge up(b) \supset hascaused(up(a), bomb) \tag{122}$$
$$up(a) \wedge up(b) \supset hascaused(up(b), bomb) \tag{123}$$

pero, en el caso general, podemos tener muchas pasos intermedios e incluso interacción con el comportamiento por defecto de la inercia o con fluentes recursivos. En otras palabras, necesitaríamos un *análisis completo* de la teoría en su conjunto con el fin de concluir los axiomas correctos para predicado el "hascaused." Por ejemplo, bajo la anterior representación, la adición de un nuevo mecanismo que permita abrir la maleta no se puede lograr con sólo añadir nuevos hechos, sino que también se requiere la inclusión de nuevas fórmulas que relacionen este mecanismo con la explosión, es decir, nos vemos obligados a añadir nuevas fórmulas en forma de efectos directos que conecten causalmente estas nuevas formas de abrir la maleta y la explosión, *bomb*. Este es un ejemplo del llamado *problema de ramificación*, identificado por Kautz [1986], que consiste en verse forzado a representar los *efectos indirectos* de las acciones como *efectos directos*.

Un segundo inconveniente de (114), viene de la asignación de una lectura de izquierda a derecha de la *implicación material*. Es importante resaltar que la diferencia entre (114), (115) y (116) simplemente atañe a su escritura, pero todas ellas son equivalentes, ya que la implicación material es simplemente un caso de disyunción, en este caso:

$$open \ \vee \ \neg up(a) \ \vee \ \neg up(b)$$

Usar la correcta lectura causal de (114) se vuelve ahora *crucial* debido a que estamos fijando la verdad del predicado "hascaused" en función de ello. Para ser preciso, mientras que de (114) queremos concluir que:

$$(114) \ \wedge \ up(a) \ \wedge \ up(b) \ \models \ hascaused(up(a), open) \tag{124}$$

Una lectura similar de (115) nos llevaría a concluir que:

$$(115) \ \wedge \ up(a) \ \wedge \ \neg open \ \models \ hascaused(\neg open, \neg up(b)) \tag{125}$$

algo que *no debería cumplirse* cuando (115) es remplazada por (114) en (125): la lectura causal deseada de (114) debería ir del antecedente (las cerraduras) hacia el consecuente (open) y no en la otra dirección. Desafortunadamente, ambas fórmulas son equivalentes en lógica clásica, y por eso, esta distinción no se pueden hacer.

Al contrario que la implicación clásica, las reglas en Programación Lógica (LP del inglés *Logic Programming*) son direccionales. Por ejemplo, (114) se escribiría como la regla:

$$open \leftarrow up(a), up(b) \tag{126}$$

El símbolo de implicación ($\supset$) es sustituido por una flecha ($\leftarrow$), el símbolo de conjunción ($\wedge$) por una coma (,) y la posición de antecedente y consecuente se invierten. Podemos leer (126), de manera *top-down*, como "para obtener $open$ podemos mover up ambas cerraduras" o, de manera *bottom-up*, como "subiendo ambas cerraduras se deriva que la maleta se abre." Incluso podemos asignarle una lectura causal: "moviendo up ambas cerraduras *causará* la apertura de la maleta." Por otro lado, siguiendo este mismo método de reescritura, (115) nos llevará a una regla diferente:

$$\overline{up}(b) \leftarrow up(a), \overline{open} \tag{127}$$

donde la negación clásica de un átomo $\neg\alpha$ es reescrita como $\overline{\alpha}$. La regla (127) se leería como "para mover up la cerradura b podemos mover up la cerradura

a y cerrar de alguna manera la maleta” o “mover up la cerradura y cerrar la maleta harán que la segunda cerradura se mueva hacia abajo,” algo claramente erróneo con respecto a nuestra percepción causal del Ejemplo C.1. Está claro que las lecturas de las reglas (126) y (127) son muy distintas, y de hecho todas las semántica de LP las tratarán como dos fórmulas claramente diferentes. Esta distinción hace que las reglas de LP sean una herramienta adecuada para la representación de *leyes causales*.

En Inteligencia Artificial (AI del ingles *Artificial Intelligence*), el problema de causalidad se ha abordado principalmente dos maneras diferentes y complementarias: por un lado, la literatura de *actual causation* se han centrado en determinar el concepto común de *causa* sin poner demasiada atención a los problemas de tolerancia a la elaboración que puedan surgir; por otro lado, los enfoques basados en suficiencia se han centrado en el uso de causalidad para resolver algunos problemas de tolerancia a la elaboración sin poner mucha atención en la posibilidad de concluir hechos de la forma “A ha causado B.” El núcleo de esta tesis se centra en la representación y razonamiento con explicaciones causales. Como punto de partida, vamos a representar a los sistemas como programas lógicos, y vamos a leer las reglas de la forma $A \leftarrow B$ como “el evento B causa efecto A.” Las explicaciones causales consisten en fórmulas en forma normal disyuntiva mínima en las que cada disjunto representa un causa en forma de grafo. En nuestro ejemplo, el átomo $bomb$ estará justificado por la siguiente causa:

$$\big(up(a) * up(b)\big) \cdot o \cdot b \tag{128}$$

donde o y b se corresponden con las reglas (126) y

$$bomb \leftarrow open \tag{129}$$

respectivamente. Intuitivamente, (128) significa que $up(a)$ y $up(b)$ han causado $bomb$ por medio de las reglas o y b (es decir (126) y (129), respectivamente). Esta representación permite fácilmente reconocer diferentes relaciones causales como son: *causa suficiente, necesaria y contributiva*.

METODOLOGÍA

La metodología utilizada en esta propuesta es la estándar en investigación en Ciencias de la Computación, una secuencia cíclica incluyendo: revisión del estado del arte, definición del problema, planteamiento de hipótesis, y derivación de su prueba formal o refutación. En concreto, esta tesis doctoral ha generado las siguientes publicaciones:

Pedro Cabalar and Jorge Fandinno, *Explaining Preferences and Preferring Explanations*. In Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation 2014, Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday. Thomas Eiter, Hannes Strass, Mirosław Truszczyński and Stefan Woltran (eds). Lecture Notes in Computer Science, Volume 9060, 2015.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *Causal graph justifications of logic programs.* In Theory and Practice of Logic Programming, TPLP 14, (4-5) 603-618, 30th International Conference on Logic Programming, July 2014.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *A complexity assessment for queries involving sufficient and necessary causes.* In Proc. of the 14th European Conf. on Logis in Artificial Inteligence, JELIA'14, Funchal, Madeira, Portugal, September 24th-26th, 2014. Lecture Notes in Artificial Intelligence (8761), pp. 300-310, Springer-Verlag, 2014.

Pedro Cabalar and Jorge Fandinno, *An algebra of causal chains*, in Proc. of the 6th International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP'13, Istambul, Turkey, 2013.

Jorge Fandinno, *Algebraic Approach to Causal Logic Programs*, Theory and Practice of Logic Programming 13 (4-5), On-line Supplement (Doctoral Consortium), 2013.

Pedro Cabalar and Jorge Fandinno, *Enablers and Inhibitors in Causal Justifications of Logic Programs*, Technical Report, University of Corunna, 2015.

RESULTADOS OBTENIDOS

En esta tesis, hemos proporcionado una semántica lógica para representar y razonar con explicaciones causales. En detalle, nuestras principales aportaciones pueden resumirse de la siguiente manera:

i) Se han definido formalmente los conceptos de grafo causal y valor causal que extienden la idea de cadena causal introducida por Lewis y que, a diferencia de ésta, permiten distinguir entre causas alternativas y conjuntivas. También se ha definido los conceptos de causa suficiente, necesaria y contributiva con respecto a estos valores causales. En particular, el concepto de *causa suficiente* está estrechamente relacionado con la idea de *prueba lógica no redundante*: de hecho, se ha demostrado su correspondencia isomorfa para el caso de aquellos programas en los que cada regla tiene una etiqueta diferente.

ii) Se han estudiado las propiedades algebraicas de estos valores causales. En particular, los valores causales pueden ser manipulados por medio de tres operaciones algebraicas $(\cdot)$, $(*)$ and $(+)$. Por otra parte, el álgebra formada por estos valores causales es isomorfo a un retículo libre, completamente distributivo generado por el conjunto de grafo causales. Como consecuencia, aquellos términos causales sin sumas que además son maximales representan el conjunto de causas de un átomo.

iii) Se han proporcionado semánticas causales para programas lógicos que son extensiones de la *least model*, la *stable model*, la *well-founded model* y la *answer set semantics*. También se han proporcionado métodos para calcular la información causal con respecto a estas semánticas. En concreto, la información causal con respecto a la *least model semantics* se puede calcular mediante una extensión del operador estándar de consecuencias directas introducido por van Emden and Kowalski [1976]. Para la *stable* y la *answer set semantics*, podemos recurrir a la idea de *reducto de un programa* [Gelfond and Lifschitz, 1988] y, finalmente, para la *well-founded semantics*, podemos utilizar una extensión de la definición proporcionada por Van Gelder [1989] en función de un operador de punto fijo alternante.

iv) Se ha explorado la aplicación de nuestra propuesta a problemas tradicionales de KR. En concreto, se ha aplicado la extensión causal de la *answer set semantics* para obtener relaciones de causa-efecto en ejemplos de la literatura de Razonamiento sobre acciones y cambio. La misma semántica también fúe empleada para representar varios ejemplos provenientes de la literatura sobre *actual causation*.

v) También se ha incorporado un nuevo tipo de literal causal que permite consultar la información causal asociada con cada literal. Estos literales permiten, a su vez, la definición de predicados concretos que definan distintos tipos de relaciones causa-efecto. Por ejemplo, hemos utilizado esta clase de literales para representar el Ejemplo C.3 mediante la regla no estándar (118) sin caer en *problemas de tolerancia a la elaboración* [McCarthy, 1998]. Para ello definimos la semántica del predicado *hascaused*(A, B) en función de un literal causal de manera que *hascaused*(A, B) es cierto cuando "el evento A ha sido suficiente para causar el evento B."

vi) Hemos explorado el coste computacional asociado con la solución de los problemas de decisión correspondientes la obtención de las *conclusiones lógicas*, las *causas suficientes* y las *causas necesarias* de un programa. Para todos ellos proporcionamos caracterizaciones completas, en concreto: todos ellos se encuentran dentro del segundo nivel de la jerarquía polinómica, y (bajo supuestos razonables) tanto el coste de la obtención de conclusiones lógicas, como la obtención de causas suficientes, no se ve incrementada con respecto a la obtención de conclusiones lógicas en LP estándar.

vii) Comparamos nuestro trabajo con varias aproximaciones en LP que tratan la obtención de justificaciones. En concreto, obtenemos una correspondencia formal entre nuestras *causas suficientes* y las justificaciones obtenidas por la aproximación *why-not provenance* [Damásio et al., 2013]. También exploramos, de maneira informal, las similitudes y diferencias entre nuestro enfoque y los enfoques seguidos por Pontelli et al. [2009] y Schulz and Toni [2013, 2014]. Con respecto a la literatura de *actual causation* observamos que nuestra aproximación corresponde con la idea de *producción* introducida por Hall [2004]. Hall distingue entre dos clases de relaciones causa-efecto que él llama *producción* y *dependencia*. En este sentido, observamos que la definición de causa contributiva que proponemos coincide con la definición de *actual cause* proporcionada por Halpern and Pearl [2001] en aquellos ejemplos en los que producción y dependencia coinciden.

viii) Finalmente, proporcionamos a implementación dun prototipo que calcula as causas suficientes de literales con respecto á *well-founded semantics* e la *stable model semantics*. Esta ferramenta incorpora o predicado *hascaused*(A, B) que permite también razoar coas relaciones de causa-efecto.

D RESUMO

Causalidade é un concepto presente en todo tipo de escenarios cotiáns, firmemente asentado no razoamento de sentido común. De feito, apareceu en diferentes culturas, distantes tanto xeográfica como temporalmente, e é un dos obxectivos centrais de moitos estudos nas ciencias físicas, da conduta, sociais e biolóxicas. O feito de que a intuición sobre relacións causa-efecto non so afecte ó sentido común, senón que tamén se atope implicitamente presente na ciencia ou no razoamento formal, demostran a importancia da obtención dunha formalización deste concepto. Sen embargo, a súa formalización foi un asunto difícil de acadar, e que xera desacordo entre expertos de diferentes campos. Pearl [2000] ilustra a importancia desta clase de relacions poñendo como exemplo a segunda lei da mecánica de Newton e como esta describe a maneira en que unha forza aplicada un obxecto *cambia o seu estado de movemento*:

> *"O cambio de movemento é proporcional á forza motriz impresa e ocorre na dirección da líña recta ó largo da cal dita forza se imprimiu."*

Esta lei é capturada pola ben coñecida ecuación matemática:

$$a = \frac{f}{m} \tag{130}$$

A maneira en que (130) está escrita contén implícita unha relación causa-efecto entre a forza f e a aceleración a que non se reflexa na semántica da ecuación. De feito, de acordo coas leis da álxebra, esta ecuación pódese reescribir de maneira equivalente como $f = m \cdot a$ ou como $m = f/a$. Sen embargo, dicimos que "a relación f/a axúdanos a *determinar* a masa m" ou que "a masa calculada m *explica* por que unha forza f dada provocou a aceleración observada," pero non que "esta forza f fora a *causa* da masa m." Do mesmo xeito, a ecuación $f = m \cdot a$ axúdanos a *planear* o que temos que facer para imprimir unha determinada aceleración a un obxecto de masa dada, pero isto non significa que as aceleracións causen forzas. Xeralmente, estas consideracións causais son tidas en conta cando os físicos usan (130), pero, segundo explica Pearl, "tales distincións non están soportadas polas ecuacións da física." Pearl [1988] tamén sinalou que algo similar sucede cando estamos formalizando unha ampla clase de coñecemento abstracto nun formalismo lóxico. Para ilustrar este feito, considere o seguinte escenario introducido por Lin [1995]:

Exemplo D.1 (La maleta)**.** *Unha maleta ten dous pechos e un resorte que abre a maleta cando ambos pechos se elevan.* □

Unha representación deste escenario podería ser a implicación:

$$up(a) \wedge up(b) \;\supset\; open \tag{131}$$

indicando que, cando ambos pechos están en posición up, a maleta ábrese. Unha teoría de lóxica clásica que consista na conxunción da implicación anterior mailo feito de que ambos pechos están en posición up:

$$up(a) \;\wedge\; up(b)$$

condúcenos á conclusión de que a maleta se abre. Unha lectura *de esquerda a dereita* da *implicación material* pode levarnos a recoñecer erroneamente unha relación causal en (131). Sen embargo, como sucede con (130), a implicación (131) tamén a podemos escribir de maneira equivalente como:

$$up(a) \wedge \neg open \;\supset\; \neg up(b) \tag{132}$$

ou

$$up(b) \wedge \neg open \;\supset\; \neg up(a) \tag{133}$$

Nestes casos, debemos cambiar a intuición destas expresións: podemos *explicar* que un pecho está up e a maleta segue pechada porque o outro pecho non está up, pero *non consideramos que* $up(a) \wedge \neg open$ causara $\neg up(b)$.

Ademais, nalgúns casos, explicar as causas que levaron a que certo evento ocorrera pode ser tan importante como a predición de que vaia ocorrer. Por exemplo, considere a seguinte variación do escenario da maleta.

Exemplo D.2 (Ex. D.1 continuación)**.** *A maleta está conectado a un mecanismo que provoca a explosión dunha bomba cando esta se abre.* □

Podemos representar este novo escenario engadindo a seguinte implicación:

$$open \;\supset\; bomb \tag{134}$$

Acostumamos a estar interesados en asignar a responsabilidade dalgún feito ocorrido a algún conxunto de feitos, especialmente naqueles casos nos que dito feito ten consecuencias relevantes como pode ser o caso da explosión dunha bomba. Cando se require unha explicación da explosión, podemos usar a fórmula (134) de novo na súa lectura de esquerda a dereita para determinar que elevar ambos pechos explica a explosión da bomba. Por outra parte, si se require una explicación detallada, podemos construír unha *proba dedutiva*, como a mostrada na Figura 54.

$$\dfrac{\dfrac{\dfrac{\dfrac{\top}{up(a)} \qquad \dfrac{\top}{up(b)}}{up(a) \wedge up(b)}}{open}\,(131)}{bomb}\,(134)$$

Figure 54: Proba do átomo *bomb* correspondente co Exemplo D.2.

Posiblemente un xuíz non estará satisfeito simplemente cunha explicación física da explosión da bomba, senón que estará máis preocupado pola aplicación dunha lei que establece o seguinte:

Exemplo D.3 (Ex. D.2 continuación)**.** *Quen causar a explosión dunha bomba será castigado con pena de prisión.* □

A formalización desta frase é un *problema desafiante* para a *Representación de Coñecemento* (KR do inglés *Knowledge Representation*) porque fala "das causas dunha explosión" sen describir explicitamente *as posibles maneiras en que, eventualmente, a explosión pode ser causada*. Se buscamos unha *solución tolerante á elaboración*, a formalización desta lei *non* debe variar cando unha nova forma de causar a explosión sexa incluída na teoría. Un formalismo é *tolerante á elaboración* na medida na que se pode ter en conta novos fenómenos ou cambios nas circunstancias modificando un conxunto de feitos expresados en dito formalismo [McCarthy, 1998]. Co fin de obter unha solución que cumpra este criterio, necesitaríamos una especie de predicado modal "hascaused(A,B)" onde A e B poidan ser formulas. Logo podemos codificar a lei do Exemplo D.3 como:

$$hascaused(up(a), bomb) \supset prison \tag{135}$$

É dicir, se $up(a)$ é unha causa de $bomb$, entón podemos concluír que o axente que realizou $up(a)$ deberá ir á cárcere. Claramente, o problema ven cando tratamos de dar un sentido ó predicado "hascaused," xa que a súa verdade depende da formalización do resto da teoría. Por exemplo, podemos derivar facilmente feitos deste predicado que se conclúan dos efectos directos engadindo as fórmulas:

$$up(a) \wedge up(b) \supset hascaused(up(a), open) \tag{136}$$
$$up(a) \wedge up(b) \supset hascaused(up(b), open) \tag{137}$$

Tamén podemos engadir unha fórmula da forma:

$$open \supset hascaused(open, bomb) \tag{138}$$

de maneira que conclúamos unha causa de *bomb* para os efectos indirectos. Desafortunadamente, este método non ten en conta o comportamento transitivo que seguen as relacións de causa-efecto: os feitos $up(a)$ and $up(b)$ *eventualmente* tamén causan *bomb*. Por suposto, tamén poderíamos engadir as seguintes implicacións:

$$up(a) \wedge up(b) \supset hascaused(up(a), bomb) \tag{139}$$
$$up(a) \wedge up(b) \supset hascaused(up(b), bomb) \tag{140}$$

pero, no caso xeral, podemos ter moitos pasos intermedios e incluso interacción co comportamento por defecto da inercia ou con flúentes recursivos. Noutras palabras, necesitaríamos un *análise completo* da teoría no seu conxunto co fin de concluír os axiomas correctos para o predicado "hascaused." Por exemplo, baixo a anterior representación, engadir un novo mecanismo que permita abrir a maleta non se pode acadar só con engadir novos feitos, senón que tamén se require a inclusión de novas fórmulas que relacionen este mecanismo coa explosión, é dicir, vémonos forzados a engadir novas fórmulas na forma de efectos directos que conecten causalmente estas novas formas de abrir a maleta e a explosión, *bomb*. Este é un exemplo do chamado *problema de ramificación*, identificado por Kautz [1986], que consiste en verse na obriga de representar os *efectos indirectos* das accións como *efectos directos*.

Un segunda inconveniente de (131), ven da asignación dunha lectura de esquerda a dereita da *implicación material*. É importante resaltar que a diferencia entre (131), (132) y (133) simplemente atane a súa escritura, pero todas elas son equivalentes, xa que a implicación material é simplemente un caso de disxunción, neste caso:

$$open \vee \neg up(a) \vee \neg up(b)$$

Usar a correcta lectura causal de (131) volvese agora *crucial* debido a que estamos fixando a verdade do predicado "hascaused" en función de elo. Para ser precisos, mentras que de (131) queremos concluír que:

$$(131) \wedge up(a) \wedge up(b) \models hascaused(up(a), open) \tag{141}$$

Unha lectura similar de (132) levarianos a concluír que:

$$(132) \wedge up(a) \wedge \neg open \models hascaused(\neg open, \neg up(b)) \tag{142}$$

algo que *non debería cumprirse* cando (132) é substituida por (131) en (142): a lectura causal desexada de (131) debería ir de antecedente (os pechos) cara o consecuente (open) e non na outra dirección. Desafortunadamente, ambas fórmulas son equivalentes en lóxica clásica, e polo tanto, esta distinción non é posible.

Ó contrario que a implicación clásica, as regras en Programación Lóxica (LP do inglés *Logic Programming*) son direccionais. Por exemplo, (131) escribiríase como a regra:

$$open \leftarrow up(a), up(b) \tag{143}$$

O símbolo de implicación ($\supset$) é substituído por unha frecha ($\leftarrow$), o símbolo de conxunción ($\wedge$) por unha coma (,) e a posición de antecedente e consecuente invértense. Podemos ler (143), de maneira *top-down*, como "para obter $open$ podemos mover up ambos pechos" ou, de maneira *bottom-up*, como "subindo ambos pechos derivase que a maleta se abre." Incluso podemos asignarlle unha lectura causal: "movendo up ambos pechos *causará* a apertura da maleta." Por outro lado, seguindo este mesmo método de reescritura, (132) levaranos a una regra diferente:

$$\overline{up}(b) \leftarrow up(a), \overline{open} \tag{144}$$

onde a negación clásica dun átomo $\neg\alpha$ é rescrita como $\overline{\alpha}$. A regra (144) leríase como "para mover up o pecho b podemos mover up o pecho a e cerrar dalgunha maneira a maleta" ou "mover up o pecho e pechar a maleta farán que o segundo pecho se mova cara abaixo," algo claramente erróneo con respecto a nosa percepción causal do Exemplo D.1. Está claro que as lecturas das regras (143) e (144) son moi distintas, e de feito todas as semántica de LP vanas tratar como dúas fórmulas claramente diferentes. Esta distinción fai que as regras de LP sexan unha ferramenta adecuada para a representación de *leis causais*.

En Intelixencia Artificial (AI do ingles *Artificial Intelligence*), o problema de causalidad abordouse principalmente de dúas maneiras diferentes e complementarias: por un lado, a literatura de *actual causation* centrouse en determinar o concepto común de *causa* sen por demasiada atención ós problemas de tolerancia a la elaboración que puideran xurdir; por outro lado, na area de Razoamento sobre Accións e Cambio, centráronse no uso de causalidade para resolver algúns problemas de tolerancia a elaboración sen por moita atención na posibilidade de concluír feitos da forma "A causou B." O núcleo desta tese centrase na representación e razoamento con explicaciones causais. Como punto de partida, imos representar ós sistemas como programas lóxicos, e imos ler as regras da forma $A \leftarrow B$ como "o fetio B causa o feito A." As explicacións causais consisten en fórmulas en forma normal disxuntiva mínima nas que cada disxunto representa un causa en forma de grafo. No noso exemplo, o átomo *bomb* estará xustificado pola seguinte causa:

$$\big(up(a) * up(b)\big) \cdot o \cdot b \tag{145}$$

onde o y b correspóndense coas regras (143) e

$$bomb \leftarrow open \tag{146}$$

respectivamente. Intuitivamente, (145) significa que $up(a)$ y $up(b)$ causou *bomb* por medio das regras o e b (é dicir (143) e (146), respectivamente). Esta representación permite facilmente recoñecer diferentes relacións causais como son: *causa suficiente, necesaria e contributiva*.

METODOLOXÍA

A metodoloxía empregada nesta proposta é a estándar en investigación nas Ciencias da Computación, unha secuencia cíclica incluíndo: revisión do estado da arte, definición do problema, formulación de hipóteses, e derivación da súa proba formal ou refutación. En concreto, esta tese doutoral xerou as seguintes publicacións:

Pedro Cabalar and Jorge Fandinno, *Explaining Preferences and Preferring Explanations*. In Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation 2014, Essays Dedicated to Gerhard Brewka on the Occasion of His 60th Birthday. Thomas Eiter, Hannes

Strass, Mirosław Truszczyński and Stefan Woltran (eds). Lecture Notes in Computer Science, Volume 9060, 2015.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *Causal graph justifications of logic programs.* In Theory and Practice of Logic Programming, TPLP 14, (4-5) 603-618, 30th International Conference on Logic Programming, July 2014.

Pedro Cabalar, Jorge Fandinno and Michael Fink. *A complexity assessment for queries involving sufficient and necessary causes.* In Proc. of the 14th European Conf. on Logis in Artificial Inteligence, JELIA'14, Funchal, Madeira, Portugal, September 24th-26th, 2014. Lecture Notes in Artificial Intelligence (8761), pp. 300-310, Springer-Verlag, 2014.

Pedro Cabalar and Jorge Fandinno, *An algebra of causal chains*, in Proc. of the 6th International Workshop on Answer Set Programming and Other Computing Paradigms, ASPOCP'13, Istambul, Turkey, 2013.

Jorge Fandinno, *Algebraic Approach to Causal Logic Programs*, Theory and Practice of Logic Programming 13 (4-5), On-line Supplement (Doctoral Consortium), 2013.

Pedro Cabalar and Jorge Fandinno, *Enablers and Inhibitors in Causal Justifications of Logic Programs*, Technical Report, University of Corunna, 2015.

RESULTADOS OBTIDOS

Nesta tese, proporcionamos unha semántica lóxica para representar e razoar con explicacións causais. En detalle, as nosas principais aportacións poden resumirse da seguinte maneira:

i) Definíronse formalmente os conceptos de grafo causal e valor causal que amplían a idea de cadea causal introducida por Lewis e que, a diferencia desta, permiten distinguir entre causas alternativas e conxuntivas. Tamén se definiron os conceptos de causa suficiente, necesaria e contributiva con respecto a estes valores causais. En particular, o concepto de *causa suficiente* está estreitamente ligado coa idea de *proba lóxica non redundante*: de feito, demostrouse a súa correspondencia isomorfa para o caso daqueles programas nos que cada regra ten una etiqueta diferente.

ii) Estudáronse as propiedades alxébricas destes valores causais. En particular, os valores causais poden ser manipulados por medio de tres operacións alxébricas $(\cdot)$, $(*)$ e $(+)$. Por outra parte, a álxebra formada por estes valores causais é isomorfa a un retículo libre, completamente distributivo xerado polo conxunto de grafos causais. Como consecuencia, aqueles termos causais sen sumas que ademais son maximais representan o conxunto de causas dun átomo.

iii) Proporcionáronse semánticas causais para programas lóxicos que son ampliacións da *least model*, a *stable model*, a *well-founded model* e a *answer set semantics*. Tamén se proporcionaron métodos para calcular a información causal con respecto a estas semánticas. En concreto, a información causal con respecto á *least model semantics* pode obterse mediante unha extensión dó operador estándar de consecuencias directas introducido por van Emden and Kowalski [1976]. Para a *stable* e a *answer set semantics*, podemos recorrer á idea de *reduto dun programa* [Gelfond and Lifschitz, 1988] e, finalmente, para a *well-founded semantics*, podemos empregar unha extensión da definición proporcionada por Van Gelder [1989] en función dun operador de punto fixo alternante.

iv) Explorouse a aplicación da nosa proposta a problemas tradicionais de KR. En concreto, aplicouse a extensión causal da *answer set semantics* para obter relacións de causa-efecto en exemplos da literatura de Razoamento sobre accións e cambio. A mesma semántica tamén foi empregada para representar varios exemplos provintes da literatura sobre *actual causation*.

v) Tamén se incorporou un novo tipo de literal causal que permite consultar a información asociada con cada literal. Estes literais permiten, a súa vez, a definición de predicados concretos que definan distintos tipos de relaciones causa-efecto. Por exemplo, empregamos esta clase de literais para representar o Exemplo D.3 mediante a regra non estándar (135) sen caer en *problemas de tolerancia á elaboración* [McCarthy, 1998]. Para elo, a semántica do predicado $hascaused(A, B)$ é definida en función dun literal causal de maneira que $hascaused(A, B)$ é certo cando "o evento A foi suficiente para causar o evento B."

vi) Exploramos o coste computacional asociado ca solución dos problemas de decisión correspondentes á obtención das *conclusións lóxicas*, as *causas suficientes* e as *causas necesarias* dun programa. Para todos eles proporcionamos caracterizacións completas, en concreto: todos estes problemas atópanse dentro do segundo nivel da xerarquía polinómica, e (baixo supostos razoables) tanto o coste da obtención de conclusións lóxicas, como a obtención de causas suficientes, non se ve incrementada con respecto á obtención de conclusións lóxicas en LP estándar.

vii) Comparamos o noso traballo con varias aproximaciones en LP que tratan a obtención de xustificacións. En concreto obtivemos unha correspondencia formal entre as nosas *causas suficientes* e as xustificacións obtidas pola aproximación *why-not provenance* [Damásio et al., 2013]. Tamén exploramos de maneira informal as similitudes e diferencias entre o noso enfoque e os enfoques seguidos por Pontelli et al. [2009] e Schulz and Toni [2013, 2014]. Con respecto á literatura de *actual causation* observamos que a nosa aproximación corresponde coa idea de *produción* introducida por Hall [2004]. Hall distingue entre dúas clases de relaciones causa-efecto que el chama *produción* e *dependencia*. Neste sentido, observamos que a nosa definición de causa contributiva coincide coa definición de *actual cause* proporcionada por Halpern and Pearl [2001] naqueles exemplos nos que produción e dependencia coinciden.

viii) Por último, proporcionamos a implementación dun prototipo que calcula as causas suficientes de literais con respecto á *well-founded semantics* e a *stable model semantics*. Esta ferramenta incorpora o predicado $hascaused(A, B)$ que permite tamén razoar coas relacións de causa-efecto.