Universidade da Coruña
Departamento de Computación

Pertinence for Causal Representation of Action Domains

Pedro Cabalar

Dissertation

presented at the *Facultade de Informática de A Coruña* following the requirements for the degree of

Doctor en Informática

A Coruña, 2001

Universidade da Coruña
Departamento de Computación

Ph.D. thesis

Pertinence for Causal Representation of Action Domains

Pedro Cabalar

Ph.D. advisor: Dr. Ramón P. Otero

Dissertation committee: Prof. Dr. Senén Barro Ameneiro (president)

Prof. Dr. Michael Gelfond

Dr. Héctor Geffner

Dr. David Pearce

Dr. Alessandro Provetti

Substitute members: Prof. Dr. Roque Marín Morales

Dr. Alvaro Barreiro García

To Carmen María

Acknowledgements

This work could not be conceived without the advice and direction of my supervisor, Ramón P. Otero. The origins of this research come from a reorientation from his initial interest in temporal expert systems into a more formal reasoning focusing. As a result, I have had the opportunity of directly witnessing the birth of his *Pertinence Logic*, from which this thesis represents, somehow, a free interpretation.

Very special thanks to my colleague, David Lorenzo, for his valuable help and close collaboration during all these years. I am also very grateful to Alessandro Provetti, for his support and encouragement, specially in my first steps in the area of nonmonotonic reasoning.

I am thankful to the members of the AI Lab. Manuel Cabarcos, Mario Otero-Díaz, José Manuel Rodríguez, Silvia Gómez, Oscar García, David Losada and Álvaro Barreiro for their fruitful discussions, specially during our "friday meetings," and for standing me during the coffee hours. I am also due to Alma Gómez and to the people at the Computer Science School of Ourense, where I spent two wonderful years, and to the members of the KR Lab. at the University of Texas at El Paso, for their hospitality during my stay there.

Finally, as personal acknowledgements, I want to thank the unconditional support of my parents, Pedro and Fina, along my whole career, and to my wife, Carmen María, for her real common sense advice and her dynamic personality (also known as impatience), without which I would have never finished.

This research has been supported by a grant of the Government of Spain (PB97-0228) and several grants of the Government of Galicia (XUGA-10503B96 and XUGA-10503B94) including a research stay grant for the author at the University of Texas at El Paso.

This document was generated using LaTeX 2E. Figures have been designed with the `xypic` LaTeX package and the graphical tools Xfig and Tgif. Escher's drawing in page 89 was downloaded from: `http://www.worldofescher.com/gallery/DrawingHands.html`

Abstract

This dissertation goes deeply into the use of causality for reasoning about actions and change, focusing not only in its application for solving typical representational problems, but also on causal knowledge as a relevant information *per se*. We show that causality is intimately related to the concept of *pertinence* with respect to the actions execution, and we propose a formal characterization of this idea, explaining its role in the semantics of causal rules. This allows us to introduce a high level language that provides a comfortable representation of action domains by using causal constructions. Besides, we present a unified view of the two existing orientations about causality, incorporating them at two different levels: at an ontological level, we use pertinence to capture the idea of causality versus persistence; whereas at an epistemological level, we propose different non-monotonic techniques to achieve the directional behavior of causal expressions.

Resumen

Este trabajo profundiza en el uso de causalidad en dominios de acciones y cambio, centrándose no sólo en su aplicación para la resolución de problemas representacionales, sino también en el interés en sí del conocimiento causal como una información significativa. Demostramos que la causalidad está íntimamente ligada al concepto de *pertinencia* con respecto a la ejecución de acciones, y proponemos una caracterización formal de esta idea, explicando su presencia en la semántica de las reglas causales. Esto nos permite la introducción de un lenguaje de alto nivel que proporciona una forma más cómoda de representar dominios de acciones utilizando construcciones causales. Proponemos además una visión unificada de las dos orientaciones existentes sobre causalidad, incorporándolas a dos niveles distintos: a un nivel ontológico, usamos pertinencia para capturar la idea de causalidad versus persistencia; mientras que a un nivel epistemológico, proponemos diferentes técnicas no monótonas para lograr el comportamiento direccional de las expresiones causales.

Contents

1 Introduction **1**

1.1 Causality in Reasoning about Actions and Change . . . 2
1.2 State of the art: two understandings of causality . . . 7
1.3 Main motivation: representing causal information . . . 8
1.4 Pertinence: a new focus on change . . . 11
1.5 Organization . . . 14

2 Nonmonotonic reasoning and logic programming **17**

2.1 Circumscription . . . 17
2.2 Default Logic . . . 19
2.3 Review of Logic programming . . . 22
2.3.1 Basic definitions . . . 22
2.3.2 Stable models . . . 23
2.3.3 Clark's completion and supported models . . . 25
2.3.4 Well Founded Semantics . . . 26
2.3.5 Cycles . . . 30
2.3.6 WFSX and the coherence problem . . . 31
2.3.7 A rewriting method for computing WFSX . . . 33

3 Static L^2 **37**

3.1 L^2 syntax and semantics . . . 37
3.2 Examples . . . 40
3.3 Extending L^2 syntax . . . 42
3.4 Encoding L^2 into classical logic . . . 44

4 Transition systems **47**

4.1 Narrative action domains . . . 47
4.2 Typical problems in action domains . . . 50
4.3 Automata and transition systems . . . 52
4.4 Adding Pertinence . . . 53

5 $\mathcal{P}$-language **59**

5.1 $\mathcal{P}$-rule syntax . . . 60
5.2 Operational semantics . . . 63
5.3 Examples . . . 65

6 Pertinence Calculus: formalizing pertinence in action domains 69
6.1 Basic definitions and axioms . . . 69
6.2 Propositional representation . . . 71
6.3 High level constructions . . . 72
6.4 Circumscribing pertinence . . . 75
6.4.1 Correspondence to operational semantics . . . 77
6.5 Encoding Pertinence into Default Logic . . . 81
6.6 Encoding pertinence into logic programming . . . 83
6.6.1 Correspondence to operational semantics . . . 85
6.6.2 Using program cycles for planning and temporal explanation . . . 86

7 Causal cycles 89
7.1 Characterizing the effects of causal cycles . . . 90
7.1.1 Positive vs. negative cycles . . . 90
7.1.2 Self-supportedness . . . 91
7.1.3 Nonexistence of successor state . . . 92
7.1.4 Nondeterminism . . . 93
7.1.5 Cumulativity and cycle ordering . . . 95
7.1.6 Coherence . . . 96
7.1.7 Summary . . . 97
7.2 Domains with cyclic dependences . . . 97

8 Comparison to other action approaches 103
8.1 Change as abnormality . . . 104
8.2 Lin's *caused* predicate . . . 104
8.3 Occlusion . . . 109
8.4 Event Calculus . . . 111
8.4.1 Original formulation . . . 112
8.4.2 Adapting the notation . . . 114
8.4.3 Relation to pertinence . . . 116
8.5 Inductive causation . . . 117
8.5.1 Initiation vs. pertinence of a formula . . . 119
8.6 Abnormality and Logic Programming . . . 121
8.7 Thielscher's approach . . . 122
8.7.1 Basic definitions and a pair of examples . . . 122
8.7.2 Comparison . . . 125
8.8 Causal explanation . . . 128
8.8.1 Literal completion. Relation to Logic Programming . . . 130
8.8.2 Pertinence postulates in Causal Explanation . . . 132
8.9 Schwind's comparative . . . 133

9 Pertinence Action Language 143
9.1 Missionaries and cannibals in PAL . . . 144
9.2 The blocks world . . . 147
9.3 Rule grounding . . . 149
9.4 Temporal projection . . . 152
9.5 Queries . . . 155

9.6 Queries for stable models: temporal explanation vs. planning 157

10 Conclusions 161

A Proofs of theorems 165

B PAL examples 175

B-1 Newton's formula 175
B-2 Lin's suitcase 176
B-3 Yale Shooting Problem 177
B-4 Lamp circuit 177
B-5 Combinatorial circuit 179
B-6 Account balance 179
B-7 The gong example 181
B-8 The alarm problem 182
B-9 The gear wheels 183
B-10 Shanahan's relay 185
B-11 The soup bowl 186
B-12 Thielscher's relay 187
B-13 The trapdoor 188

Chapter 1

Introduction

The concept of causality, frequently present in human daily discourse, has been traditionally absent from scientific formal reasoning. A well known example is the famous formula of Newtonian physics $f = m \cdot a$, expressing the relation among force, mass and acceleration. The formula is non-directional in the sense that it can be indistinctly used for computing any of the three magnitudes, knowing the values of the other two. However, we implicitly handle the intuition about a cause-effect direction – from force to acceleration – that allows us considering the computation of each magnitude as a *different* reasoning process:

- computing a is predicting the *effect* of the application of f to m;
- computing f is, somehow, planning which force must be applied to m in order to *cause* acceleration a, and finally,
- computing m is explaining why the application of f *has caused* the observed acceleration a.

In any case, we know, for instance, that obtaining m from f and a does not mean that we can vary the particle's mass by changing the force and the acceleration (at least, using Newtonian physics). All this information is behind the correct understanding of the formula, but not explicitly represented in it.

Closer to our concerns, something similar happens with the formalization of abstract reasoning about actions using classical logic. Consider, for instance, the following scenario, introduced in [65]:

> "A suitcase has two locks, and a spring-loaded mechanism that will *open* the suitcase when both locks are in the up position ($up(1)$ and $up(2)$)."

A possible representation of this domain would be the classical formula:

$$up(1) \wedge up(2) \supset open \tag{1.1}$$

This type of formula, known as *material implication*, does not reflect any causal knowledge either. In fact, it is logically equivalent to:

$$up(1) \wedge \neg open \quad \supset \quad \neg up(2) \tag{1.2}$$

and, as it happens with Newton's formula, it is possible to use it in different ways, such as:

$$\{\ (1.1),\ up(1),\ up(2)\ \}\ \vdash\ open$$
$$\{\ (1.1),\ up(1),\ \neg open\ \}\ \vdash\ \neg up(2)$$

although, again, we actually have different understandings of these applications. For instance, the first case, predicts the effect, *open*, of moving up both locks, whereas the second case explains that, if the suitcase is closed while the first lock is down, it is because the second lock is up. Notice the difference in the interpretation: $\neg up(2)$ *is not considered an effect* of $up(1)$ and $\neg open$.

Since in both examples we were reasoning about a snapshot of the world's configuration, causality has not become necessary for computing facts. However, it is clear that it acts behind our understanding of dynamic domains, and so, it will affect any attempt of formalizing our reasoning processes. Thus, it is no surprise that causality has become interesting for Artificial Intelligence (AI) [76], and particularly for the area of Reasoning about Actions and Change which, due to its goal of formalizing the common sense understanding of dynamic systems, it has experienced an increasing proliferation of causal proposals along its evolution.

1.1 Causality in Reasoning about Actions and Change

The appeal to causality has sporadically appeared throughout the evolution of research in actions and change in a more or less explicit way. Let us briefly outline the main research topics of this field in the order they appeared[1], paying special attention to the ones more related to causal reasoning.

The foundation of the area of reasoning about actions is usually identified with McCarthy's historical paper [73]. In this work, McCarthy established, as one of the goals of AI, the development of computer programs capable of performing common sense reasoning tasks like prediction, explanation or planning. To this aim, the domain knowledge was explicitly described using some kind of logical representation. Following this idea, McCarthy and Hayes introduced in [81] the *Situation Calculus*, a (many sorted) first order logic formalization which became the first and perhaps most widely used representation of action domains.

Although this dissertation does not use Situation Calculus, it is useful to outline it. Its notation identifies three basic sorts of logical objects: *fluents*, *actions* and *situations*. A fluent represents a property of the domain that may vary along time (for instance, the locks positions and the truth value of *open* in the previous example). An action is an intervention we may perform in order to alter the configuration of the domain (for example, toggling any of the locks). Finally, situations represent sequences of actions that are carried out in the domain. Syntactically, they are terms recursively constructed either with constant s_0 (the *initial* situation) or successive applications of function[2] $do(A,S)$, which denote the resulting situation of applying action A in S, a situation term in its turn. Using Situation Calculus, our formula (1.1) would be typically represented as:

$$\forall S, A.\ \big(holds(up(1),S) \wedge holds(up(2),S) \supset holds(open,S)\big) \tag{1.3}$$

[1]For a detailed revision, consult for instance [101] which has been recently reviewed with the comments in [79, 63, 99, 102].

[2]Other formalizations denote this function as $result(A,S)$. Besides, for homogeneity with the later use of logic programming, we follow the convention of upper case initials for variables and lower case initials for constants.

where A and S are variables for the sorts of actions and situations, respectively. Note the differences with respect to (1.1): we have introduced situation arguments to represent different instants along time. Besides, predicates $up(1), up(2)$ and *open* have been transformed into fluent objects[3] so that they become quantifiable now.

The original definition of Situation Calculus [81] already included *causal assertions*, which simply consisted in delays of an unspecified number of situations between the condition (or cause) and its resulting effect. However, causality was not the focus of the initial research in reasoning about actions. Instead, efforts were centered on solving the so-called *frame problem* which motivated great part of the research in the area and is still indirectly related to most subjects currently under study.

The frame problem (which was also first described in [81]) consists in the unfeasibility of representing explicitly the persistence of all the unaffected facts when an action is performed. Consider again the suitcase example and assume that we want to predict what happens with $up(1)$ in situation $do(\mathit{lift}(1), s_0)$ (we lift lock 1). Of course, we should include formulas that describe the effects of actions:

$$\forall S, N.\ holds(up(N), do(\mathit{lift}(N), S))$$

with N any lock number. However, it is easy to assume that our theory may contain more facts, $color\_of(suitcase, yellow)$, $at(suitcase, room)$, $dark(room)$, etc, that have nothing to do with toggling the locks. The list of unrelated facts may increase as we update our knowledge for a more accurate or complete representation of the domain. The problem is that, in order to avoid uncertainty for these facts, we must explicitly add formulas, called *frame axioms*, asserting what does not change. For instance, we must also add:

$$\begin{array}{rcl}
\forall S, N.\ \big(holds(dark(room), do(\mathit{lift}(N), S)) & \equiv & holds(dark(room), S)\big) \\
\forall S, N, L.\ \big(holds(at(suitcase, L), do(\mathit{lift}(N), S)) & \equiv & holds(at(suitcase, L), S)\big) \\
\forall S, N, C.\ \big(holds(color_of(suitcase, C), do(\mathit{lift}(N), S)) & \equiv & holds(color_of(suitcase, C), S)\big) \\
& \vdots &
\end{array}$$

for any location L and color C. Moreover, we must assert that moving one lock does not affect the other:

$$\begin{array}{rcl}
\forall S.\ \big(holds(up(1), do(\mathit{lift}(2), S)) & \equiv & holds(up(1), S)\big) \\
\forall S.\ \big(holds(up(2), do(\mathit{lift}(1), S)) & \equiv & holds(up(2), S)\big)
\end{array}$$

This means a serious problem of representation, since the mere addition of a new fluent may force us to reconsider the whole formulation of a given domain, *deciding which actions may affect the fact or not*, and usually, adding frame axioms for most of the existing actions. Besides, this task is even harder when we allow effects that depend on concurrent actions or that may be obtained indirectly, as we will see later.

The solution to the frame problem involves the addition of a default mechanism, called the common sense law of *inertia*, that can be enunciated as follows:

under no evidence on the contrary, fluents must remain unchanged.

[3]This transformation is usually called *reification*, which means "making into a *thing*."

This apparently simple criterion means a problem for classical logic reasoning, since we need to draw conclusions based on lack of evidence. This is not possible due to the monotonicity of the consequence relation of classical logic – the more formulas we add, the more consequences we obtain, that is: $Cn(T) \subseteq Cn(T \cup T')$. To see why inertia must be nonmonotonic assume, in our example, that we handle a theory T containing $holds(dark(room), s_0)$ (the room was dark). We would want to conclude by default that $holds(dark(room), do(lift(1), s_0)) \in Cn(T)$. However, if we are later told that the suitcase has an internal lamp which lights up the room:

$$\forall S.\ \big(holds(open, S) \supset \neg holds(dark(room), S)\big) \tag{1.4}$$

the previous conclusion must be clearly retracted:

$$holds(dark(room), do(lift(1), s_0)) \quad \notin \quad Cn(T \cup (1.4)).$$

The need for encoding the inertia law moved the attention to selecting an adequate nonmonotonic reasoning technique. Predicate *Circumscription* [74] and *Default Logic* [94] were the two solutions that received the most attention in the beginning. The first attempts relied on a *minimal change* orientation: select those classical models with less changes among situations. Technically, the solutions consisted in introducing a special predicate, *ab* (for *abnormal*), pointing out an exception to inertia. For instance, $ab(open, lift(1), S)$ points out that fluent *open* is free from inertia at situation S after performing action $lift(1)$. This can be expressed as:

$$\forall A, F, S.\ \big(\neg ab(F, A, S) \supset (holds(F, do(A, S)) \equiv holds(F, S))\big) \tag{1.5}$$

that rephrases the frame axioms in a more compact way, and so, it is usually called the *universal frame axiom*. Thanks to this axiom, the inertia default amounts to a models selection minimizing the extent of *ab*.

Unfortunately, a naïve application of the minimal abnormality solution may easily lead to counterintuitive results, as soon discovered by Hanks and McDermott with the Yale Shooting Problem (YSP) [48]. They proved, both in Circumscription and Default Logic, that the simple criterion of selecting the models with less abnormality extent was too weak, as it can be influenced by the position of the changes along the sequence of situations.

The Yale Shooting scenario consists in trying to kill a turkey by shooting a gun that must be previously loaded:

$$\forall S.\ holds(loaded, do(load, S)) \tag{1.6}$$
$$\forall S.\ \big(holds(loaded, S) \supset \neg holds(alive, do(shoot, S))\big) \tag{1.7}$$
$$\forall S.\ \big(holds(loaded, S) \supset \neg holds(loaded, do(shoot, S))\big) \tag{1.8}$$

To introduce some uncertainty, a waiting situation is inserted between loading and shooting:

$$\begin{aligned} s_1 &= do(load, s_0) \\ s_2 &= do(wait, s_1) \\ s_3 &= do(shoot, s_2) \end{aligned}$$

As a result, we get two models with minimal abnormality: the expected one in which the turkey results killed (the fluent *alive* becomes abnormal and false after shooting), and another one in which the the turkey remains surprisingly alive (the fluent *loaded* becomes abnormal and false after waiting). The problem is that, although both models have a minimal set of abnormalities, we know that the abnormal unloading should not be justified.

Notice the causal component of the YSP: the undesired model seems to arise because of making conclusions "against the tide:" we consider first the inertia of *alive* in the last situation and then we conclude backwards that the gun must have become unloaded. Moreover, in that inference process, the formula (1.5) is applied by contraposition: from a change of value in fluent *loaded*, we conclude that it is abnormal. This contrapositive application seems to violate our causal understanding of (1.5). In fact, we could have used the the equivalent formula:

$$\forall A, F, S.\ \big(\neg(holds(F,S) \equiv holds(F, do(A,S))) \supset ab(F,A,S)\big) \tag{1.9}$$

but something seems to point out that persistence should follow from non-abnormality and not vice versa. Furthermore, it seems that the authors had this directionality in mind, since they provided a different formulation of effect axioms like (1.7), being originally expressed as:

$$\forall S.\ \big(holds(loaded,S) \supset \neg holds(alive, do(shoot,S)) \wedge ab(alive, shoot, S)\big)$$

Although this change does not vary the final result, it somewhat emphasizes that the abnormality of *alive* should actually follow from the gun shot. A similar intuition has motivated a recent solution, due to Turner [109], which uses Default Logic, but making an appropriate use of inference rules to capture the right directionality. For instance, in our example, we could use the inference rules:

$$\frac{holds(loaded,S)}{\neg holds(alive, do(shoot,S)) \wedge ab(alive, shoot, S)}$$

$$\frac{holds(loaded,S)}{\neg holds(loaded, do(shoot,S)) \wedge ab(loaded, shoot, S)}$$

$$\frac{holds(F,S) \wedge \neg ab(F,A,S)}{holds(F, do(A,S))}$$

$$\frac{\neg holds(F,S) \wedge \neg ab(F,A,S)}{\neg holds(F, do(A,S))}$$

plus the minimal abnormality default:

$$\frac{:\neg ab(F,A,S)}{\neg ab(F,A,S)}$$

which in Default Logic means that *ab* should become false whenever it is consistent to assume so.

Despite of its clear relation to causality, the first solutions to the YSP were actually more concerned with its purely chronological aspect. *Chronological minimizations* [55, 52, 104] consisted in establishing a prioritized ordering, so that abnormalities were still minimized, but they had to appear as late as possible along time. In this way, the preferred model would be the one in which the turkey is killed. Apart from the complexity introduced in the minimization process, these solutions present a drawback in temporal explanation problems, since the explanations for a given observation are always placed in the last situations.

The second group of solutions became what we can call the "first wave" of approaches explicitly labeled as "causal." The so-called *causal minimizations* [56, 50] were inspired by the following idea: only those facts that are caused are allowed to change. To achieve this behavior, a predicate $causes(A, F, V)$ was defined, expressing that when action A is successfully executed, fluent F is caused to have value V. This predicate is minimized and combined with ab[4] so that any fluent is abnormal iff it has been caused to take one of its possible values. The solution to the YSP is then very simple – it amounts to include the assertions $causes(shoot, alive, false)$ and $causes(load, loaded, true)$. Since no assertion $causes(wait, loaded, V)$ is stated, *loaded* becomes not abnormal after *wait*, and the undesired model is ruled out.

The problem of these solutions is that their interpretation of causality just amounts to a simple table describing direct relations action/fluent-value. This over-simplified view disables a correct treatment of more complex shapes of causality like context dependent causation or indirect effects. These limitations, together with the fact that causality was not actually needed for solving the YSP[5] cooled down the initial interest on causality, which was not renewed until the study of the *ramification problem* came to scene.

The ramification problem, identified by Kautz in [52], consists in obtaining an appropriate description of the indirect effects of actions. We want to avoid describing *any* effect of an action as a direct effect. For instance, imagine that we have several different mechanisms for moving the locks. It is clear that this does not affect to the way in which we decide the suitcase status. However, if we force to describe everything as direct effects, we would have to repeat the specification on how *open* is affected by the lock positions *for each possible action that may move a lock*. At a first sight, the most direct way for describing these indirect effects is by adding formulas called *state constraints*, like (1.3), which must be satisfied in every situation. However, as we had explained in the beginning, a formula like this does not capture a directionality among the involved fluents and, in the presence of the inertia assumption (again the frame problem), it may lead to counterintuitive results. Assume we try to find out what happens in $do(lift(1), s_0)$ having:

$$\neg holds(up(1), s_0) \wedge holds(up(2), s_0) \wedge \neg holds(open, s_0)$$

A minimal change policy would result in two equally valid minimal models. One of them, as expected, considers that *open* changes to true, because $up(2)$ has persisted (formula (1.3) is applied from the antecedent to the consequent). However, a second minimal model considers instead that *open* persists false and (applying (1.3) by contraposition) obtains as *effect* that $up(2)$ becomes false, i.e., lock 2 is moved down!

Note that, despite of the similarity to the two minimal-abnormality models of the YSP, there is no chronological component now, since we only have one action execution (in fact, chronological minimization cannot be applied to solve this problem). Initial attempts, like for instance [57], consisted in classifying the fluents into different categories, so that the inertia of $up(2)$ would have more priority than the change in *open*. The problem for this technique is that, as explained in [107], any example can be easily extended with new layers of causal dependences so that we are forced to reclassify the set of fluents introducing new categories. Rather than a fluent classification, it seems that this prioritized reasoning should be automatically extracted from a suitable modification of state constraints: *causal* rules. This idea, in fact, motivated a more prolific "second wave" of causal solutions (just to cite some of them [69, 65, 47, 107, 28, 100, 103]) that has continued to the present.

[4]Predicate *ab* was actually called *Affects* in [56] and *Clipped* in [50].

[5]Apart from chronological minimizations, Baker also proposed [8] a different circumscription technique that relies on varying the interpretation of the *do* function.

1.2 State of the art: two understandings of causality

As we have said, during the last six years, a considerable amount of causal approaches have been proposed for simultaneously solving the frame and the ramification problems. Although all these solutions agree in introducing causal rules as expressions semantically different from state constraints (like, for instance, (1.3)), there is no agreement on what shape these causal rules should have. Some approaches rely on a classical logic representation, so that one difference between a causal rule and a constraint is that the former refers to some special-purpose predicate (similar to *abnormal* or *causes* seen before) apart from the one for expressing the fluent's truth value (usually, predicate *holds*). Other approaches use instead some type of (nonclassical) conditional logic to capture the directional behavior of causal rules, without including new predicates apart from *holds*. We will name these two different trends respectively as:

a) *change-based causality*

b) *inferential causality*

Let us describe more in detail each orientation.

a) Change-based causality

The first type of causal approaches interprets causality as something intimately related to the concept of change, that is, as the *counterpart of persistence*. As we said before, they define an additional predicate whose name can be *abnormal, occlude, caused, affects, clipped*, etc. Despite of the differences, all these predicates have a common purpose: they are used to represent that some fluent or fact has changed, becoming an exception to inertia. We will talk about the *change predicate* to refer to any of these special purpose predicates. Paradoxically, some of the approaches we will include in this category have not been traditionally classified as causal when, in fact, the technical implementation for all of them is very similar. It seems that classifying whether an approach is causal or not has been more influenced by the chosen name for the change predicate than by the formal semantics attached to it. This formal semantics presents two important features. First, causal rules are represented as classical material implications that involve references to the change predicate although, unfortunately, a different pattern is used in each approach. Second, nonmonotonicity is obtained as a result of applying some minimization policy for the change predicate. The main purpose of this method is to obtain as less exceptions to inertia as possible but, as a side effect, it must also avoid the contrapositive application of causal rules. The first formalization that solved the frame and ramification problems using a change predicate was proposed by Lin in [65]. Lin introduces a variation of predicate $causes(A, F, V)$, denoted $caused(F, V, S)$, standing for "fluent F is caused to have value V at situation S" (note how the action A has been replaced by a situation S). The most important part of Lin's approach is the minimization technique[6] applied:

i) minimize *caused* for the whole theory *excepting* the universal frame axiom,

ii) select those minimal models that satisfy the universal frame axiom.

[6]The general technique of minimizing only a part of the theory was, in fact, previously proposed by Sandewall [96] who called it *filtered preferential entailment*. Lin's proposal is a particular case in which the universal frame axiom is not included in the minimization.

In fact, this same technique was later followed by other approaches [47, 103].

b) Inferential causality

The second trend is based on the following idea: using a nonmonotonic formalism that allows general default reasoning (not only inertia) but simultaneously provides conditionals that behave as *inference rules.* A prototypical example of this philosophy would be the already mentioned Turner's use of Default Logic to solve the YSP. The roots of this trend come from the application of declarative logic programming to reasoning about actions. The interest of declarative logic programming, under the inferential understanding point of view, is that it provides both default reasoning (thanks to the availability of default negation) and directionality (by using program rules). Many proposals followed the methodology established in [41] in which, as a first step, a "high level" action language is defined (possibly involving causal expressions) and, afterwards, its semantics can be alternatively described in terms of a logic program (in that case, interpreted under the *answer sets* semantics). As examples of approaches following this methodology we could cite [10, 109, 61] (see [42] for an overview).

A similar line, which is closely related to the use of logic programming, relies on defining new conditional logics that capture the inferential behavior. The first relevant approach in this direction was [69], where inference rules were interpreted using a fully model-theoretic description, and inertia was presented as a simple fixpoint characterization of the successor state. Later, this approach was technically simplified, and generalized for capturing general default assumptions, giving raise to the so called *Causal Explanation Theories* [70, 60, 111]. The idea behind this approach is that any consequence of a causal theory must be causally explained. Thus, causal rules are seen as the description of those conditions that make the effects be directly caused. It must be noticed that there is an important difference between the understandings of "being caused" handled by change-based causality and by causal explanation. While, for instance, in Lin's approach, facts obtained by inertia *are not* caused, in causal explanation, inertia is represented as *one more causal rule* and so, its effects must also be caused. In fact, as stated by the so-called *principle of universal causation*: "everything must result caused."

These two main understandings of causality are not mutually excluding. For instance approaches like [107], [28] or even [12] have shown the possibility of a mixed orientation, combining a change predicate[7] with the application of inference rules or the use of logic programs. This suggests that it is possible to understand the inferential approach as a generic *lower level* option on which to represent the change-based formalization, which is more specific of action domains. This dissertation will follow, in fact, this organization, putting the stress in the shape of the change-based representation, but also providing different choices for the nonmonotonic inference to be used below.

1.3 Main motivation: representing causal information

Going back to the change-based approaches, it is surprising how causality is understood just as a mean for ruling out undesired models and not as a interesting phenomenon to be studied *per se.*

[7] In [107], it is not exactly a predicate, but a distinction between normal facts and *effects* (those that have been affected by the action).

As a result, many approaches are only concerned with the solution to the frame and ramification problems, paying little or no attention to the meaning of causality itself. For instance, it is usual that, after rejecting the undesired models, causal information is simply disregarded. In other words, there is no worry about the final extent of the change predicate and, of course, this makes difficult to provide an intuitive meaning for it.

This is exactly the point that gives rise to the *main motivation* of this thesis. Our purpose is to use the information of the change predicate as a significant part of the domain knowledge, acquiring the same level of importance as the fluent values or the performed actions. While in the earlier attempts, causal information is just used to describe a model selection policy, here it will become part of the ontology, being included inside interpretations. Of course, this will be done showing how to associate an intuitive meaning to this new information. This meaning will be related to the idea of *pertinence* or relevance of any formula with respect to the performed actions. To introduce these ideas, let us see a simple example.

Example 1 **(The lamp circuit)**
Consider the circuit in figure 1.1, introduced in [107], where a lamp lightbulb is on if and only if two switches are closed:

$$\forall S.\ \big(holds(light, S) \equiv holds(sw(1), S) \wedge holds(sw(2), S)\big) \tag{1.10}$$

□

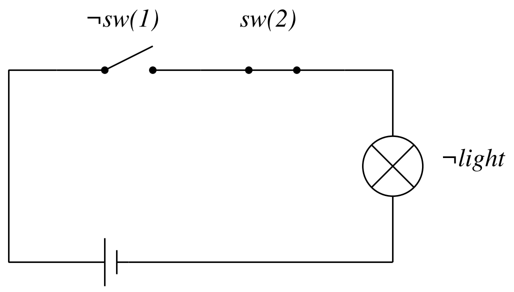


Figure 1.1: A simple circuit.

Assume that in some resulting situation s we have:

$$\neg holds(sw(1), s),\ holds(sw(2), s),\ \neg holds(light, s)$$

Our purpose is to represent not only the information described by this state (that is, the fluent values), but also how this information has been obtained. That is, which of these facts have been caused and which are due to inertia instead. Let us focus on fluent *light*, which has resulted false. Our commonsense intuition seems to point out that this false value is somehow "different" depending on the performed action and the previous state. Thus, we consider the four possibilities:

1. opening switch 1 while switch 2 was closed,

2. closing switch 2 while switch 1 was open,

3. opening switch 1 and closing switch 2 simultaneously,

4. or finally, performing no action on the switches.

For instance, we know that in case 1 *light* has been caused to be false, whereas in cases 2 and 4 it has remained unchanged. Case 3 is the most interesting one: the light has remained off, but depending on how simultaneous is the switches movement, it may have happened that the light experimented a momentary flash. A detailed description of a transient like that falls outside the scope of our representation[8]. However, it seems that it is the switching that *causes* the light to be off. Or, in different terms: we cannot guarantee that *light* has persisted false.

The example suggests that, in order to represent these ideas, we should incorporate into the state additional information about what is caused and what has persisted, *establishing a excluding distinction* between both possibilities. We could, for instance, define for each fluent F the new predicates:

$$\begin{array}{lcl} \mathit{caused}(F,V,S) & & (\text{with } V \in \{\mathbf{false},\mathbf{true}\}) \\ \mathit{persisted}(F,S) & \stackrel{\text{def}}{=} & \neg\mathit{caused}(F,\mathbf{true},S) \wedge \neg\mathit{caused}(F,\mathbf{false},S)\big) \end{array}$$

following Lin's notation [65]. However, predicate *holds* already represents the truth value V of the fluent. Thus, we can just express that the fluent is caused, regardless its truth value, introducing the notation: $\mathit{pert}(F,S)$. This points out that the truth value for F (which is separatedly specified by predicate *holds*) at S has been fixed by a (direct or indirect) causal intervention due to the performed actions. On the other hand, the negation, $\neg\mathit{pert}(F,S)$, allows us representing that the fluent F has followed inertia.

The use of the term "*pert*" comes from the idea of seeing the fact for *light* as relevant or *pertinent* with respect to the actions execution. The idea of associating causation to pertinence has been, in fact, inspired by a computational feature: only the facts for pertinent fluents need to be recomputed as we move from one situation to another. This feature is crucial when we try to efficiently compute the outcome of executing a sequence of actions in a large system, and furthermore, it allows displaying the results in a much more compact way (only pertinent facts need to be shown). As a matter of fact, this described displaying method is the one used by a real tool for designing temporal expert systems called *Medtool* [82] which has inspired great part of the intuitions for defining the idea of pertinence.

Back to our example, the next question is: how should we formulate the domain in order to obtain the right meaning for $\mathit{pert}(\mathit{light},s)$? A first, trivial, solution could be providing a direct

[8]Of course, some objections can be raised about which is the appropriated representation granularity to be used here. A full detailed description of the domain should perhaps include the physical delays between the switches movements and, whatnot, even the variations in the electrical current until a stable state is reached. This kind of representation would be the one handled by a physicist, whose interest is to describe the real world as accurately as possible, in order to improve predictions. However, in the area of reasoning about actions, we do not try to model the real world but, instead, the intelligent understanding of that world, and this usually means handling an abstraction (see for instance McCarthy's *approximate theories* [78]). In this way, anyone can understand and reason about the lightbulb circuit in example 1 without being aware of all the physical equations describing its behavior in detail. Furthermore, even inside Physics, we usually deal with simplified theories that allow ignoring low level details for concentrating on higher level problems. For instance, the design of combinatorial circuits using logical gates and Boole's algebra is but an abstraction of the electrical behavior of the different families of transistors and diodes. Similarly, in the lightbulb example, we are not interested in possible microscopical delays between the switches movements and the propagation of their effects. This is roughly "converted into" a simultaneous movement that results in a unique state. Then, causal information is used as a "footprint" of whether a fact may have been obtained via a chain of delayed changes or not.

description for each action execution:

$$\begin{array}{rcl}
\forall S.\ \big(holds(sw(2),S) & \supset & pert(light,do(open(1),S))\big) \\
\forall S.\ \big(\neg holds(sw(1),S) & \supset & \neg pert(light,do(close(2),S))\big) \\
\forall S. & & \neg pert(light,do(wait,S)) \\
\vdots & &
\end{array}$$

This solution clearly suffers from the frame and ramification problems, since we would always have to decide the formulas for *pert* in a nonmodular way, being forced to reformulate the axioms when new actions or indirect effects are specified. A second possibility could just be understanding *pert* as a change of truth value:

$$\forall A,S.\ \Big(pert(light,do(A,S)) \equiv \big(holds(light,S) \not\equiv holds(light,do(A,S))\big)\Big) \tag{1.11}$$

Unfortunately, formula (1.11) does not capture the desired behavior, since case 3 was an example of the light being pertinent without having experimented a change of value (at least, at a macroscopical level).

In fact, it seems that the truth for $pert(light,S)$ should be somehow related to the formula (1.10) and that this relation is in part what makes that formula to become a causal rule. The study on how to formalize this relation of pertinence and causal rules constitutes the main topic of this dissertation. But, before going into detail, it is perhaps more interesting to enunciate the basic properties of pertinence that our formalization should satisfy.

1.4 Pertinence: a new focus on change

The theoretical formalization of pertinence was introduced by Otero in [85], and most of its features were inspired by his previous work in a tool for the design and implementation of temporal expert systems called *Medtool* [82], and particularly by the inference mechanism he designed for that tool. This dissertation just deepens in the description of pertinence but paying special attention to its application for causal action domains. In this section, we summarize the expected meaning of pertinence by extrapolating our example into general postulates.

As we have seen, the central role of pertinence is to point out that a given fluent value has been obtained by application of a causal influence:

P1) *A fact will be* pertinent *whenever it is a direct or indirect effect of the performed actions, that is, whenever it is "affected" by them.*

As stated in P1, $pert(F,S)$ represents that F is affected by the currently performed actions, *whichever they are*. Note that we refer to some situation S, but not to any explicit action name argument. In this way, pertinence becomes an *inherent property* of the fluent, unlike other predicates used in the literature which may include additional arguments like the fluent value (as in $caused(F,V,S)$), the performed action (like in $ab(F,A,S)$) or even both (as in $causes(A,F,V)$).

A second interesting feature is that we will always understand $\neg pert(F,S)$ as a guarantee of real persistence. In this way:

P2) *Any effect will be obtained as a result of applying a causal chain or the result of inertia, but not both.*

Although this focusing seems to be present in most of the change-based approaches, we will see in Chapter 8 that, in many cases, there is not a clear excluding separation between applying causal rules and obtaining fluent values from inertia laws.

In this way, the only possibility for experimenting a change in a fluent value is due to pertinence, that is, due to the application of some causal rule:

P3) *A change of fluent value can only occur due to pertinence, i.e., due to an effect of a causal chain application.*

However, notice that the opposite does not hold, as we saw in case 3 of the lightbulb example:

P4) *Pertinence does not necessarily mean a change of value.*

The light could be off by pertinence (due to the switches movement) even though it was also previously off. Finally, in order to satisfy the previous postulates, we obtain the following one that establishes the role of pertinence inside a causal rule:

P5) *A causal rule implication must be applied if and only if its condition holds and it is pertinent.*

Actually, P5 is more a criterion than a real postulate: it is the only one that establishes a condition about the shape of causal rules. We have found that this condition is crucial for the fulfillment of the other four postulates. Moreover, in many change-based approaches, the nonsatisfaction of P5 leads to problems in the interpretation of the change predicate, when seen under the pertinence postulates. Assume we did not require pertinence of the rule condition, or in other words, that we allowed applying a rule whose condition has just persisted true. Then, the effect of the rule would become pertinent, (it is the result of a causal chain), but at the same time, there would not be any external intervention to justify its pertinence: the truth for the condition is obtained just by inertia. This violates postulate P1 – the effect would not be due to any action. On the other hand, assume that the rule condition requires *something else* apart from being true and pertinent. For instance, some approaches further require that the truth value of the condition has changed from false to true. This may easily lead to generate less pertinent effects than actually needed. Think again in case 3 of the lightbulb example: the rule condition $up(1) \wedge up(2)$ results false as in the previous state *but* we cannot guarantee that it has persisted false and so, the rule should be applied to obtain that *light* is pertinent.

The most relevant consequence of postulate P5 is that pertinence information will not just point out fluent values were were obtained but, what is more important, *it will affect rule conditions*, and so, the resulting state may vary depending on the pertinence information. Another very important consequence is that the idea of "pertinent" is applied here to a whole rule condition, not just to a single fluent: we can still naturally talk about the pertinence of $up(1) \wedge up(2)$ or its persistence. This means that pertinence becomes also a property of the formula, similar to its truth value. This extended application of pertinence can be intuitively explained as follows. When all the fluents involved in a formula ϕ persist, the valuation of ϕ will also be the same than in the previous situation. Analogously, when one of the fluents involved in ϕ is pertinent, we cannot guarantee the persistence of ϕ, and so, we say that *the formula* is pertinent.

In order to emphasize the importance of this last postulate, let us consider the following example.

Example 2 (Account balance)
We wish to maintain the balance of a checking account together with the amount of its last

transaction. To this aim, we handle the fluents $balance(X)$ and $transac(Y)$, where the latter points out the total amount of the last transaction (either positive or negative). We wish to recompute the balance only when a transaction is made. □

Of course, we are interested in representing both the current value of the last transaction (fluent *transac*) and the *balance* of all the transactions we were provided until now. Consider the rule for modifying *balance* depending on *transac*. Of course, the straightforward formulation would consist in referring to the actions used for establishing the transaction value, let us call them $deposit(Y)$ and $withdraw(Y)$, including formulas like[9]:

$$\forall S.\ holds(transac(Y), do(deposit(Y), S) \tag{1.12}$$
$$\forall S.\ holds(transac(-Y), do(withdraw(Y), S) \tag{1.13}$$
$$\forall S.\ \big(holds(balance(X), S) \supset holds(balance(X+Y), do(deposit(Y), S))\big) \tag{1.14}$$
$$\forall S.\ \big(holds(balance(X), S) \supset holds(balance(X-Y), do(withdraw(Y), S))\big) \tag{1.15}$$

However, this solution may easily present the ramification problem: it would not be so strange that the amount of the transaction was obtained as a result of a complex computation, rather than as a direct effect of an action. Furthermore, there might exist more different computations, caused by new actions, detailing the type of transaction we have performed. Thus, when $transac(Y)$ is not a direct effect, the above representation would force us to include an implication like (1.14) or (1.15) for each possible action that can modify *transac*, and so, the *balance* too.

To avoid this problem, we must represent the dependence between fluents *balance* and *transac*, without explicitly mentioning the performed action. A naïve attempt to formulate that relation could be the constraint:

$$\forall S, A.\ \big(holds(balance(X), S) \wedge holds(transac(Y), do(A, S)) \supset \\ holds(balance(X+Y), do(A, S))\big) \tag{1.16}$$

Unfortunately (and this is the key problem) this formula is not enough to differentiate between the case of two consecutive transactions for the same amount, and the case of simple persistence of the last *transac*, which could improperly modify the *balance*.

Postulate P5 claims that, when we use causal rules to model ramification dependences, we must require the rule condition not just to be true, but also pertinent. In other words, in order to modify the balance, *transac* must be pertinent (i.e. influenced by the latest action). As we will see in Chapter 8, some change-based causal approaches formulate the conditions of causal rules by excllusively referring to fluent values, thus violating postulate P5. As a result, undesired effects may follow without any external intervention, just as a result of inertia (for instance, an incorrect update of *balance* when *transac* actually persists).

A possible alternative is to formulate the causal rule so that it is applied when its condition *becomes* true (i.e., it is true, but was previously false). Remember that a change in truth value always implies pertinence (but not vice versa). In this way, we would be requiring a condition which is stronger than pertinence, and that it would work in most of the cases, but not all of them. In our example, the case of two consecutive transactions with the same amount would be

[9] We assume that additional axioms would be included in order to avoid simultaneous different values for fluents *transac* and *balance*.

understood as persistence (the rule condition maintains its truth value) and the *balance* would remain unchanged.

Finally, one more possible solution could be to define an auxiliary fluent, *new_transac* to point out that $transac(Y)$ must be taken into account for computing the *balance*. In this way, any rule that affected to $transac(Y)$ should also make *new_transac* true. On the other hand, the rule for *balance* should require the truth of *new_transac* (otherwise, the value must be disregarded, and these two fluents would persist). This is in fact the most modular solution, but coincides exactly with our description of pertinence: just rename $holds(new\_transac, S)$ as $pert(transac, S)$. Note how the presence of *new_transac* in the rule condition for *balance* is nothing else but the consequence of postulate P5.

If the chosen language already handles a predicate Π for representing that the fluent is caused (for instance $caused(transac, Y, S)$, $occluded(transac, S)$, $ab(transac, A, S)$ etc) it seems quite clear that this auxiliary fluent, *new_transac*, is not needed. Unfortunately, if the pertinence postulates are not satisfied, Π does not provide the required behavior and its utility just amounts to a purely technical application to enforce a model selection policy. We will come back to this example when doing the comparison with related work, in Chapter 8.

1.5 Organization

This dissertation is organized as follows. The next chapter is an overview of the basic technical definitions and known results about nonmonotonic reasoning that we will use in the following. We cover two of the best well known nonmonotonic techniques, Circumscription and Default Logic, but we also pay special attention to the use of declarative logic programming semantics as a practical tool for nonmonotonic reasoning. In Chapter 3, we make a first approach to the formalization of pertinence, restricting the analysis to a hypothetical successor state, where no additional dynamic information is still detailed. As a result, we obtain a logic, L^2, with two simultaneous valuations: the standard one for truth values plus a new one for pertinence.

The next chapter describes a typical narrative framework for reasoning about actions, particularly focusing on transition systems. We explain how the structure of a finite state machine (where pertinence becomes an output function) results appropriate for representing all answers to prediction, postdiction and planning problems, thus providing a complete visual representation of the domain behavior. Unfortunately, finite state machines present serious problems when used as a representational formalism due to their lack of modularity and, which is more important, to their impossibility for capturing the underlying causal relations in the domain (in fact, they only show the resulting effects).

To overcome these difficulties, Chapter 5 introduces a basic causal syntax, called $\mathcal{P}$-language (in the style of the $\mathcal{A}$-language). The $\mathcal{P}$-rules are interpreted under an operational semantics, that is, by applying an algorithm, which implements in practice the pertinence postulates, and allows computing all the possible transitions. In this way, we can generate the corresponding finite state machine starting from the causal representation. This method, however, is still far from constituting a real semantics, since it exclusively relies on an algorithmic description. Besides, its applicability will be intentionally restricted to acyclic causal dependences, since we will see how most formalizations exclusively differ in the interpretation of causal cycles.

The following chapters, 6 and 7, provide the logical formalization for the introduced basic causal language and its operational interpretation. Chapter 6 presents the basic monotonic framework we have called *Pertinence Calculus* (by analogy with Situation Calculus or Event

Calculus) plus the applications of different nonmonotonic techniques like circumscription, default logic, and logic programming (under three different declarative semantics). Essentially, pertinence calculus is an elaboration of L^2 that incorporates the needed ontology for our actions framework, defining multiple valued fluents, actions and situations. We prove correspondence theorems with respect to the operational semantics for all the nonmonotonic techniques studied before, under the assumption of acyclicity of causal dependences. Chapter 7 contains a study of causal cycles, comparing their behavior under all the alternatives.

Chapter 8 is a comparison of pertinence to several well-known action approaches and their treatment of causality. We show how most of them share many intuitions we defined for pertinence (specially change-based approaches), but in all the cases some of its postulates are not satisfied.

Next, Chapter 9 has a more practical orientation, describing an extension of $\mathcal{P}$-language, called *Pertinence Action Language* (PAL) that allows a more comfortable representation of action domains. We provide an overall description of this language, briefly commenting its syntax with several examples, and presenting some applications of the corresponding interpreter we have implemented. This interpreter works at two different levels: a *front-end* level for translating PAL expressions into $\mathcal{P}$-rules, and a *back-end* level for interpreting these rules under some of the nonmonotonic choices presented before.

Chapter 10 contains a final discussion, summarizing our results and outlining possible directions for future work.

For readability sake, we have included most of the proofs in Appendix A. Finally, Appendix B contains the representation in PAL of most of the examples presented in the thesis.

Chapter 2

Nonmonotonic reasoning and logic programming

This is an overview of well-known concepts of nonmonotonic reasoning and logic programming we will use along this work. An experienced reader should perhaps skip the whole chapter, excepting, at most, section 2.3.6, where we provide some contributions related to the bottom-up computation of well founded semantics with explicit negation. Detailed introductions to nonmonotonic reasoning can be found in [18, 36, 5] and, for the case of logic programming, in the surveys [66, 30, 6, 59, 31].

We have organized this background presentation in three sections. The first two sections describe the two general-purpose nonmonotonic approaches used in this work: *circumscription* and *default logic*. The last section is devoted to declarative logic programming and, more concretely, to three well known semantics for logic programs: *Clark's completion*, *stable models* and *well founded semantics*.

2.1 Circumscription

Circumscription [74] is one of the most popular nonmonotonic techniques and for which more variations have been proposed. Intuitively, given some classical theory T, we want to *circumscribe* the extent of some predicate p exclusively to those facts explicitly asserted in T.

Given two predicates p and P, we write $P \leq p$ and $P = p$ to respectively stand for:

$$\begin{aligned} P \leq p &\stackrel{\text{def}}{=} \forall X.\ (P(X) \supset p(X)) \\ P = p &\stackrel{\text{def}}{=} \forall X.\ (P(X) \equiv p(X)) \end{aligned}$$

being X a vector of variables. As usual, $p < P$ is an abbreviation of $p \leq P \wedge \neg(p = P)$. Then, we can provide the following definition:

Definition 1 (Circumscription) The *circumscription* of a theory T for a predicate p is denoted as $\mathrm{CIRC}[T;p]$ and defined as the second order formula:

$$\mathrm{CIRC}[T;p] \stackrel{\text{def}}{=} T(p) \wedge \neg\exists P.\ (T(P) \wedge P < p) \tag{2.1}$$

□

It must be noticed that circumscription *is not* a nonmonotonic logic, but a technique for obtaining nonmonotonic consequences of a classical theory. Note also that the definition of CIRC$[T;p]$ is using second order logic (we quantify over a predicate variable P). The intuitive meaning is clear: we fix the minimal extent for p among all the possible P's that satisfy the formulas in T.

In fact, as shown in [58] we may actually bear in mind the following alternative semantic characterization. Let M_1 and M_2 be two (first order logic) interpretations. We write $M_1 \leq^p M_2$ to express that M_1 and M_2 coincide in their universes and valuations for all constants and predicates, excepting the valuation for p, which satisfies: $M_1[p] \subseteq M_2[p]$. Then:

Property 1 *The models of* CIRC$[T;p]$ *are the* $\leq^p$*-minimal models of* T. □

In other words, circumscribing p in T corresponds to selecting those models of T that have minimal extent for p, while all the rest is fixed. The resulting effect of this minimization is, informally speaking, to introduce an ordering in the reasoning process, so that, we first freely decide the valuation of the fixed constants and, *afterwards*, we obtain a minimal extent for p. In this sense, there exists some sort of directional reasoning, although only for two levels, which may be insufficient sometimes.

The typical example for showing this insufficiency is the following one:

$$\begin{array}{r} \forall X.\ \big(\neg ab(X) \wedge german(X) \supset drinks\_beer(X)\big) \\ german(peter) \end{array}$$

Let us call T_1 to this theory which tries to represent that, by default, Germans drink beer, and that *peter* is german. The expected result of circumscribing predicate ab is that we should get $\neg ab(peter)$, and so, $drinks\_beer(peter)$. However, we may first freely propose $\neg drinks\_beer(peter)$ obtaining as a consequence $ab(peter)$, which actually constitutes a second model of CIRC$[T_1; ab]$. Notice that both models are not comparable using the $\leq^{ab}$ relation, since they do not coincide in the extent of *drinks_beer*, which is one of the fixed predicates.

To allow some flexibility, a modification called *variable circumscription* [75] was introduced.

Definition 2 (Variable circumscription) The *circumscription* of a predicate p in a theory T, varying the vector of predicates z is denoted as CIRC$[T;p;z]$ and defined as:

$$\mathrm{CIRC}[T;p;z] \stackrel{\mathrm{def}}{=} T(p,z) \wedge \neg\exists P, Z.\ (T(P,Z) \wedge P < p)$$

□

The corresponding ordering relation $M_1 \leq^{p;z} M_2$ used for the models minimization holds when:

1. $M_1[q] = M_2[q]$ for $q \neq p$ and q not in z,
2. $M_1[p] \subseteq M_2[p]$

that is, we allow now to vary the extent of predicates in z. Thus, for instance, our previous example can be simply solved by using the variable circumscription CIRC$[T_1; ab; drinks_beer]$. In this way, we are somehow proposing a directional reasoning organized in three levels:

1. decide freely the extent for $german(X)$,

2. minimize the extent of $ab(X)$,

3. and finally, obtain the consequences for $drinks\_beer(X)$ from the remaining models after the previous two steps.

Of course, this scheme is still rather rigid, since it just provides three levels. Although we will not actually use it, a natural generalization is the so called *prioritized circumscription* [54] in which we introduce a complete priority ordering for a chain of minimized predicates $p^1, \ldots, p^k$. As proved in [58], prioritized circumscription can be defined as:

$$\mathrm{CIRC}[T; p^1 > \cdots > p^k; z] \quad \equiv \quad \bigwedge_{i=1}^{k} \mathrm{CIRC}[T; p^i; p^{i+1}, \ldots, p^k, z]$$

Note that it would be possible to use this directional ordering to obtain a causal behavior for an stratified set of material implications, fixing the predicates to be included at each stratum or priority-level. However, this solution for implementing causal rules is not very appropriated because: (1) causal rules are not always stratified; (2) the ordering relation among predicates is not always fixed, but may dynamically vary; and (3) what is even more problematic, this ordering may be easily affected by simple elaborations like adding a new causal dependence.

One final interesting variation is the so-called *parallel circumscription* [75]. The idea is to allow the *simultaneous* minimization of several predicates.

Definition 3 (Parallel circumscription) The *parallel circumscription* of the vector of predicates $p = p^1, \ldots, p^k$ in the theory T, $\mathrm{CIRC}[T; p^1, \ldots, p^k;]$, is defined as the second order formula:

$$T(p) \wedge \neg \exists P.\ (T(P) \wedge P < p)$$

where P is a vector of predicate variables $P^1, \ldots, P^k$ and $P < p$ represents the formula:

$$P^1 \leq p^1 \wedge \cdots \wedge P^k \leq p^k \wedge \neg (P^1 = p^1 \wedge \cdots \wedge P^k = p^k)$$

□

This is sometimes particularly interesting when moving from a first order theory T with a finite domain universe to a propositional theory T', where all the quantifiers and variables are replaced by (finite) ground conjunctions or disjunctions. In such a case, the original circumscription $\mathrm{CIRC}[T; p]$ actually corresponds to the propositional parallel circumscription $\mathrm{CIRC}[T'; p(a_1), \ldots, p(a_n)]$, where $p(a_i)$ are all the ground instances of predicate p.

2.2 Default Logic

Default logic, conceived by Ray Reiter [94], is perhaps one of the most widely used nonmonotonic formalisms. As discussed in [62], some of the aspects that seem to have contributed to its success are:

1. its original definition has proved to be expressive enough to cope with most of the research topics, without excessive need for additional improvements or variations,

2. as shown in [109], its use of inference rules allows avoiding the frame, the Yale Shooting and the ramification problems,

3. and last, but not least, declarative logic programming (particularly, the *stable models* [38] semantics) can be actually used as a practical implementation of (a subset of) default logic.

This last feature has been frequently used as a powerful connection between the areas of nonmonotonic reasoning and logic programming, so that results in one area can be extrapolated to the other. In fact, we will actually follow the presentation of default logic introduced in [43], due its similarity with the description of stable models.

The intuitive idea behind default logic is to extend the use of classical inference rules so that part of their antecedent is not required to be present among the current consequences, but just to be consistent with them. A *default rule* is a structure of the form:

$$\frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma}$$

where α, γ and the β_i's are ground formulas. Although rules with variables are usually allowed, they are actually understood as rule patterns, replacing each variable by all its possible ground instances. Under default logic terminology, α is called the *prerequisite*, each β_i is a *justification*, whereas γ receives the name of *consequent*. When $\alpha = \top$, we simply write:

$$\frac{: \beta_1, \ldots, \beta_m}{\gamma}$$

Furthermore, if the default has the shape:

$$\frac{: \gamma}{\gamma}$$

it is said to be *normal*.

As said before, the informal behavior of a default rule is that when α has been obtained and all the β_i are consistent (i.e., their negations are not among the current consequences), we add the new consequence γ. The problem for formalizing a process like this is that, if we simply apply a transitive closure of rule applications, some of the justifications β_i we had assumed as consistent may finally become false, invalidating the whole process. In other words, unlike the classical inference closure, default reasoning cannot be captured by a constructive process.

For a formal description, let us begin first with usual (nondefault) inference rules. A (propositional) *inference rule* is a structure of the shape:

$$\frac{\alpha}{\gamma} \tag{2.2}$$

Definition 4 (Closed theory) A theory E is said to be *closed* with respect to a set of inference rules D iff for any rule (2.2) in D such that $\alpha \in E$ then $\gamma \in E$. □

In short, E is closed w.r.t. D iff no more rules can be applied to increase E.

Inference rules are typically used in classical logic proof theory. Notice that they behave as "blind" syntactic transformations. For instance, a typical logical inference rule is:

$$\frac{\alpha \wedge \beta}{\beta \wedge \alpha} \tag{2.3}$$

to guarantee commutativity of conjunction. Besides, rules like this are typically specified as general patterns, using meta-variables (like α and β above) to represent any propositional formula.

We will call *logical rules* to the usual set of inference rule patterns for propositional calculus. We say that E is *logically closed* when it is closed with respect to the logical rules, i.e., E includes itself all its possible logical consequences. The *logical closure* of a theory E, denoted as $Cn(E)$, is the least superset of E that is logically closed.

Assume that we handle now a set D of *nonlogical* rules. Our interest is to get the consequences of some initial theory W under the set of rules in D. For simplicity sake, we may assume that $W \subseteq D$ and that any classical formula ϕ actually represents the nonlogical rule:

$$\frac{\top}{\phi} \tag{2.4}$$

Thus, we simply define the set of *consequences* of D, $Cn(D)$, as the closure of the empty theory $\emptyset$ with respect to both logical rules and rules in D. In other words, $Cn(D)$ is the least set of formulas that are both logically closed and closed with respect to D. Notice that, by abuse of notation, we have used the same name, 'Cn', than for logical consequences. However, this does not lead to ambiguity, since, when D only contains classical formulas, like (2.4), $Cn(D)$ corresponds to the logical closure, as before.

The theory $Cn(D)$ can be obtained incrementally by successive applications of rules. To this purpose, we define the operator of *direct consequences*, T_D, that maps each logically closed theory E into the new logically closed theory:

$$T_D(E) = Cn\left(\left\{ \gamma \;\middle|\; \frac{\alpha}{\gamma} \in D \;\; \text{and } \alpha \in E \right\} \right)$$

Successive applications of this operator, starting with $E_0 = Cn(\emptyset)$ (i.e., the set of all the tautologies), will always lead to a fixpoint (no more rules are applicable), which is the final set of consequences:

$$T_D(E_i) = E_i = Cn(D)$$

Let us consider now a set D of *default* rules, that is, inference rules with justifications. As explained before, the justifications do not need to be among the consequences – we just require that they are not finally denied by them. The problem, of course, is that during the inference process we cannot predict whether the assumptions will be later denied or not, and so, simply iterating some variation of T_D will not be enough now. Instead, we proceed as follows: we first choose an arbitrary logically closed theory E for valuating the consistence of justifications in all the rules; afterwards, if the set of resulting consequences happens to coincide with E, then E is a possible "valid" assumption and is called an *extension* of D.

Formally, given a set of default rules D and a logically closed theory E, we define the set of inference rules D^E (read D *modulo* E) as:

$$D^E = \left\{ \frac{\alpha}{\gamma} \;\middle|\; \frac{\alpha : \beta_1, \ldots, \beta_m}{\gamma} \in D, \text{ and there is no } \neg\beta_j \in E \right\}$$

That is, D^E contains each rule where the justifications are consistent w.r.t E, and so, they are omitted. Since the resulting D^E consists of usual inference rules, we can compute its consequences $Cn(D^E)$ as before. Thus, extensions are characterized by the following fixpoint condition:

Definition 5 (Extension) A logically closed theory E is an *extension* of a set of default rules D iff $E = Cn(D^E)$. □

A default theory may have several extensions or no extension at all. For instance, it can be easily seen that D_1:

$$\frac{:\neg p}{q} \qquad \frac{:\neg q}{p}$$

has two extensions $E_1 = Cn(\{p\})$ and $E_2 = Cn(\{q\})$, whereas D_2:

$$\frac{:\neg p}{p}$$

has no extension. Notice also that having no extension is different from having an inconsistent one. For example, the default theory D_3:

$$\frac{\top}{p \wedge \neg p}$$

has as unique extension the inconsistent theory $Cn(\{\bot\})$, i.e., the whole set of well formed formulas.

A default theory D is said to be *consistent* iff it has at least one consistent extension. We usually say that a formula is a *consequence* of D when it belongs to all its extensions.

2.3 Review of Logic programming

In [16] an important connection was discovered: if we momentarily forget the control strategy of Prolog, we may essentially understand that a program with negation as failure like, let us say:

```
p :- \+ q.
r :- p, \+ s.
```

tries to solve the same problem as the default theory:

$$\frac{:\neg q}{p} \qquad \frac{p:\neg s}{r}$$

This relation opened the cross research between the fields of nonmonotonic reasoning and logic programming. On the one hand, nonmonotonic approaches are used to study declarative semantics for logic programming and its generalizations. On the other hand, logic programs provide an excellent tool for obtaining practical applications of nonmonotonic reasoning.

We study in this section three well known logic programming semantics, beginning with the *stable models* because of its closeness to default logic.

2.3.1 Basic definitions

The syntax of logic programs is defined starting from a finite set of ground atoms $\mathcal{H}$ called the *Herbrand base*. We assume that all the variables have been previously replaced by their possible ground instances. A *program literal* is either an atom $a \in \mathcal{H}$ or its default negation *not* a (which is called a *default literal*). A *normal logic program* is a finite set of *rules* of shape:

$$H \quad \leftarrow \quad L_1, \ldots, L_n$$

where $n \geq 0$, H is an atom called the *head* of the rule, and the $L_i's$ are program literals which receive the name of *body* of the rule. When $n = 0$, we usually write 'H' to stand for '$H \leftarrow$', and say that H is a program fact. A normal logic program is said to be *positive* (or *definite*) when it does not contain any default literal. Note that, if we interprete ',' and '$\leftarrow$' as classical conjunction and implication, then a positive logic program just corresponds to a set of Horn clauses.

Some syntactic restrictions on the presence of cycles will be later interesting. Given a normal logic program P we define its corresponding graph $G(P)$ as follows. Each node corresponds to an atom in the Herbrand base $a \in \mathcal{H}$, whereas each arc (a, b) denotes that atom b occurs in the body of a rule with head a. If b occurs as an atom, the arc is said to be *positive* and labeled with a '+', whereas if b occurs in a default literal, *not* b, the arc is *negative* and labeled with a '-'. A *cycle* is any path in $G(P)$ from a to a. The cycle is positive iff all the arcs involved in it are positive. Otherwise, the cycle is said to be negative. When $G(P)$ does not contain cycles, P is said to be a *hierarchical* program, whereas when it does not contain negative cycles, it is said to be *stratified*. Another way of defining a hierarchical program P is by requiring the existence of a (nonnegative) integer mapping:

$$level : \mathcal{H} \rightarrow \mathbb{N} \cup \{0\}$$

so that, for each rule with a as head, $level(a) < level(b)$ for any atom b occurring in the body of the rule.

For defining the different semantics of logic programs we will use the following intuitions. We will handle propositional models which more or less will play the same role than extensions in default logic, but restricted now to sets of atoms. In this way, we lose the use of classical disjunction and negation. As a result, some care will be needed with the meaning of falsity under this new focusing. For instance, given a model M, if $a \notin M$ for some atom a, we understand in logic programming that a is false, but we would actually understand that a is unknown, when seeing M as an extension in default logic.

A (2-valued) *interpretation* M is defined as any subset of $\mathcal{H}$, $M \subseteq \mathcal{H}$. It can also be seen as a function $M : \mathcal{H} \rightarrow \{\mathtt{t}, \mathtt{f}\}$ mapping a truth value for each atom in $\mathcal{H}$, so that $M(a) = \mathtt{t}$ iff $a \in M$. The interpretation can be extended to provide valuation of any formula ϕ, so that $M(\phi)$ follows the standard propositional definitions, where '*not* ', ',' and '$\leftarrow$' represent classical negation, conjunction and material implication, respectively. When $M(\phi) = \mathtt{t}$ we usually say that M *satisfies* ϕ and write $M \models \phi$. An interpretation is said to be a *model* of a program P iff it satisfies all its rules.

2.3.2 Stable models

The definition of the *stable models* [38] semantics naturally arises from the already explained connection between default logic and logic programming. Beginning with positive programs, it can be easily observed that each program rule like

$$c \leftarrow a_1, \ldots, a_n$$

with $\{c, a_1, \ldots, a_n\} \subseteq \mathcal{H}$ can just be seen as the inference rule:

$$\frac{a_1 \wedge \cdots \wedge a_n}{c}$$

Thus, in order to compute the consequences of a positive program P, we may follow similar steps as for inference rules. Since we exclusively deal with atoms, the definition of "being closed" with respect to P simply amounts now to require $M \models P$. From this, we obtain that $Cn(P)$, that is, the least theory closed with respect to P, becomes now the *least model* of P, let us denote it $LM(P)$. Finally, we may also use T_P operator[1] iteratively:

$$T_P(M) = \{c \mid (c \leftarrow a_1, \ldots, a_n) \in P, \text{ and } a_i \in M \text{ for all } i \in [1,n]\}$$

to compute $LM(P)$.

Property 2 *Given a positive logic program P, the least fixpoint of T_P is the least model of P: $LM(P)$.* □

In order to define the semantics for default negation, we provide a definition of program modulo in a similar way as we did for default theories. Given a normal logic program P and an interpretation M, we define the positive program P^M (read *P modulo M*) as:

$$P^M = \{c \leftarrow a_1, \ldots, a_n \mid (c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m) \in P, \text{ and there is no } b_j \in M\}$$

Since P^M is a positive program, it has a least model $LM(P^M)$. We will usually write $\Gamma_P(M) \stackrel{\text{def}}{=} LM(P^M)$ or simply $\Gamma(M)$ when there is no ambiguity.

Definition 6 (Stable model) An interpretation M is a *stable model* of a normal logic program P iff M is a fixpoint of Γ, that is, $M = \Gamma(M)$. □

Of course, it can be easily seen that any stable model M is a model of the program. Note that the only difference w.r.t. default logic is that literals $not\ b_j$ represent here the atoms that *must not belong* to M, whereas justifications β_j represented there the formulas for which $\neg\beta_j$ should not belong to E. In this way, the program rule:

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

actually corresponds to the default:

$$\frac{a_1, \ldots, a_n : \neg b_1, \ldots, \neg b_m}{c}$$

Following this correspondence, assume that $D(P)$ denotes the representation of a program P as a default theory. Then, the following result [39] applies:

Property 3 *Given a normal logic program P, M is a stable model of P iff $Cn(M)$ is an extension of $D(P)$, seeing M as a theory consisting of atomic formulas.* □

Exactly as in default logic, a logic program may have several stable models, like in the program:

$$\begin{array}{rcl} p & \leftarrow & not\ q \\ q & \leftarrow & not\ p \end{array}$$

with the stable models $\{p\}$ and $\{q\}$, or no stable model at all, like in:

$$p \leftarrow not\ p$$

However, we will see later that these possibilities only arise in presence of cycles. Hierarchical programs always have a unique stable model (which, in fact, coincides with the result of the other two semantics).

[1]In fact, this is the original T_P operator introduced by van Emdem and Kowalski in their historical paper [112].

2.3.3 Clark's completion and supported models

The idea of *program completion* was first introduced by Clark [25]. To understand its motivation, assume we have the positive logic program P_1:

$$\begin{array}{lcl} p & \leftarrow & q, r \\ p & \leftarrow & s \\ r & & \end{array}$$

As a propositional theory, $\{q \wedge r \supset p,\ s \supset p,\ r\}$, this program has many models which, for instance, allow varying the truth of p. However, when working with a logic program we actually handle the following intuition: the two rules for p are the *only* possibilities to derive p. So, rather, than a single implication, we should handle $p \equiv (q \wedge r) \vee (s)$.

The *completion* of a normal logic program P, COMP$[P]$, is defined as the classical propositional theory:

$$\text{COMP}[P] \stackrel{\text{def}}{=} \{p \equiv B_1 \vee \cdots \vee B_n \mid p \in \mathcal{H},\ \text{and the } B_i \text{ are all the bodies for head } p\}$$

following some conventions:

1. an empty body is considered as the constant $\mathtt{t}$ (true),
2. an empty disjunction corresponds to the constant $\mathtt{f}$ (false),
3. the commas and the default negations are respectively seen as classical conjunctions '$\wedge$' and negations '$\neg$'.

The theory COMP$[P_1]$ would correspond to:

$$\begin{array}{lcl} p & \equiv & (q \wedge r) \vee s \\ q & \equiv & \mathtt{f} \\ r & \equiv & \mathtt{t} \\ s & \equiv & \mathtt{f} \end{array}$$

which clearly has the unique model $\{r\}$, which in this case, coincides with the least model $LM(P_1)$. However, this coincidence is not general, even for positive programs. The simple program P_2:

$$p \leftarrow p$$

would lead to the completion $p \equiv p$ which has two possible models, $\emptyset$ and $\{p\}$, being $LM(P_2) = \emptyset$. As we can see, the completion of a positive program may lead to more than one model.

To see an example of completion for a nonpositive program, consider P_3:

$$\begin{array}{lcl} a & \leftarrow & \mathit{not}\ b, \mathit{not}\ c \\ a & \leftarrow & d \\ c & \leftarrow & d \\ d & \leftarrow & c \end{array}$$

and its completion COMP$[P_3]$:

$$\begin{array}{rcl} a & \equiv & (\neg b \wedge \neg c) \vee d \\ b & \equiv & \mathtt{f} \\ c & \equiv & d \\ d & \equiv & c \end{array}$$

which has two models: $\{a, c, d\}$ and $\{a\}$.

Although the completion of a program P is a syntactic transformation, there also exists a semantic characterization of the models of COMP$[P]$.

Property 4 *Let P be a normal logic program and M a 2-valued interpretation. Then $M \models$ COMP$[P]$ iff $M = T_P(M)$.* □

That is, the models of completion are all the fixpoints of operator T_P, which receive the name of *supported models*. This means, for instance, that when the program is positive, completion is a weaker semantics than the least model (which selects the *least* fixpoint of T_P).

2.3.4 Well Founded Semantics

The main idea for introducing *well founded semantics* [113] (WFS) is to allow interpretations in which some atoms are not defined (neither true nor false). The motivation for this is that cycles like $p \leftarrow \mathit{not}\ p$, which may cause the nonexistence of stable model, may have a well founded model where p is left undefined, without affecting other atoms not actually involved in the cycle. We will present three alternative definitions of WFS, beginning with the most semantic one, which was introduced by Przymusinski [92], who used a direct generalization of stable models for the three-valued case.

We define a 3-valued or *partial interpretation* M, as a pair (M^+, M^-) of disjoint sets of atoms, so that the sets M^+, M^- and $\mathcal{H} - (M^+ \cup M^-)$ represent the *true*, *false* and *undefined* atoms, respectively. Given M^-, it will be sometimes convenient to refer to the complementary set $\mathcal{H} - M^-$, that is, the nonnegative atoms. It is clear that, since M^+ and M^- are disjoint, $M^+ \subseteq \mathcal{H} - M^-$. We say that M is *complete* iff $M^+ = \mathcal{H} - M^-$, i.e., $\mathcal{H} = M^+ \cup M^-$. Note that we may characterize any complete M only by its set of true atoms, M^+, which can be understood as a 2-valued interpretation.

Valuation of formulas assigns now to each formula one of the three possible values $\{\mathtt{f}, \mathtt{u}, \mathtt{t}\}$ (where $\mathtt{u}$ stands for *undefined*) following the rules:

$$1)\ \ M(p) = \begin{cases} \mathtt{t} & \text{if } p \in M^+ \\ \mathtt{f} & \text{if } p \in M^- \\ \mathtt{u} & \text{otherwise} \end{cases}$$

$$2)\ \ M(\mathit{not}\ \phi) = \begin{cases} \mathtt{t} & \text{if } M(\phi) = \mathtt{f} \\ \mathtt{f} & \text{if } M(\phi) = \mathtt{t} \\ \mathtt{u} & \text{otherwise} \end{cases}$$

$$3)\ \ M(\ (\phi, \psi)\ ) = \mathit{min}(M(\phi), M(\psi))$$

$$4)\ \ M(\phi \leftarrow \psi) = \begin{cases} \mathtt{t} & \text{if } M(\psi) \leq M(\phi) \\ \mathtt{f} & \text{otherwise} \end{cases}$$

5) $M(\mathtt{t}) = \mathtt{t}$, $M(\mathtt{f}) = \mathtt{f}$, $M(\mathtt{u}) = \mathtt{u}$

Again, we say that a partial interpretation M *satisfies* a formula ϕ, written $M \models_3 \phi$, when $M(\phi) = \mathtt{t}$ and that M is a *partial model* of a program P iff it satisfies all its rules.

We will use two ordering relations for comparing partial interpretations.

Definition 7 (Truth ordering, $\leq$) We say that interpretation $M_1 = (M_1^+, M_1^-)$ assigns *less truth* than interpretation $M_2 = (M_2^+, M_2^-)$, denoted as $M_1 \leq M_2$, iff $M_1^+ \subseteq M_2^+$ and $M_2^- \subseteq M_1^-$. □

Definition 8 (Information or Fitting's ordering, $\leq_F$) We say that interpretation $M_1 = (M_1^+, M_1^-)$ contains *less information* than interpretation $M_2 = (M_2^+, M_2^-)$, denoted as $M_1 \leq_F M_2$, iff $M_1^+ \subseteq M_2^+$ and $M_1^- \subseteq M_2^-$. □

Note that, if we extend the truth ordering to truth values $\mathtt{f} \leq \mathtt{u} \leq t$, then for any expression ϕ, $M_1(\phi) \leq M_2(\phi)$ iff $M_1 \leq M_2$. When we handle complete interpretations, $M_1 \leq M_2$ simply amounts to $M_1^+ \subseteq M_2^+$, whereas all the complete interpretations are $\leq_F$-maximal (they cannot contain more information).

As happened with 2-valued interpretations, any positive program has also a unique least partial model.

Property 5 *Let P be a positive program. There exists a unique $\leq$-minimal model of P, called its* Least Partial Model, *$LPM(P)$.* □

In fact, when the positive program does not contain truth constants in its body, the least partial model is complete and coincides with $LM(P)$. However, if we allow truth constant $\mathtt{u}$ to be one of the body atoms, then the $LPM(P)$ may be not complete. For example, the program P_4:

$$
\begin{array}{lll}
p & & \\
q & \leftarrow & p \\
r & \leftarrow & p, \mathtt{u} \\
s & \leftarrow & s
\end{array}
$$

has the following models:

	M^+	M^-
M_0	$\{p, q\}$	$\emptyset$
M_1	$\{p, q\}$	$\{s\}$
M_2	$\{p, q, s\}$	$\emptyset$
M_3	$\{p, q, r\}$	$\{s\}$
M_4	$\{p, q, r, s\}$	$\emptyset$
M_5	$\{p, q, r\}$	$\emptyset$

To select the $\leq$-minimal models, we take as less positive atoms and as much negative ones as possible, and so $LPM(P_4) = M_1$, which makes p and q true, s false, and r undefined.

As the natural following step, we extend the modulo operation to the 3-valued case. When M is a partial interpretation, P^M is defined by replacing in P all the default literals *not* p by $M(\textit{not } p)$. Note that we can simplify the program by deleting the rule, when $M(\textit{not } p) = \mathtt{f}$, and deleting the default literal from the body, when $M(\textit{not } p) = \mathtt{t}$. The resulting program is positive (free from default literals) and so, it has a least partial model $LPM(P^M)$.

Definition 9 (Partial Stable Model) A partial interpretation M is said to be a *partial stable model* of a program P iff $M = LPM(P^M)$. □

In this way, we can see any 2-valued stable model M as the partial stable model $(M, \mathcal{H} - M)$ which happens to be complete. The interest of partial stable models is that, for any normal logic program, there always exists a particular one with the least amount of information.

Property 6 *Given any normal logic program P, there exists a unique $\leq_F$-minimal partial stable model of P, which receives the name of* Well Founded Model *(WFM).* □

Notice that this immediately means that:

Corollary 1 *Let W be the well founded model of a normal logic program P. For any 2-valued stable model M of P, we have that $W \leq_F (M, \mathcal{H} - M)$.* □

Corollary 2 *Let $W = (W^+, W^-)$ be the well founded model of a normal logic program P. If W is complete, i.e. $W^+ = \mathcal{H} - W^-$, then W^+ is the unique 2-valued stable model of P.* □

Notice that a complete WFM means a unique stable model, but not the opposite. For example, the program P_5:

$$\begin{array}{rcl} b & \leftarrow & \mathit{not}\ c \\ c & \leftarrow & \mathit{not}\ b \\ d & \leftarrow & \mathit{not}\ c \\ d & \leftarrow & \mathit{not}\ d, \mathit{not}\ b \end{array}$$

has a unique stable model, $\{b, d\}$, but its WFM, $(\emptyset, \emptyset)$, is not complete (all the atoms are undefined).

At a first glimpse, computing the WFM involves more complexity than the 2-valued stable models. Potentially, there exist more possible partial stable models than complete ones. Besides, after computing them, we must make an additional minimization (computing the $\leq_F$ least partial stable model). However, due to the shape of logic programs, the WFM can be computed using an *incremental* procedure, and so, it is actually simpler than computing the stable models.

The second characterization of WFS we will use actually corresponds to the most usual method for computing the WFM. This method relies on the Γ operator we had defined for stable models which is antimonotonic with respect to inclusion among sets of atoms. As a result, Γ^2, that is, Γ applied twice, results to be monotonic, and so, the Knaster-Tarski's theorem [106] is applicable: there exists a least (resp. greatest) fixpoint, $lfp(\Gamma^2)$ (resp. $gfp(\Gamma^2)$), which is computable by iteration on the least set of atoms $\emptyset$ (resp. the greatest set of atoms $\mathcal{H}$). The interest of Γ^2 is clarified by the following result:

Property 7 *The well founded model of a normal logic program P corresponds to the 3-valued interpretation $(lfp(\Gamma^2), \mathcal{H} - gfp(\Gamma^2))$. Moreover, each fixpoint of this pair can be computed in terms of the other: $gfp(\Gamma^2) = \Gamma(lfp(\Gamma^2))$ and $lfp(\Gamma^2) = \Gamma(gfp(\Gamma^2))$.* □

Finally, we will also present a third method of computing the WFM which is the most procedural, but also the the most efficient one. This method was developed by Brass and Dix (for a recent update, see [17]) and relies on successively applying the following intuitive program transformations:

1. *Failure* $\stackrel{\mathtt{F}}{\longmapsto}$: delete all rules containing the positive literal p such that p is not head of any rule.

2. *Positive reduction* $\stackrel{\mathtt{P}}{\longmapsto}$: delete all literals *not* p with p not head of any rule.

3. *Success* $\stackrel{\mathtt{S}}{\longmapsto}$: delete any positive literal p such that p is a fact.

4. *Negative reduction* $\stackrel{\mathtt{N}}{\longmapsto}$: delete all rules containing *not* p such that p is a fact.

5. *Positive loop detection* $\stackrel{\mathtt{L}}{\longmapsto}$: delete all rules containing the positive literal p such that $p \notin \Gamma(\emptyset)$

Property 8 *(see theorem 4.17 in [17]) The transformations* $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$ *are sound w.r.t. WFS and provide a confluent calculus which is strongly terminating. Furthermore, if* P *is the final program (where no new transformation is applicable) then the WFM,* $W = (W^+, W^-)$*, satisfies:*

$$\begin{array}{rcl} W^+ & = & facts(P) \\ W^- & = & \mathcal{H} - heads(P) \end{array}$$

where $facts(P)$ *(resp.* $heads(P)$*) denotes the set of atoms that occur as a program fact (resp. as a head) in* P*.* □

The first four transformations allow simplifying the program by ruling out those atoms with trivial or direct interpretation. For instance, when an atom p is one of the program facts, we can be sure that it will be finally true, and so we can replace any program literal containing p by its final truth value (transformations $\stackrel{\mathtt{S}}{\longmapsto}$ and $\stackrel{\mathtt{N}}{\longmapsto}$). Analogously, when an atom is not head of any rule, it will be finally false (there is no way to obtain evidence for it) and we can also simplify the corresponding program literals, using transformations $\stackrel{\mathtt{F}}{\longmapsto}$ and $\stackrel{\mathtt{P}}{\longmapsto}$. As shown in [17] (theorem 4.9), the exhaustive application of these four rules allows obtaining the so-called *Fitting's model* [35] of a normal logic program. Although we will not use Fitting's semantics in this thesis, it is interesting to note that Fitting's model would correspond[2] to assign these direct or trivial truth values until a cycle is reached (either positive or negative cycle), leaving undefined any atom involved in (or depending on) the cycle. In this way, the fifth transformation, $\stackrel{\mathtt{L}}{\longmapsto}$, becomes the real "contribution" of WFS with respect to Fitting's semantics. Positive cycles are solved by applying the following criterion: if, under an optimistic view, we compute the consequences of the program assuming all the default literals to be true (this is the real meaning of $\Gamma(\emptyset)$) but still some atoms cannot be obtained, then these atoms will always be false (since we had assumed the most optimistic case for default negation). This also means that, for a hierarchical program, the rewriting method for computing the WFM actually amounts to the rules $\stackrel{\mathtt{P}}{\longmapsto}$, $\stackrel{\mathtt{N}}{\longmapsto}$, $\stackrel{\mathtt{S}}{\longmapsto}$ and $\stackrel{\mathtt{F}}{\longmapsto}$, since $\stackrel{\mathtt{L}}{\longmapsto}$ exclusively affects to positive cycles.

Let us make a more detailed study on the interpretation of program cycles under the different semantics we have introduced.

[2]The usual definition of Fitting's model is as the $\leq_F$-least fixpoint of operator Φ_P, which is a three-valued generalization of T_P.

2.3.5 Cycles

Despite of the differences among all these semantics, it is possible to establish some syntactic cases in which their interpretations of a logic program coincides. Most of these syntactic restrictions are very related to the cyclic references present in the program. For instance, under a complete absence of cycles, the three semantics we have presented lead to the same result:

Property 9 *Let P be a hierarchical logic program. Then:*

1. *there exists a unique fixpoint M of T_P,*
2. *M is the unique stable model of P,*
3. *$(M, \mathcal{H} - M)$ is the WFM of P,*

$\square$

Furthermore, if we also require that the program is positive, we obtain:

Property 10 *Let P be a hierarchical and positive logic program. Then:*

$$\mathrm{COMP}[P] \equiv \mathrm{CIRC}[P; \mathcal{H}]$$

$\square$

In other words, for a program without cycles nor default negation, the parallel circumscription of all the Herbrand atoms is equivalent to the Clark's completion (and so, as the program is hierarchical, to the unique stable model and to the complete WFM). Of course, having no default negation at all is a too strong restriction, but this result will be interesting for computing circumscription in some particular cases.

Proposition 9 allows us to understand each semantics as a different way of interpreting logic program cycles. Let us think first, for instance, about positive cycles like $P_2 = \{p \leftarrow p\}$. As we already saw, this program has two supported models: $\{p\}$ and $\emptyset$. Note that model $\{p\}$ shows a "strange" feature: in order to explain which rules justify p, we need p itself as part of the explanation. Stable models disable these "auto-explanations," satisfying instead the property of *well-supportedness*:

Definition 10 (Well-supportedness) A (2-valued) interpretation M is called *well-supported* w.r.t. to a program P iff for some well-founded ordering $<$ on $\mathcal{H}$:

> $p \in \mathcal{H}$ implies that there exists some rule $p \leftarrow L$, $M \models L$ and $q < p$ for all the positive literals q in L

$\square$

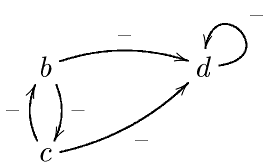
In other words, M is well-supported iff each $p \in M$ has an explanation that does not use p. In fact, the concept of being well-supported coincides with being a stable model (see for instance theorem 6.18 in [6]). This well-supported behavior for positive cycles is present also in the WFS, as shown by the property:

Property 11 *Let P be a stratified logic program, and M its unique stable model. Then, the complete 3-valued interpretation $(M, \mathcal{H} - M)$ is the well founded model of P.* $\square$

In other words, these two semantics only differ in the treatment of *negative* cycles. Roughly speaking, WFS tries to simplify the cases in which there is no stable model or there are multiple ones. This is done by identifying those atoms that can be clearly set as true or false, independently of problematic negative cycles. Using corollaries 1 and 2, it is easy to see that if a program has multiple stable models, any defined atom in its WFM will have the same valuation in all of them.

However, the WFM does not correspond exactly to the intersection of the stable models. For instance, in program P_5 the unique stable model is $\{b, d\}$ whereas the WFM leaves all the atoms undefined. This difference is related to a property of inference relations not satisfied by stable models: *cumulativity*. In few words, an entailment relation $\models$ is cumulative iff having $\alpha \models \beta$ and $\alpha \models \gamma$, adding one of the conclusions to α preserves the rest of conclusions: $\alpha \cup \beta \models \gamma$. Looking at program P_5, the unique stable model entails b and d true. However, once d is added to the program, we obtain two stable models $\{b, d\}$ and $\{c, d\}$, and b is no longer entailed.

The WFM of P_5 can be understood as an ordered interpretation of the cycles present in the program. Looking at the dependences graph:



we intuitively first solve the negative cycle between b and c, which in WFS leads to both atoms undefined. After that, d is also left undefined because it depends on the undefined atoms b and c, and it is also involved in another negative cycle.

2.3.6 WFSX and the coherence problem

As we explained previously, in order to use logic programming as a nonmonotonic reasoning tool, we have had to restrict the representation for dealing with atoms, rather than with arbitrary formulas, as in default logic. However, when we work with typical problems in reasoning about actions, we usually have to represent boolean (or even multiple-valued) fluents, needing to distinguish the fact of being explicitly false from the default negation. The addition of a second negation (named in different works as *classical*, *explicit* or *strong* negation) leads to the so called *extended logic programming* [40, 89, 2, 3].

We will handle now two types of atoms: 'p', to represent that p has value *true*; and '$\overline{p}$', to represent[3] that p has value *false*. Normal logic programs dealing with this extended signature receive the name of *extended logic programs*. We call *objective literal*, denoted as L, to either p or $\overline{p}$, and *default literal* to any *not* L. Besides, we will use the notation $\overline{L}$ to stand for the complementary objective literal of L, assuming $\overline{\overline{p}} \stackrel{\text{def}}{=} p$. Furthermore, given a set of atoms M, we denote $\overline{M}$ to stand for the set of complementary atoms:

$$\overline{M} \quad \stackrel{\text{def}}{=} \quad \{\overline{p} \mid p \in M\}$$

The extension of the stable models semantics for extended logic programming is extremely simple. We simply rule out the stable models containing any pair $\{p, \overline{p}\}$, that is, we take

[3]For our later convenience, we deviate here from the usual notation of extended logic programming which denotes objective negated literals as $\neg p$.

charge of explicit inconsistency. We usually talk about *answer sets* [40] when referring to these (consistent) stable models of extended logic programs.

At a first glimpse, it seems that something similar can be done for the WFS, considering the program to be *contradictory* when the WFM makes true both p and $\overline{p}$. However, as pointed out in [89, 2], a direct application of this method may lead to counterintuitive results. To understand the problem, it must be first noted that we handle now more possible epistemic states for a given atom. Rather than saying that L is *true* (when $L \in M^+$) or that it is *false* (when $L \in M^-$), we will say instead that it is *founded* or *unfounded*, respectively. In this way, we may distinguish between being *unknown* (that is, both $p \in M^-$ and $\overline{p} \in M^-$ are unfounded) and being *undefined*, which means that for some truth value of p we cannot establish whether it is founded or not ($p \notin M^+ \cup M^-$ or $\overline{p} \notin M^+ \cup M^-$).

In principle, we may have that p is undefined and $\overline{p}$ defined, or vice versa. However, it seems that there should exist a connection between complementary objective literals: when $L \in M^+$ is founded, we should have $\overline{L} \in M^-$ unfounded. Unfortunately, this property, called in [89] the *coherence principle*, is not satisfied by the usual definition of WFS. The typical example is the program P_6:

$$\begin{array}{lcl} p & \leftarrow & not\ q \\ q & \leftarrow & not\ p \\ \overline{p} & & \end{array}$$

Intuitively, as we know that $\overline{p}$ is founded, its default negation *not* p should be immediately true, making q also founded. That is, we should obtain the complete model $(\{\overline{p}, q\}, \{p, \overline{q}\})$, with $\overline{p}$ and q founded and their complements unfounded. However, it is easy to see that the WFM of P_6 is $(\{\overline{p}\}, \{\overline{q}\})$, which leaves both p and q undefined. The reason for this is that WFS does not provide any connection between p and $\overline{p}$ and so, we are not able to establish that the default literal *not* p should be true when p is foundedly false.

To overcome this difficulty, Alferes and Pereira introduced a variation of WFS called WFSX (Well Founded Semantics with Explicit Negation). For simplicity sake, we will just provide the iterative method to compute the WFM under WFSX semantics, which relies on a variation of the application of Γ^2. Most of the properties presented here have been directly extracted from [2]. Later, we will use this definition of WFSX to introduce a second computation method, which is an extension of Brass and Dix's transformations.

Let r be a rule $H \leftarrow B$ of an extended normal logic program. By r_s we denote the seminormal version of r:

$$r_s \stackrel{\text{def}}{=} H \leftarrow B, not\ \overline{H}$$

Given a extended normal program P, we write P_s to stand for the seminormal version of P:

$$P_s \stackrel{\text{def}}{=} \{r_s \mid r \in P\}$$

For any set of atoms M and any fixed program P, we write $\Gamma_s(M)$ to denote the least model of P_s^M. The function Γ_s is not defined for a contradictory M, i.e., M containing both p and $\overline{p}$. A program is *contradictory* in WFSX iff it has no fixpoints for $\Gamma\Gamma_s$.

Property 12 *For noncontradictory programs, there exists a least fixpoint of* $\Gamma\Gamma_s$*, denoted as* $lfp(\Gamma\Gamma_s)$*.* □

The combined function $\Gamma\Gamma_s$ is monotonic (on inclusion of sets of atoms), and so, its least fixpoint (when defined) can be computed by iteration on the least possible set of atoms, $\emptyset$. This is usually denoted as $\Gamma\Gamma_s \uparrow (\emptyset)$. The well founded model is then defined in terms of $lfp(\Gamma\Gamma_s)$ as follows:

Definition 11 (WFSX's Well founded model) The *well founded model* (WFM) of a (non-contradictory) extended logic program under the WFSX semantics corresponds to the three-valued interpretation:

$$\langle lfp(\Gamma\Gamma_s), \Gamma_s(lfp(\Gamma_s\Gamma))\rangle$$

□

It is interesting to note that the iteration of $\Gamma\Gamma_s$ may also be used to detect contradictory programs, as stated by the following property:

Property 13 *If the iteration of* $\Gamma\Gamma_s \uparrow (\emptyset)$ *reaches an interpretation that contains both* p *and* $\overline{p}$*, then the program is contradictory in WFSX.* □

Finally, another important property (proved in theorem 4.3.6 in [2]) is that WFSX actually generalizes WFS:

Property 14 *For programs without explicit negation, WFSX coincides with WFS.* □

Unfortunately, this property does not help to establish the differences between WFS and WFSX when the program actually contains explicit negation. To this aim, we introduce next an alternative characterization of WFSX that provides straightforward results comparing both semantics and helps in identifying when WFSX is actually needed.

2.3.7 A rewriting method for computing WFSX

As explained above, in this section we introduce an alternative method for computing the WFSX relying on the rewriting transformations defined by Brass and Dix. We will incorporate the following two new transformations:

6. *Coherence failure* $\stackrel{\mathtt{C}}{\longmapsto}$: delete all rules containing the positive literal $\overline{p}$ such that p is a fact.

7. *Coherence reduction* $\stackrel{\mathtt{R}}{\longmapsto}$: delete all literals *not* $\overline{p}$ such that p is a fact.

Notice how, in both cases, we simplify literals for $\overline{p}$ provided that p is trivially true (it is a fact). In such a case, the coherence reduction, $\stackrel{\mathtt{R}}{\longmapsto}$, transforms *not* $\overline{p}$ into true. In fact, this transformation is the direct implementation of the coherence principle: default negation follows from explicit negation. As for transformation $\stackrel{\mathtt{C}}{\longmapsto}$, it allows replacing $\overline{p}$ by false, momentarily assuming that the program will be noncontradictory. As we will show later, even though this assumption is not finally satisfied, the rewriting method (including these two new rules) is still capable of detecting contradiction.

We begin introducing the following definition:

Definition 12 (Trivial interpretation of a program) Let P be a program not containing contradictory facts. We say that $U = (U^+, U^-)$ is the *trivial interpretation* of P iff.

$$\begin{array}{rcl} U^+ & \stackrel{\text{def}}{=} & facts(P) \\ U^- & \stackrel{\text{def}}{=} & (\mathcal{H} - heads(P)) \cup \overline{facts(P)} \end{array}$$

□

In other words, we consider as true any atom which is a fact in the program, and as false any atom p which is not head or for which $\overline{p}$ is a fact.

Example 3 Consider the program P_7:

$$\begin{array}{rcl} a & \leftarrow & not\ b \\ b & \leftarrow & not\ a \\ \overline{p} & \leftarrow & b \\ p & & \end{array}$$

Its trivial interpretation is:

$$\begin{array}{rcl} U^+ & = & \{p\} \\ U^- & = & \{\overline{a}, \overline{b}, \overline{p}\} \end{array}$$

□

Notice that, this program cannot be further transformed using the WFS transformations $\{\mathtt{F},\mathtt{P},\mathtt{S},\mathtt{N},\mathtt{L}\}$ and its WFM would be:

$$\begin{array}{rcl} W^+ & = & \{p\} \\ W^- & = & \{\overline{a}, \overline{b}\} \end{array}$$

leaving $\overline{p}$ undefined, but the trivial interpretation goes even further, considering $\overline{p}$ unfounded, provided that p is a fact. The trivial interpretation is interesting because it will always contain less or equal information than the WFM (under WFSX), as stated by the following theorem:

Theorem 1 *Let P be a noncontradictory program, $W = (W^+, W^-)$ its well founded model (under WFSX) and $U = (U^+, U^-)$ its trivial interpretation. Then $U \leq_F W$.*
Proof
(In appendix A) □

We will prove now that the whole set of transformations, $\{\mathtt{F},\mathtt{P},\mathtt{S},\mathtt{N},\mathtt{L},\mathtt{C},\mathtt{R}\}$ are sound with respect to WFSX, that is, all the successive transformed programs either have the same WFM or are contradictory. To this aim, we provide first a lemma that establishes that the fixpoints for $\Gamma\Gamma_s$ remain unchanged after each transformation. We introduce here a remark on notation. When $P \stackrel{\mathtt{x}}{\longmapsto} P'$ with some transformation rule $\stackrel{\mathtt{x}}{\longmapsto}$, we write Γ' and Γ'_s to express that these functions implicitly correspond to P', instead of P.

Lemma 1 *For any transformation $\stackrel{\mathtt{x}}{\longmapsto}$ with $\mathtt{x} \in \{\mathtt{F},\mathtt{P},\mathtt{S},\mathtt{N},\mathtt{L},\mathtt{C},\mathtt{R}\}$, if $P \stackrel{\mathtt{x}}{\longmapsto} P'$ then:*

(a) *if* $M = \Gamma\Gamma_s(M)$ *then* $\Gamma_s(M) = \Gamma'_s(M)$ *and* $M = \Gamma'\Gamma'_s(M)$.

(b) *and vice versa, if* $M = \Gamma'\Gamma'_s(M)$ *then* $\Gamma_s(M) = \Gamma'_s(M)$ *and* $M = \Gamma\Gamma_s(M)$.

Proof (In appendix A) □

Using this lemma, the soundness theorem is almost straightforward.

Theorem 2 (Soundness) *The transformations* $\overset{\mathtt{x}}{\longmapsto}$, *with* $\mathtt{x} \in \{\mathtt{F},\mathtt{P},\mathtt{S},\mathtt{N},\mathtt{L},\mathtt{C},\mathtt{R}\}$ *are sound w.r.t. WFSX. In other words, if a program* P *has a WFM then any resulting* P', $P \overset{\mathtt{x}}{\longmapsto} P'$ *has the same WFM. Otherwise, if* P *has no WFM then* P' *has no WFM.*
Proof
(In appendix A) □

Before studying completeness, we can already use this result to compare WFSX to WFS. As the transformations $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$ used for WFS are a subset of the ones we have just proved to be sound for WFSX, we immediately get that:

Theorem 3 *Let* P *be any extended normal logic program and let* $W = (W^+, W^-)$ *be its WFM under WFS. Then:*

(i) *If* W *is contradictory (it makes true both* p *and* $\overline{p}$*) then* P *is contradictory in WFSX.*

(ii) *If the program* P *is noncontradictory and* X *is its WFM under WFSX then* $W \leq_F X$.

Proof
(In appendix A) □

Note that the opposite for (i) does not hold, that is, we may have a program which has a noncontradictory WFM in WFS but has no solution in WFSX. As a simple counterexample, consider the program P_8:

$$\begin{array}{lll} a & \leftarrow & not\ a \\ \overline{a} & & \end{array}$$

It is easy to see that in WFS, we get a undefined and $\overline{a}$ true (remember that there is no connection between both atoms). So, the result is not contradictory. However, in WFSX, there is no fixpoint for $\Gamma\Gamma_s$. For instance, if we try to compute its least fixpoint by iteration, we obtain that $\Gamma\Gamma_s(\emptyset) = \{\overline{a}\}$, and $\Gamma\Gamma_s(\{\overline{a}\}) = \{a, \overline{a}\}$ and so, it is not defined any more (WFSX has no fixpoints). If we focus on the transformation rules, note that P_8 cannot be further transformed using rules $\{\mathtt{P},\mathtt{N},\mathtt{S},\mathtt{F},\mathtt{L}\}$. However, in WFSX, we still can apply rule $\overset{\mathtt{R}}{\longmapsto}$ (coherence reduction): as a is explicitly false, *not a* must become true. As a result, we get the program $P'_8 = \{a, \overline{a}\}$ which is clearly contradictory.

Theorem 3 also leads to the less general, but also useful result:

Corollary 3 *Let* W *be the well founded model under WFS for some program* P, *and let* W *be noncontradictory and complete. Then,* W *is also the well founded model of* P *under WFSX.* □

To end up with the comparison, as for any hierarchical program, WFS leads to a complete WFM, we also obtain:

Corollary 4 *WFS and WFSX coincide for any hierarchical program.* □

Although these results help in comparing the effects of applying WFSX w.r.t. WFS for extended logic programs, we have not proved yet that applying exhaustively the complete set of transformations {F,P,S,N,L,C,R} actually ends providing the WFM in WFSX. We only know by now that, if we reach a pair of contradictory facts, then the program is WFSX-contradictory and, otherwise, the three-valued interpretation obtained from the program has less or equal information than the final WFM.

Definition 13 (Non-reducible program) An extended normal logic program P is said to be *non-reducible* iff no transformation in {F,P,S,N,L,C,R} is applicable for P. □

Theorem 4 *Let P be an non-reducible program not containing contradictory facts. Then, the trivial interpretation of P:*

$$\begin{aligned} U^+ &\stackrel{\text{def}}{=} facts(P) \\ U^- &\stackrel{\text{def}}{=} (\mathcal{H} - heads(P)) \cup \overline{facts(P)} \end{aligned}$$

is its WFM under WFSX, that is:

i) $U^+ = lfp(\Gamma\Gamma_s)$

ii) $U^- = \mathcal{H} - \Gamma_s(U^+)$

Proof
(In appendix A) □

Looking at the proof of the previous theorem (included in the appendix), it can be noticed that we use the premise of nonapplicability of all the transformations *excepting* the failure $\stackrel{\mathtt{F}}{\longmapsto}$ which, therefore, is not actually necessary. The explanation for this is that, as it is easy to see, failure is actually a particular case of positive loop detection $\stackrel{\mathtt{L}}{\longmapsto}$. Both transformations delete rules containing positive literals that satisfy a given condition which in $\stackrel{\mathtt{F}}{\longmapsto}$ is stronger than in $\stackrel{\mathtt{L}}{\longmapsto}$: $L \notin heads(P) \supset L \notin \Gamma(\emptyset)$. However, maintaining transformation $\stackrel{\mathtt{F}}{\longmapsto}$ is interesting because of a pair of reasons. On the one hand, it is less costful than loop detection and so, it may mean an efficiency improvement in many cases. On the other hand, it is interesting from the theoretical point of view, since, as we had seen, the subset of transformations {P, N, S, F} completely establish the Fitting's model of the program.

Chapter 3

Static L^2

In this chapter we overview the basic definitions of Otero's Pertinence Logic [85], although the presentation is closer to that of [86]. As we have explained in the introduction, the idea of pertinence (at least, in this dissertation) is absolutely related to the concept of change in dynamic systems. However, we will present now, for clarity sake, a formalization as *static* as possible, initially omitting features like actions, fluents, situations or even the nonmonotonic inertia default[1]. However, some implicit informal dynamic intuitions will be needed for a better comprehension.

Suppose that we handle a finite set of atoms or propositional variables, and that we study some particular state or configuration of their truth values, using to this aim a classical propositional interpretation. We want to go further and represent not only the truth values, but also the fact that some of them have are result of some external intervention[2] whose detailed description is not provided yet. Thus, we will say that an atom is *pertinent* when it happens to have been affected by this intervention, and we will say that it is *nonpertinent* otherwise. As a result, we extend the interpretation to capture this pertinence mapping for all the atoms. We will also bear in mind an hypothetical previous state (not explicitly represented yet) which would assign, for all the nonpertinent atoms, the same truth valuation than in the current state. Thus, we intuitively identify nonpertinence with persistence.

The main motivation for introducing a new logic is that this new pertinence information can also be extended to complex formulas. For instance, think about the formula $up(1) \wedge up(2)$. If the pertinence mapping points out that both $up(1)$ and $up(2)$ are nonpertinent, we know that the whole formula $up(1) \wedge up(2)$ would have the same truth value in a previous state, and so, we can say that *the formula* has persisted (it is nonpertinent). Thus, the pertinence mapping can also be extended to non-atomic formulas, becoming somehow a second valuation, parallel to the usual truth valuation.

3.1 L^2 syntax and semantics

We begin describing L^2 syntax by defining a finite nonempty set of propositional atoms, Σ, called the *signature*. An L^2 *formula* is recursively defined as follows:

[1] We deviate here from the original Otero's presentation which, in the basic definition of L^2, already includes a nonmonotonic entailment with a particular minimization process. Instead, we have preferred to delay the introduction of nonmonotonic reasoning until inertia is tackled (chapter 6).

[2] This "intervention-oriented" focusing seems, in fact, strongly related to Pearl's interpretation of causality for probabilistic reasoning (see for instance sections 1.3.1 and 3.4 in [88]).

Definition 14 (L^2 formula) Given any atom $a \in \Sigma$ and any pair ϕ, ψ of L^2 formulas, the following expressions are also L^2 *formulas*:

$$\bot,\ a,\ \neg\phi,\ \phi \wedge \psi,\ \phi \vee \psi,\ \phi \Leftarrow \psi$$

$\square$

We also include the usual propositional abbreviations:

$$\begin{aligned} \top &\stackrel{\text{def}}{=} \neg\bot \\ \phi \supset \psi &\stackrel{\text{def}}{=} \neg\phi \vee \psi \\ \phi \equiv \psi &\stackrel{\text{def}}{=} (\phi \supset \psi) \wedge (\psi \supset \phi) \end{aligned}$$

As we can see, the basic L^2 syntax introduces just one new connective w.r.t. classical propositional logic: the binary connective '$\Leftarrow$' which will be used in the encoding of causal rules. As usual, an L^2 *theory* is a set of L^2 formulas.

The semantics of L^2 is characterized by defining two valuation functions: one for the truth of the formula (as in classical logic) and the other for its pertinence. We define the set of truth values $\{\mathtt{t}, \mathtt{f}\}$ respectively standing for "true" and "false" and the set of pertinence values $\{\mathtt{p}, \mathtt{n}\}$ which correspond to "pertinent" and "nonpertinent" respectively. In this way, an L^2 *interpretation* M is a pair (M^t, M^p) of subsets of Σ that respectively point out the sets of true and pertinent atoms. The *double valuation* will be a function from the set of formulas to the set of pairs $\{\mathtt{t}, \mathtt{f}\} \times \{\mathtt{p}, \mathtt{n}\}$. For commodity sake, we will denote each one of these pairs simply as two consecutive letters, $\mathtt{tp}, \mathtt{tn}, \mathtt{fp}, \mathtt{fn}$, and so, we may also consider L^2 as a four-valued logic. A possible intuitive reading for the resulting combinations above, would respectively be: "caused true," "persists true," "caused false" and "persists false."

Definition 15 (Double valuation of a formula) Given an interpretation M, the corresponding *double valuation* of a formula ϕ, denoted as $M(\phi)$, is a pair of values $M^t(\phi)M^p(\phi)$, where $M^t(\phi) \in \{\mathtt{t}, \mathtt{f}\}$ is called the *truth value* of ϕ and $M^p(\phi) \in \{\mathtt{p}, \mathtt{n}\}$ is called its *pertinence value*, such that:

i) the interpretation of $\bot$ is fixed to false, as expected, additionally assuming that it does not yield pertinence:

- $M^t(\bot) = \mathtt{f}$,
- $M^p(\bot) = \mathtt{n}$.

ii) For any atom a:

- $M^t(a) = \mathtt{t}$ iff $a \in M^t$,
- $M^p(a) = \mathtt{p}$ iff $a \in M^p$.

iii) For any propositional formula ϕ:

- $M^t(\phi)$ is the usual Tarski's truth valuation of ϕ using M^t as a standard propositional interpretation, that is:

  (a) $M^t(\neg\phi) = \mathtt{t}$ iff $M^t(\phi) \neq \mathtt{t}$,

  (b) $M^t(\phi \wedge \psi) = \mathtt{t}$ iff $M^t(\phi) = \mathtt{t}$ and $M^t(\psi) = \mathtt{t}$,

ϕ	ψ	$\neg\phi$	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \Leftarrow \psi$
fn	fn	tn	fn	fn	tn
fn	fp	tn	fp	fp	tn
fn	tn	tn	fn	tn	tn
fn	tp	tn	fp	tp	fp
fp	fn	tp	fp	fp	tn
fp	fp	tp	fp	fp	tn
fp	tn	tp	fp	tp	tn
fp	tp	tp	fp	tp	fp
tn	fn	fn	fn	tn	tn
tn	fp	fn	fp	tp	tn
tn	tn	fn	tn	tn	tn
tn	tp	fn	tp	tp	fp
tp	fn	fp	fp	tp	tn
tp	fp	fp	fp	tp	tn
tp	tn	fp	tp	tp	tn
tp	tp	fp	tp	tp	tp

Figure 3.1: Double valuation of formulas.

(c) $M^t(\phi \vee \psi) = \mathtt{t}$ iff $M^t(\phi) = \mathtt{t}$ or $M^t(\psi) = \mathtt{t}$.

- $M^p(\phi) = \mathtt{p}$ iff there occurs an atom a in ϕ, such that $a \in M^p$. In other words:
  (a) $M^p(\neg\phi) = \mathtt{p}$ iff $M^p(\phi) = \mathtt{p}$,
  (b) $M^p(\phi \wedge \psi) = \mathtt{p}$ iff $M^p(\phi) = \mathtt{p}$ or $M^p(\psi) = \mathtt{p}$,
  (c) $M^p(\phi \vee \psi) = \mathtt{p}$ iff $M^p(\phi) = \mathtt{p}$ or $M^p(\psi) = \mathtt{p}$.

iv) For any pair of L^2 formulas, ϕ and ψ:

- $M^t(\phi \Leftarrow \psi) = \mathtt{t}$ iff $M(\psi) \neq \mathtt{tp}$ or $M(\phi) = \mathtt{tp}$,
- $M^p(\phi \Leftarrow \psi) = \mathtt{p}$ iff $M(\psi) = \mathtt{tp}$.

□

As we can see, the two values of classical propositional formulas are obtained independently (the truth is not used for pertinence and vice versa). Notice that, for these formulas, pertinence is just an "occurrence" test: it checks whether the formula contains some pertinent atom or not. The reason for this is that, when no atom in ϕ is pertinent, we will have the guarantee that its truth value has not changed with respect to the previous state. The presence of a pertinent atom in ϕ drops this guarantee, and so the whole formula is interpreted as pertinent. An alternative tabular description of L^2 double valuation is provided in figure 3.1.

We will say that an interpretation M is a *model* of an L^2 theory T iff for any formula $\phi \in T$, $M^t(\phi) = \mathtt{t}$. In other words, a model is an interpretation that assigns "true" to all formulas in T, regardless their pertinence. We also define several kinds of equivalence. Two formulas ϕ and ψ are said to be *truth equivalent* (resp. *pertinence-equivalent*) iff $M^t(\phi) = M^t(\psi)$ (resp. $M^p(\phi) = M^p(\psi)$) for any interpretation M. We say that ϕ and ψ are *equivalent* when they are both truth and pertinence-equivalent, that is, $M(\phi) = M(\psi)$ for any interpretation M. It must be noticed that, in order to get the same set of models for two formulas ϕ and ψ, it is enough with requiring their truth equivalence, since the definition of model exclusively depends on the truth value.

3.2 Examples

Let $\Sigma = \{a, b, c, d, e\}$ and let M_0 be the interpretation:

$$\begin{aligned} M_0^t &= \{a, b, e\} \\ M_0^p &= \{b, c, e\} \end{aligned}$$

Then, the double valuation of atoms corresponds to:

$$\begin{aligned} M_0(a) &= \texttt{tn} \\ M_0(b) &= \texttt{tp} \\ M_0(c) &= \texttt{fp} \\ M_0(d) &= \texttt{fn} \\ M_0(e) &= \texttt{tp} \end{aligned}$$

Intuitively this will mean that our current state for truth values M_0^t has been obtained because of an intervention in atoms M_0^p, that is, in atoms b, c and e. The truth value of a formula like $\neg(a \vee d)$ is simply obtained by classical propositional valuation using M^t, i.e., $M_0^t(\neg(a \vee d)) = \texttt{f}$. As for its pertinence value, since neither a nor d belong to M_0^p, the formula is nonpertinent: $M_0^p(\neg(a \vee d)) = \texttt{n}$. Thus:

$$M_0(\neg(a \vee d)) = \texttt{fn}$$

This intuitively points out that the formula is currently false and that there was no external intervention to obtain such a truth value. In other words, we know for sure that the formula would have also been false in some sort of previous state.

As another example, $M_0(a \wedge \neg c) = \texttt{tp}$ which means that the formula has been caused true. That is, we know that the formula is true now but, since one of its atoms, c, has been affected by change, we cannot guarantee that the formula has really persisted true with respect to a previous state (c could have been false before, or moreover, it could have happened to be also true, but the intervention may have made it momentarily unstable).

Let us see now some examples about L^2 conditionals. A conditional $\phi \Leftarrow \psi$ is true when, if ψ is true and pertinent then ϕ is also true and pertinent. For instance, we have that $M_0^t(c \Leftarrow a) = \texttt{t}$, since the antecedent a is nonpertinent. We also have that $M_0^t(e \Leftarrow \neg c) = \texttt{t}$, since both the antecedent and the consequent are true and pertinent. Pertinence of a conditional corresponds to requiring that the antecedent is true and pertinent. That is:

$$\begin{aligned} M_0(c \Leftarrow a) &= \texttt{tn} \\ M_0(e \Leftarrow \neg c) &= \texttt{tp} \end{aligned}$$

As an example of a false conditional, $M_0(a \Leftarrow e) = \texttt{fp}$, since the antecedent is true and pertinent but the consequent is not pertinent.

Conditionals in L^2 are somehow atypical operators in the sense that they do not satisfy several "standard" properties in other conditional logics. For instance, an L^2 conditional is sensitive to (truth) tautologies, so that $\phi \Leftarrow \psi$ is not generally equivalent to $\phi \Leftarrow \psi \wedge (\alpha \vee \neg\alpha)$. As an example, think for instance in the pair of rules:

$$d \Leftarrow a \tag{3.1}$$
$$d \Leftarrow a \wedge (c \vee \neg c) \tag{3.2}$$

under interpretation M_0. On the one hand $M_0(3.1) = \mathtt{tn}$, since $M_0(a) = \mathtt{tn}$ and the antecedent is directly nonpertinent. On the other hand, $M_0(3.2) = \mathtt{fp}$, because the antecedent is still true, but *has become pertinent* due to the reference to c, $M_0(a \wedge (c \vee \neg c)) = \mathtt{tp}$, whereas the consequent $M_0(d) = \mathtt{fn}$.

This effect of relevance of tautologies can only be explained if we recall the dynamic intuition about pertinence. Consider, for instance, the combinatorial circuit in figure 3.2. Assume that $a = 1$ and that we change the value of c from 0 to 1. It is clear that, in the resulting state, the output of the OR gate will always be 1 (it represents a tautology) and so, d will have the same value as a. However, along the change of c, we cannot guarantee a complete persistence of d: the output of the OR gate may be momentarily undefined, affecting also to the AND gate. Therefore, d must result pertinent, that is, *caused true* because c is modified. On the other hand, b is not affected at all by c, and so, we can guarantee that it *persists false*.

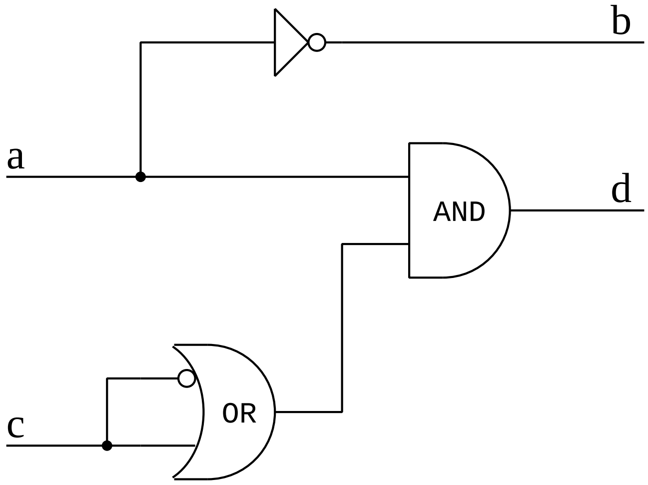


Figure 3.2: A simple combinatorial circuit.

Another property that is not generally satisfied is the decomposition of a disjunctive antecedent into two rules. For instance, interpretation M_0 is model of the theory T_1:

$$d \Leftarrow a \tag{3.3}$$
$$d \Leftarrow c \tag{3.4}$$

but is not model of T_2:

$$d \Leftarrow a \vee c \tag{3.5}$$

The formula (3.5) is valuated as false in M_0, since $M_0(c \vee a) = \mathtt{tp}$ whereas $M_0(d) = \mathtt{fn}$. However both (3.3) and (3.4) are valuated as true since $M_0(a) \neq \mathtt{tp}$ and $M_0(c) \neq \mathtt{tp}$. The reason for this nonequivalence relies on the valuation of the disjunction: the truth of formula $a \vee c$ is provided by a whereas its pertinence is provided by c.

Something similar happens for a conjunctive consequent. For instance, M_0 is model of:

$$a \wedge e \quad \Leftarrow \quad b \tag{3.6}$$

but is not model of the theory T_3:

$$e \quad \Leftarrow \quad b \tag{3.7}$$
$$a \quad \Leftarrow \quad b \tag{3.8}$$

The reason for this is that, in order to make pertinent the conjunction $a \wedge e$, it is enough to make one of its atoms pertinent. However, if we separate these atoms in different rules, then each of them must be pertinent.

As an interesting remark, note that conditionals can be used to fix the pertinence of any given atom. Consider the formula:

$$\bot \Leftarrow (a \vee \neg a)$$

Since its consequent is always false, and its antecedent always true, the only way in which the conditional can be made true is by making a nonpertinent. In a similar way, the formula $\neg(\bot \Leftarrow a \vee \neg a)$ can be used to represent that a must be pertinent. Of course, these constructions are slightly bizarre and, as we will see in the next section, we will define additional operators that allow a more comfortable representation. However, the following result shows that, in fact, the $\Leftarrow$ operator is enough for covering all the expressivity needed for L^2 models:

Theorem 5 *Let C be an arbitrary set of L^2 interpretations. There always exist an L^2 theory, let us call it T_C, such that its set of models is precisely C.*

Proof

We can define for each $M_i \in C$, the conjunction $\phi(M_i)$:

$$\{a \mid M_i^t(a) = \mathtt{t}\} \cup \{\neg a \mid M_i^t(a) = \mathtt{f}\} \cup \{\neg(\bot \Leftarrow a \vee \neg a) \mid M_i^p(a) = \mathtt{p}\} \cup \{(\bot \Leftarrow a \vee \neg a) \mid M_i^p(a) = \mathtt{n}\}$$

and then construct T_C simply as:

$$T_C = \bigvee_i \phi(M_i)$$

where we assume that an empty disjunction is equivalent to $\bot$. Then, using the already proved effect of $\bot \Leftarrow a \vee \neg a$, by an straightforward application of truth-satisfaction of formulas in L^2, we get that the set of models of T_C is precisely C. □

3.3 Extending L^2 syntax

The number of operators that can be defined in a logic with 4 possible values is considerably greater than in classical propositional logic. To compute the number of possible unary operators, the truth of $\langle op \rangle \phi$ can be described as the subset of the 4 values for ϕ that lead to $\mathtt{t}$, for instance. Analogously, its pertinence value, can be described as the subset of values that lead to $\mathtt{p}$. In this way, we have 2^4 possible truth valuations and 2^4 possible pertinence valuations, that is,

ϕ	$!\phi$	$\mathbb{C}\phi$	$\mathbb{P}\phi$
fn	fn	fn	fn
fp	tp	fp	fp
tn	fn	fn	tn
tp	tp	tp	fp

Figure 3.3: Extended L^2 syntax: unary operators.

$2^4 \cdot 2^4 = 256$ possible unary operators. But this amount is small compared to the number of binary operators. For each pair of formulas ϕ, ψ, we have $4 \cdot 4 = 16$ possible cases. The truth and pertinence valuations for a binary operator $\phi\langle op\rangle\psi$ are subsets of these 16 cases: i.e. $2^{16} \cdot 2^{16} = 2^{32}$. In other words, more than 4 billions possible binary operators!

Despite of this considerable amount of possibilities, not all the imaginable operators have a clear meaning with respect to the postulates of pertinence we wish to represent formally, and so, we will focus on the ones specially related to the use of L^2 for action domains. For instance, although we have seen that the formula $\neg(\bot \Leftarrow a \vee \neg a)$ allows fixing a as pertinent, it is clearly more elegant to define an unary operator for this purpose. Thus, we denote $!\phi$ to stand for "formula ϕ is pertinent," and, therefore, its truth value is defined as:

i) $M^t(!\phi) = \mathtt{t}$ iff $M^p(\phi) = \mathtt{p}$

Notice that, in its turn, we must also define the pertinence of $!\phi$. To this aim, we simply follow the same criterion as for classical operators: the formula is pertinent iff ϕ contains some pertinent atom[3]. In other words:

ii) $M^p(!\phi) = \mathtt{p}$ iff $M^p(\phi) = \mathtt{p}$

For our convenience, we introduce the abbreviations:

$$\begin{aligned} \mathbb{C}\phi &\stackrel{\text{def}}{=} \phi\wedge!\phi \\ \mathbb{P}\phi &\stackrel{\text{def}}{=} \phi \wedge \neg!\phi \end{aligned}$$

respectively standing for "ϕ is caused" (the formula is true and pertinent) and "ϕ persists" (the formula is true, but not pertinent). The resulting valuations for these three unary operators are described by table in figure 3.3.

As an interesting result, notice that a conditional $\phi \Leftarrow \psi$ is truth equivalent to the formula[4] $\mathbb{C}\psi \supset \mathbb{C}\phi$, that is, they have the same models.

Together with these unary operators, we will also introduce what we will call the *safe* versions of classical conjunction and disjunction, respectively '&' and '|'. The motivation for these new operators can be explained by the following example.

[3]It can be objected that this is perhaps an arbitrary criterion. For instance, when ϕ is currently nonpertinent but was pertinent in the previous state, the formula $!\phi$ *changes* its truth value, and so, it should also become pertinent, if we understand that postulate P3 is applicable to formulas, and not only to fluents. However, we will later restrict the study to transition systems where the pertinence of the current state becomes an output function, independent from the pertinence of the previous state.

[4]As explained later, in Section 8.8, despite of the strong similarity between these conditionals and Turner's UCL causal rules [111], the interpretation of the $\mathbb{C}$ ("caused") operator is very different in each case.

Example 4 Assume we want to encode the lamp domain using the formulas:

$$light \Leftarrow sw(1) \wedge sw(2) \tag{3.9}$$
$$\neg light \Leftarrow \neg sw(1) \vee \neg sw(2) \tag{3.10}$$

Assume also that, under a given interpretation M_1, $sw(1)$ persists false but $sw(2)$ has become true and pertinent:

$$M_1(sw(1)) = \texttt{fn}$$
$$M_1(sw(2)) = \texttt{tp}$$

□

The valuation of the antecedent in (3.10) would become true and pertinent:

$$M_1(\neg sw(1) \vee \neg sw(2)) = \texttt{tp}$$

and so, if M_1 is model of (3.9) and (3.10), we obtain that $M_1(light) = \texttt{fp}$ must hold. Thus, we obtain that $\neg light$ is pertinent: it does not hold by persistence, but as a result of applying a conditional. It can be objected that, *depending on how the switches circuit is physically built*, we may sometimes be sure that when one of the switches is off, the light is not affected even though we perform actions on the other switch. In this sense, we can say that the switches mechanism is "safe" with respect to single manipulations.

In order to represent systems like this we define the *safe conjunction*, $\phi \& \psi$, so that whenever one of the conjuncts persists false, the whole formula persists false. Analogously, the *safe disjunction*, $\phi|\psi$, becomes true and non-pertinent when at least one of its disjuncts has persisted true, i.e., is true and non-pertinent. The complete valuations for these two operators is described in table of figure 3.3. The differences with respect to $\wedge$ and $\vee$ have been underlined for a better comparison. As an interesting property, it can be easily checked that:

1. $\neg(\phi \& \psi)$ is equivalent to $\neg\phi|\neg\psi$,
2. $\neg(\phi|\psi)$ is equivalent to $\neg\phi \& \neg\psi$,

Thus, using these safe operators:

$$light \Leftarrow sw(1) \& sw(2) \tag{3.11}$$
$$\neg light \Leftarrow \neg sw(1) | \neg sw(2) \tag{3.12}$$

we have that the antecedent of rule (3.12) directly persists:

$$M_1(\neg sw(1) | \neg sw(2)) = \texttt{tn}$$

and so it does not force $\neg light$ to become pertinent.

3.4 Encoding L^2 into classical logic

For a better understanding, we will include a translation of L^2 into classical logic. To this aim, we provide first a set of transformations that translate any L^2 formula into a propositional

ϕ	ψ	$\phi \& \psi$	$\phi \vert \psi$
fn	fn	fn	fn
fn	fp	f**n**	fp
fn	tn	fn	tn
fn	tp	f**n**	tp
fp	fn	f**n**	fp
fp	fp	fp	fp
fp	tn	fp	t**n**
fp	tp	fp	tp
tn	fn	fn	tn
tn	fp	fp	t**n**
tn	tn	tn	tn
tn	tp	tp	t**n**
tp	fn	f**n**	tp
tp	fp	fp	tp
tp	tn	tp	t**n**
tp	tp	tp	tp

Figure 3.4: Valuation of "safe" operators.

combination of atomic expressions with shape a or $!a$, for any $a \in \Sigma$:

$$\phi \Leftarrow \psi \longrightarrow \mathbb{C}\psi \supset \mathbb{C}\phi \quad (3.13)$$
$$\mathbb{C}\phi \longrightarrow \phi \wedge !\phi \quad (3.14)$$
$$\mathbb{P}\phi \longrightarrow \phi \wedge \neg !\phi \quad (3.15)$$
$$\phi \& \psi \longrightarrow \phi \wedge \psi \quad (3.16)$$
$$\phi | \psi \longrightarrow \phi \vee \psi \quad (3.17)$$
$$!\neg\phi \longrightarrow !\phi \quad (3.18)$$
$$!(\phi \vee \psi) \longrightarrow !\phi \vee !\psi \quad (3.19)$$
$$!(\phi \wedge \psi) \longrightarrow !\phi \vee !\psi \quad (3.20)$$
$$!(\phi \& \psi) \longrightarrow (!\phi \wedge !\psi) \vee (!\phi \wedge \psi) \vee (!\psi \wedge \phi) \quad (3.21)$$
$$!(\phi | \psi) \longrightarrow (!\phi \wedge !\psi) \vee (!\phi \wedge \neg\psi) \vee (!\psi \wedge \neg\phi) \quad (3.22)$$
$$!!\phi \longrightarrow !\phi \quad (3.23)$$

It can be easily seen that these rules are sound with respect to L^2 truth-equivalence. If we exhaustively apply these transformation rules, it is easy to see that the shape of the final expressions would only present a unique nonclassical feature: the application of operator ! to atoms. This allows us to rename any atom in the scope of ! as:

$$!a \longrightarrow pert(a) \quad (3.24)$$

and the rest of atoms as:

$$a \longrightarrow holds(a) \quad (3.25)$$

We can understand the final theory as a classical propositional one, using the signature:

$$\Sigma^* = \{holds(a) : a \in \Sigma\} \cup \{pert(a) : a \in \Sigma\}$$

Let $M^* \subseteq \Sigma^*$ be a classical interpretation for this type of theories. The correspondence between M^* and an L^2 interpretation M is straightforward:

$$\begin{array}{rcl} M^t & = & \{a : holds(a) \in M^*\} \\ M^p & = & \{a : pert(a) \in M^*\} \end{array}$$

This result is interesting from a practical point of view, since it allows us to consider L^2 operators as mere syntactic abbreviations that can be "unfolded" using rules (3.13)-(3.25) to handle classical propositional semantics afterwards. From a purist point of view, however, the L^2 double valuation is actually needed if one wants to provide a real semantics for these syntactic transformations.

As an example of application of rules (3.13)-(3.23), we transform the formula (3.9) consecutively as:

$$\begin{array}{ll}
\mathbb{C}(sw(1) \wedge sw(2)) \supset \mathbb{C}light & \text{by rule (3.13)} \\[2ex]
sw(1) \wedge sw(2) \wedge !\big(sw(1) \wedge sw(2)\big) \supset light \wedge !light & \text{by rule (3.14)} \\[2ex]
sw(1) \wedge sw(2) \wedge \big(!sw(1) \vee !sw(2)\big) \supset light \wedge !light & \text{by rule (3.20)} \\[2ex]
holds(sw(1)) \wedge holds(sw(2)) \wedge & \\
\quad \big(pert(sw(1)) \vee pert(sw(2))\big) \supset holds(light) \wedge pert(light) & \text{by rules (3.24) and (3.25)}
\end{array}$$

The most complex transformations are perhaps those for the safe operators, '&' and '|'. For instance, the formula $\mathbb{C}(\neg sw(1) | \neg sw(2))$ successively becomes:

$$\begin{array}{ll}
\big(\neg sw(1) | \neg sw(2)\big) \wedge !\big(\neg sw(1) | \neg sw(2)\big) & \text{by (3.13)} \\[2ex]
\big(\neg sw(1) \vee \neg sw(2)\big) \wedge !\big(\neg sw(1) | \neg sw(2)\big) & \text{by (3.17)} \\[2ex]
\big(\neg sw(1) \vee \neg sw(2)\big) \wedge & \\
\quad \Big(\big(!\neg sw(1) \wedge !\neg sw(2)\big) \vee \big(!\neg sw(1) \wedge sw(2)\big) \vee \big(!\neg sw(2) \wedge sw(1)\big)\Big) & \text{by (3.21)} \\[2ex]
\big(\neg sw(1) \vee \neg sw(2)\big) \wedge & \\
\quad \Big(\big(!sw(1) \wedge !sw(2)\big) \vee \big(!sw(1) \wedge sw(2)\big) \vee \big(!sw(2) \wedge sw(1)\big)\Big) & \text{by (3.18)} \\[2ex]
\big(\neg holds(sw(1)\big) \vee \neg holds(sw(2))\big) \wedge & \\
\quad \Big(\big(pert(sw(1)) \wedge pert(sw(2))\big) \vee & \\
\quad \big(pert(sw(1)) \wedge holds(sw(2))\big) \vee & \\
\quad \big(pert(sw(2)) \wedge holds(sw(1))\big) \ \Big) & \text{by (3.24) and (3.25)}
\end{array}$$

Chapter 4

Transition systems

Once we have seen the "static" version of pertinence, we will analyze how to apply this concept to dynamic systems, focusing by now on their components and features, rather than on a logical formalization, which is delayed until chapter 5. We introduce first a typical transition-based framework[1] for reasoning about actions and change, paying special attention to automata descriptions where the states contain the information about the fluent values, whereas the transitions are labeled with the performed actions. The idea of pertinence is incorporated into this scheme by defining an output function that completes the information provided by the state.

4.1 Narrative action domains

As in most approaches for reasoning about actions, we begin identifying two types of entities:

- a finite nonempty set $\mathcal{F}$ of *fluents*[2], which will be properties of the system that vary along time,
- a finite nonempty set $\mathcal{A}$ of *actions*, which will be the way in which an exogenous agent may "operate" the system, causing it to change from one state to another.

To put an example, we define the fluents $\mathcal{F} = \{up(1), up(2), open\}$, and the actions $\mathcal{A} = \{toggle(1), toggle(2)\}$ (for alternatively changing the position of each lock). We will indistinctly call *symbol* to any fluent or action, that is, to any $p \in \mathcal{A} \cup \mathcal{F}$. At each moment in time, each symbol p may be associated to a unique value v, which in the switches example, corresponds to one of the boolean values $\{\mathtt{t}, \mathtt{f}\}$ (standing for "true" and "false" respectively).

Although most examples will deal with boolean domains, we will establish, for the general case, a third nonempty set, $\mathcal{V}$, called the set of possible *values* and which, for simplicity sake, we will assume to be finite. In this way, for any symbol p, we denote $range(p) \subseteq \mathcal{V}$, $range(p) \neq \emptyset$, to point out the possible values that can be associated to p along time[3]. We will use letters p, a,

[1] This initial transition-based framework has been partly inspired by the definitions described in [42].

[2] The term was coined by Newton as part of his celebrated *method of fluxions* which was the origin (together with Leibniz's contribution) of the infinitesimal calculus. Newton's understanding of a curved path was strongly bound to the idea of a point moving through space for a period of time. This moving point was called *fluent* and its velocity received the name of *fluxion*.

[3] Although this set of values is sometimes referred using the term *domain*, we have preferred to use the term *range*, reserving the former for the possible values of the symbol parameters, so that we follow the standard functional terminology.

f and v for respectively denoting arbitrary symbol, action, fluent and value names. An *action signature* is defined as the tuple $\langle \mathcal{F}, \mathcal{A}, \mathcal{V}, range \rangle$.

While most action approaches consider that fluents have an associated value, it is not so usual that this same feature holds for actions. To put some examples of its utility, think about an action *push* with a given strength, an action $move(block_1)$ with a given target location, or an action *set_bit* with a given truth value. In most action formalizations, the "action value" is frequently encoded as one more parameter: $push(7), move(block_1, table), set\_bit(\mathtt{f})$, etc. Once we introduce action values, it is necessary to explain what happens when an action is not performed. We will assume that any non-performed has *no defined value*. In this way, contrarily to many approaches, which represent this case by asserting that the action "is false," in our case, we allow distinguishing between an action (like *set_bit*) performed with value f, and the case in which it is not performed at all. Following this criterion, most actions used in boolean domains (like $toggle(1)$) will have the singleton range $\{t\}$.

At each moment in time, we will represent a particular world configuration in which each symbol will have (at most) one associated value. This mapping is separatedly defined for fluents and actions as follows.

Definition 16 (Fluents state) A *fluents state* is a function $\sigma : \mathcal{F} \longrightarrow \mathcal{V}$ that, for any fluent f, maps a value $\sigma(f) \in range(f)$. □

Definition 17 (Compound action) A *compound action* is a partial function $\alpha : \mathcal{A} \rightharpoonup \mathcal{V}$ that maps for some of the actions $a \in \mathcal{A}$ a value $\alpha(a) \in range(a)$. □

We will usually represent these functions as sets of elementary pairs called *value facts.*

Definition 18 (Value Fact) Given a symbol $p \in \mathcal{A} \cup \mathcal{F}$ and any value $v \in range(p)$, a *value fact* is defined as the construction: $holds(p, v)$. When $v \in \{\mathtt{t}, \mathtt{f}\}$ we allow the abbreviations p, $\overline{p}$ for respectively denoting $holds(p, \mathtt{t})$, $holds(p, \mathtt{f})$. □

Using value facts, a possible state could be expressed as:

$$\sigma_0 = \{\overline{up(1)}, up(2), \overline{open}\} = \{holds(up(1), \mathtt{f}), holds(up(2), \mathtt{t}), holds(open, \mathtt{f})\}$$

As examples of compound actions, we could have, for instance:

$$\begin{aligned} \alpha_1 &= \emptyset \\ \alpha_2 &= \{toggle(1)\} = \{holds(toggle(1), \mathtt{t})\} \\ \alpha_3 &= \{toggle(1), toggle(2)\} = \{holds(toggle(1), \mathtt{t}), holds(toggle(2), \mathtt{t})\} \\ &\vdots \end{aligned}$$

Of course, since σ and α actually represent functions, when we represent them as sets of value facts, they cannot contain two different facts for the same symbol p. We say that any set of facts is *consistent* when it satisfies this restriction. Furthermore, since σ represents a complete function, it must contain a fact for each possible value. A set of fluent facts satisfying (resp. not satisfying) this constraint is called *complete* (resp. *partial*). In order to denote the symbol of a given fact, we write:

$$symb(holds(p, v)) \stackrel{\text{def}}{=} p$$

For some scenarios, actions cannot be performed concurrently. We say that an scenario has *nonconcurrent actions* iff any compound action α has zero or one element ($|\alpha| \leq 1$), and we say that allows *concurrent actions* otherwise.

In order to represent time, we rely on temporal sequences of states, rather than on real numbers. For simplicity sake, our approach will handle a linear representation.

Definition 19 (Narrative) We define a *narrative*, ν, as a finite sequence:

$$\sigma_0, \quad \alpha_1, \sigma_1, \quad \ldots, \quad \alpha_{n-1}, \sigma_{n-1}, \quad \alpha_n, \sigma_n \tag{4.1}$$

with $n \geq 0$. □

The integer value n is called the *length* of the narrative (we will always consider finite narratives). We call *situation* to each one of the integer positions $i \in [0, n]$. Note the difference between a state, which may appear several times in the narrative, and a situation, which is an unique index. Situation 0 is called the *initial* situation.

We use the (somehow) nonstandard convention of placing actions in the same situation than their effects (other approaches [70, 42] place actions in the precedent situation). In this way, we may assume that no action is performed at the initial situation, considering the implicit existence of a fixed $\alpha_0 = \emptyset$, which will be usually omitted.

An example of narrative ν_1 could be the following sequence:

$$\begin{array}{rcl}
\sigma_0 & = & \{\overline{up(1)}, \overline{up(2)}, \overline{open}\} \\
\alpha_1 & = & \{toggle(2)\} \\
\sigma_1 & = & \{\overline{up(1)}, up(2), \overline{open}\} \\
\alpha_2 & = & \{toggle(1), toggle(2)\} \\
\sigma_2 & = & \{up(1), \overline{up(2)}, \overline{open}\} \\
\alpha_3 & = & \{toggle(2)\} \\
\sigma_3 & = & \{up(1), up(2), open\}
\end{array}$$

An alternative representation for narratives which will be closer to their logical formalization is to handle "situated facts," that is, facts with a third argument pointing out a situation index. We will call *atoms* to these situated facts, since they will later give shape to our propositional signature.

Definition 20 (Atom) An *atom* is a structure $holds(p, v, i)$ where $holds(p, v)$ is a value fact and i is any situation index $i \in [0, n]$. When $v \in \{\mathtt{t}, \mathtt{f}\}$ we allow the abbreviations p_i, $\overline{p}_i$ for respectively denoting $holds(p, \mathtt{t}, i)$ and $holds(p, \mathtt{f}, i)$. □

As we can see, any state or compound action in a narrative can be simply represented as a set of atoms. For any situation $i \in [0, n]$, we denote:

$$\begin{array}{rcl}
atoms(\sigma_i) & \stackrel{\text{def}}{=} & \{holds(f, v, i) \mid holds(f, v) \in \sigma_i\} \\
atoms(\alpha_i) & \stackrel{\text{def}}{=} & \{holds(a, v, i) \mid holds(a, v) \in \alpha_i\}
\end{array}$$

and also, for the whole narrative (4.1):

$$atoms(\nu) \stackrel{\text{def}}{=} \bigcup_{i \in [0,n]} (atoms(\alpha_i) \cup atoms(\sigma_i))$$

In this way, the narrative ν_1 would simply become the set:

$$\begin{aligned} atoms(\nu_1) = \{ \; & \overline{up(1)}_0, \overline{up(2)}_0, \overline{open}_0, \\ & toggle(2)_1, \overline{up(1)}_1, up(2)_1, \overline{open}_1, \\ & toggle(1)_2, toggle(2)_2, up(1)_2, \overline{up(2)}_2, \overline{open}_2, \\ & toggle(2)_3, up(1)_3, up(2)_3, open_3 \; \} \end{aligned}$$

which can also be seen as a propositional interpretation. Notice that for obtaining ν_1 from $atoms(\nu_1)$ there is no ambiguity: atoms referring to action symbols describe the compound actions α_i, whereas atoms referring to fluents describe the states σ_i.

The *system behavior*, $\mathcal{B}$, is a subset of all the possible narratives. For instance, we could describe our switches example by mentioning all the sequences (like the one seen above) that agree with the physical behavior of the suitcase. Of course, a description like this is clearly unfeasible (the narratives length is finite, but as long as desired) and so, we will need to handle a more compact representation. To this aim, in [42], for instance, two different types of languages are defined:

1. *Action description languages*, used for establishing the system behavior $\mathcal{B}$.

2. *Action query languages*, used for checking properties satisfied by the narratives in $\mathcal{B}$.

In our case, we will introduce in the next chapter a basic description language consisting of elementary causal constructions that will allow obtaining the set $\mathcal{B}$ of allowed narratives. As for the query language, it will simply consist of observations:

Definition 21 (Observation) We call *observation* to any propositional combination of atoms. □

As an example:

$$holds(up(1), \mathtt{t}, 1) \wedge holds(up(2), \mathtt{f}, 2) \vee \neg holds(up(1), \mathtt{t}, 3)$$

4.2 Typical problems in action domains

Given a system behavior $\mathcal{B}$, a narrative $\nu \in \mathcal{B}$ and some observation ϕ, it is usual to define the satisfaction relation $\models_{\mathcal{B}}$ so that by $\nu \models_{\mathcal{B}} \phi$ we mean that ϕ is satisfied by ν. We can also extend the use of this same symbol, $\models_{\mathcal{B}}$ to denote the entailment relation among observations, so that $\phi \models_{\mathcal{B}} \psi$ means that all the narratives (in $\mathcal{B}$) satisfying the set of observations ϕ, also satisfy observations ψ. Given this entailment relation, we can generally describe the usual tasks to be solved in typical problems about action domains.

Definition 22 (Temporal projection or prediction) A *temporal projection* (or *prediction*) problem consists in, given an initial state σ_0 and a sequence of compound actions $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$ decide whether:

$$atoms(\sigma_0) \cup atoms(\vec{\alpha}) \models_{\mathcal{B}} \phi$$

for some observation ϕ usually referring to the final situation n. □

For instance, given $\sigma_0 = \{\overline{up(1)}, \overline{up(2)}, \overline{open}\}$ and $\alpha_1 = \{toggle(1)\}$, $\alpha_2 = \{toggle(1), toggle(2)\}$ and $\alpha_3 = \{toggle(1)\}$ we want to predict whether this entails $open_3$ or not.

Definition 23 (Temporal explanation or postdiction) A *temporal explanation* (or *postdiction*) problem consists in, given some (usually incomplete) set of observations Obs about the actions execution and the resulting state, try to provide consistent explanations about the observed outcome. Formally, the temporal explanation is some set of observations $Expl$ such that:

$$Obs \cup Expl \not\models_{\mathcal{B}} \perp$$

that is, there exists some narrative ν that satisfies $Obs \cup Expl$. □

Usually, the observations Obs provide a complete actions execution and the explanation $Expl$ must complete the description of the initial or intermediate states. However, as we can handle concurrent actions, we may also allow incomplete action descriptions. For instance, given the observation:

$$toggle(2)_1 \wedge \overline{open}_1 \wedge up(2)_1$$

we should obtain as valid explanations:

- lock 1 was down and it has not been moved: $\overline{up(1)}_0 \wedge \neg toggle(1)_1$
- lock 1 was up but we have toggled it: $\overline{up(1)}_0 \wedge toggle(1)_1$.

Definition 24 (Planning) A *planning* problem is enunciated by providing a complete initial state σ_0 and a set of observations for the last state G_n called the *goals*, trying to obtain the complete sequence of compound actions that guarantees that the goals are reached. Formally, we want to find the sequence $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, called the *plan*, satisfying both:

$$atoms(\sigma_0) \cup atoms(\vec{\alpha}) \cup G_n \not\models_{\mathcal{B}} \perp \tag{4.2}$$

$$atoms(\sigma_0) \cup atoms(\vec{\alpha}) \models_{\mathcal{B}} G_n \tag{4.3}$$

□

Notice the difference between temporal explanation and planning. While the former just provides actions executions that are consistent with respect to our observations, the later requires that the actions provided as solution *guarantee* obtaining the desired goals G_n. Thus, in the general case, planning is stronger than temporal explanation.

As an example of the distinction, assume that a fire can be lighted either using a lighter or a pair of stones. The use of the stones is nondeterministic, so that they cause a spark that may light the fire or not. If we are said that in the resulting state the fire was observed, we would have two possible temporal explanations: either someone has used the lighter, or the stones have been successfully used. However, if the fire were a goal for planning, the last explanation is not a valid plan, since it does not guarantee that the goal is achieved. It can be easily seen that this distinction disappears when handling deterministic systems.

4.3 Automata and transition systems

As we have seen, the system behavior can be specified as a set of narratives which, in the general case, do not need to satisfy any particular temporal restriction. However, in most cases, the kind of systems we handle are more restrictive. For instance, sometimes, it is enough with describing σ_{i-1} and α_i for completely fixing the possible resulting state(s) σ_i. In other words, it is usual that successor states exclusively depend on the previous state and the action execution. The advantage of these systems is that, rather than describing the whole set of allowed narratives $\mathcal{B}$, we may provide instead the corresponding transition relation.

Definition 25 (Transition system) Given an action signature $\langle \mathcal{F}, \mathcal{A}, \mathcal{V}, range \rangle$, we define a *transition system* as a tuple $\langle S, A, \varrho \rangle$ where

- S is the set of *allowed fluent states*,
- A is the set of *allowed compound actions*,
- ϱ is the *transition relation*, $\varrho \subseteq S \times A \times S$.

□

In this way, the narratives in the behavior $\mathcal{B}$ are obtained from the finite sequences of consecutive applications of the transition relation:

$$\mathcal{B} = \{\nu \text{ of shape (4.1)} \mid \text{for all } i \in [1, n],\ \langle \sigma_{i-1}, \alpha_i, \sigma_i \rangle \in \varrho\}$$

Thus, a *transition* is simply seen as a narrative with $n = 1$.

It is also easy to define now the set of all possible *resulting* or *successor states* of applying α in a given state σ as:

$$Res(\sigma, \alpha) \stackrel{\text{def}}{=} \{\sigma' \mid \langle \sigma, \alpha, \sigma' \rangle \in \varrho\}$$

A compound action α is said to be *nonexecutable* in σ iff $|Res(\sigma, \alpha)| = 0$. Besides, a transition system is called *deterministic* when, for all $\sigma \in S$ and $\alpha \in A$, $|Res(\sigma, \alpha)| \leq 1$. In such a case, it is usual to talk about ϱ as a partial function, rather than a relation: $\sigma' = \varrho(\sigma, \alpha)$.

Transition systems are directly related to the well known structures of finite automata. An *automaton* is defined as a tuple $\langle S, A, \varrho, \sigma_0, F \rangle$ where A is called the *set of input symbols* and the two additional components correspond to:

1. the *initial state*, $\sigma_0 \in S$
2. the set of *acceptance states*, $F \subseteq S$

The typical use of automata is to decide the membership of a sequence of input symbols to a given *language* (set of sequences). In our case, the input symbols clearly correspond to the set of possible compound actions A. However, the idea of "language" is less intuitive: it should correspond to all the sequences of actions given *any* possible initial state $\sigma_0 \in S$ and accepting as final *any* allowed state, that is, assuming $F = S$. Thus, it does not really make sense to fix σ_0 and F.

For compactness sake, we will adopt some conventions for the automata representation. A set of arcs sharing the same source and target are abbreviated as a single arc with several labels, whereas a double-direction arc, is the abbreviation of the corresponding two single-direction arcs. Besides, when the automaton corresponds to a boolean domain (which will be in most of the cases) we will adopt an abbreviated notation for state and arc labels. States will be encoded as strings of truth values, whereas for compound actions, we will use strings containing the letter `p` to point out the action occurrence, and `n` to represent its nonoccurrence. The correspondence between values and fluents or actions is established by an ordering in an attached legend. For instance, given the legend:

$$toggle(1)\ toggle(2) : up(1)\ up(2)\ open$$

state `tft` corresponds to $\{up(1), \overline{up(1)}, open\}$ whereas compound action `pn` would be $\{toggle(1)\}$.

Automaton in figure 4.1 shows the transition system corresponding to the suitcase example. Together with the state string, we have also depicted the suitcase appearance.

As unique remark, notice that state `ttf` $= \{up(1), up(2), \overline{open}\}$ may seem counterintuitive, but is not necessarily impossible. We have understood that Lin's description of the suitcase mechanism just talks about the process for opening the suitcase. Thus, we leave open the possibility of this configuration, for instance, because the suitcase is manufactured with this shape. Of course, it could always be possible to define this state as non-allowed, deleting it from our representation.

4.4 Adding Pertinence

Let us see now how to modify transition systems to cope with the idea of pertinence. As we explained in the previous chapter, we will be interested in representing not only the fluent values, that is, the state σ, but also whether they have been affected by the performed actions α or not. To this aim, we need a new function to capture this information.

Definition 26 (Pertinence mapping) A *pertinence mapping*, is a function $\pi : \mathcal{F} \cup \mathcal{A} \longrightarrow \{\mathtt{p}, \mathtt{n}\}$ that associates a pertinence value to each symbol p. □

As happened with fluent states and compound actions, we will represent π as a set of facts. To this aim, we define a second type of fact:

Definition 27 (Pertinence Fact) Given a symbol p, a *pertinence fact* is a structure $pert(p, v)$, with $v \in \{\mathtt{p}, \mathtt{n}\}$. Since the set of values is fixed and binary, we will always allow the abbreviations $!p$ and $\overline{!p}$ for respectively denoting $pert(p, \mathtt{p})$ and $pert(p, \mathtt{n})$. □

An example of pertinence mapping could be:

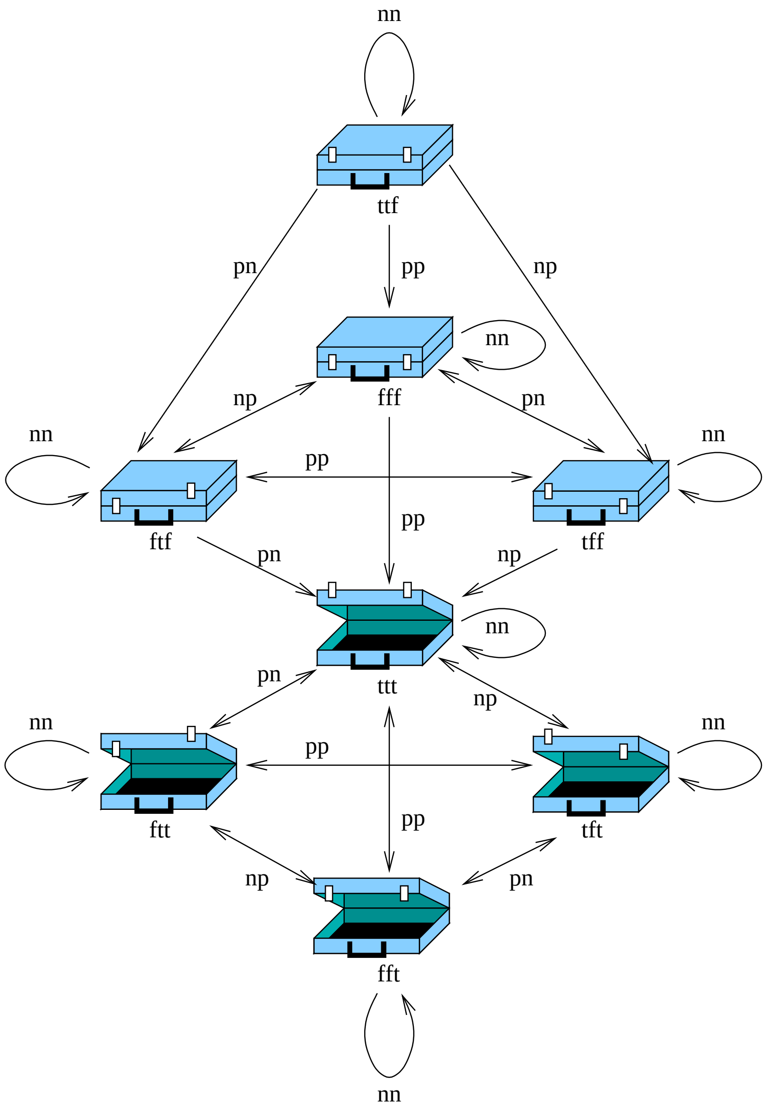
$$\{!toggle(1), \overline{!toggle(2)}, !up(1), \overline{!up(2)}, \overline{!open}\}$$

pointing out that action $toggle(1)$ has been performed and that the value for fluent $up(1)$ became affected by this action execution. Since $!p \in \pi$ stands for "pertinent with respect to the performed actions", any performed action is trivially pertinent.

Definition 28 (Pertinence narrative) A *pertinence narrative* ν is a finite sequence:

$$\sigma_0,\quad \alpha_1, \sigma_1, \pi_1,\quad \ldots,\quad \alpha_n, \sigma_n, \pi_n \tag{4.4}$$

with $n \geq 0$, that satisfies:



toggle(1) toggle(2) : up(1) up(2) open

Figure 4.1: Automaton for Lin's suitcase.

C1) The pertinence value for any action a at any situation $i \in [1, n]$ is fixed with respect to α_i:

$$\pi_i(a) = \begin{cases} \mathtt{p} & \text{if there is some } holds(a, v) \in \alpha_i \\ \mathtt{n} & \text{otherwise} \end{cases}$$

C2) As before, we consider an implicit $\alpha_0 = \emptyset$, adding also an implicit π_0 which contains nonpertinence for all the symbols, i.e., $\pi_0 = \{\overline{!p} : p \in \mathcal{A} \cup \mathcal{F}\}$.

C3) Given any $i \in [1, n]$, if a fluent f is nonpertinent, $\pi_i(f) = \mathtt{n}$, then $\sigma_i(f) = \sigma_{i-1}(f)$.

□

Condition C1 expresses that an action is pertinent iff it has been performed (regardless its value). As for C2, notice that, since no action is performed at 0, we consider that no symbol may become pertinent at that situation. Finally, condition C3 points out that any fluent f nonpertinent at i maintains its previous value at $i - 1$. In other words, a change of value is only accepted if f is pertinent (postulate P3).

In order to maintain the alternative representation of narratives as sets of atoms, we define the new type of atom:

Definition 29 (Pertinence atom) A *pertinence atom* is a structure $pert(p, v, i)$ where $pert(p, v)$ is a pertinence fact and i is any situation index $i \in [0, n]$. Abbreviations $!p_i$ and $\overline{!p}_i$ respectively denote $pert(p, \mathtt{p}, i)$ and $pert(p, \mathtt{n}, i)$. □

Note how we handle now atoms like $holds(p, v, i)$ and $pert(p, v, i)$, which can be seen as specializations of $holds(a)$ and $pert(a)$, we had used for translating L^2 theories into propositional logic. Analogously, we extend the notation $atoms(X)$ as follows:

$$\begin{aligned} atoms(\pi_i) &\stackrel{\text{def}}{=} \{pert(p, v, i) \mid pert(p, v) \in \pi_i\} \\ atoms(\nu) &\stackrel{\text{def}}{=} \bigcup_{i \in [0,n]} (atoms(\alpha_i) \cup atoms(\sigma_i) \cup atoms(\pi_i)) \end{aligned}$$

As an example, consider the extension of ν_1 so that we add pertinence atoms:

$$\begin{aligned} \nu_1' = \nu_1 \cup \{ \quad & \overline{!toggle(1)}_0, \overline{!toggle(2)}_0, \overline{!up(1)}_0, \overline{!up(2)}_0, \overline{!open}_0, \\ & \overline{!toggle(1)}_1, !toggle(2)_1, \overline{!up(1)}_1, !up(2)_1, \overline{!open}_1, \\ & !toggle(1)_2, !toggle(2)_2, !up(1)_2, !up(2)_2, \overline{!open}_2, \\ & \overline{!toggle(1)}_3, !toggle(2)_3, \overline{!up(1)}_3, !up(2)_3, !open_3 \ \} \end{aligned}$$

Pertinence narratives contain a considerable amount of redundant information due to conditions C1, C2 and C3. We will sometimes use a more concise sequential representation, just pointing out the initial fluent values and, for each transition, the values for pertinent symbols. For instance, narrative ν_1' would correspond to:

0) $\overline{up(1)}$, $\overline{up(2)}$, $\overline{open}$

1) $toggle(2)$, $up(2)$

2) $toggle(1)$, $toggle(2)$, $up(1)$, $\overline{up(2)}$

3) *toggle*(2), *up*(2), *open*

In this way, anything not explicitly asserted is assumed to be nonpertinent, and so, to have persisted. For instance, at situation 2, *open* has persisted false since situation 0, whereas at 3, *up*(1) has persisted true since situation 2. This representation is also more efficient for practical purposes, since it allows storing narratives in a more compact way. Besides, it emphasizes also an interesting representational feature: at each moment, we can easily establish the *most recent situation* in which the value of a fluent became pertinent. As shown in [22], this enables an easy introduction of temporal[4] constraints among the instants in which different fluents are caused.

To end up with the introduction of pertinence into transition systems, we must also extend the automaton representation. A simple way to do this could be to include the pertinence facts into each automaton state, together with the fluent values. Unfortunately, this would imply an explosion of the number of automaton states to be represented, which is already a problem when we just consider fluent values. However, we can avoid this problem because we will actually handle systems satisfying the restriction:

π_i is fixed by α_i and σ_{i-1}.

Since we will center the attention in transition systems satisfying this constraint, it will be possible to represent pertinence separatedly as an *output relation*, handling in this way *finite state machines*, rather than automata.

Definition 30 (Pertinence transition system) Given an action signature $\langle \mathcal{F}, \mathcal{A}, \mathcal{V}, range \rangle$, we define a *pertinence transition system* as a tuple $\langle S, A, P, \varrho, \omega \rangle$ where

1. S is the set of *allowed fluent states*,
2. A is the set of *allowed compound actions*,
3. P is the set of *allowed pertinence mappings*,
4. ϱ is the *transition relation*, $\varrho \subseteq S \times A \times S$.
5. ω is the *output relation*, $\omega \subseteq S \times A \times P$.

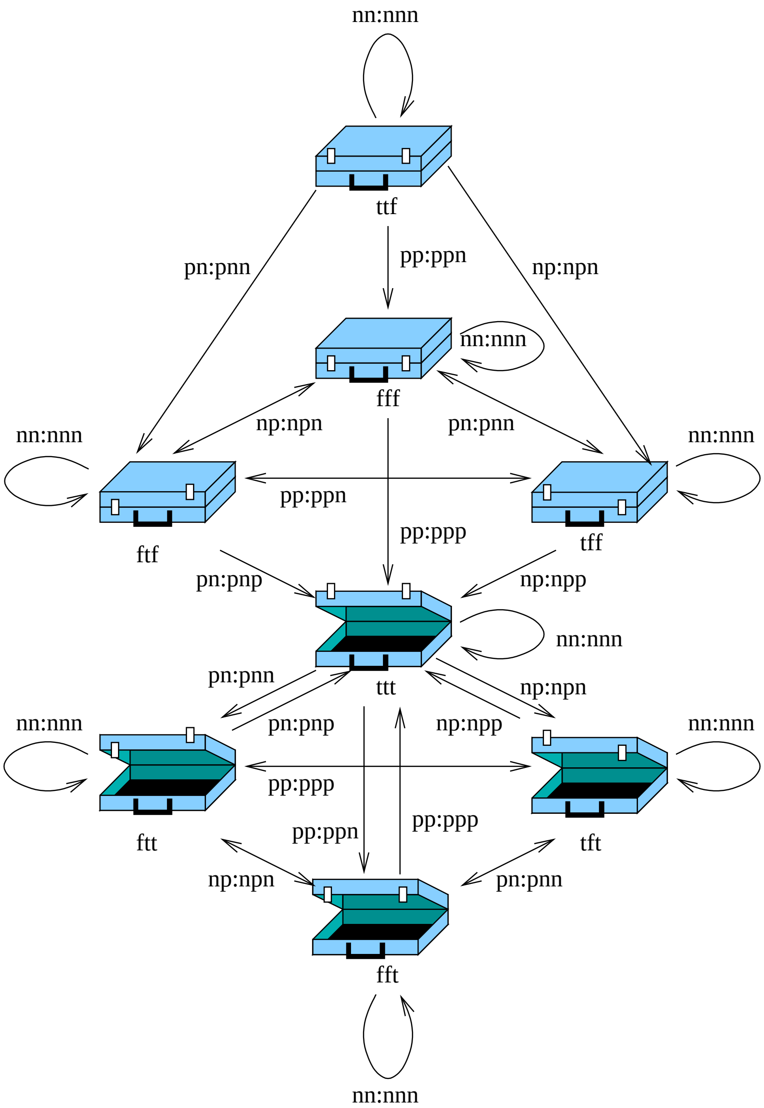
□

As happened with the transition function, ϱ, when the system is deterministic, the output relation will become a partial function: $\pi = \omega(\sigma, \alpha)$.

Finally, the introduction of pertinence information into the graphical representations of boolean domains is quite straightforward. In fact, we had already represented compound actions using their pertinence values. The only needed addition is the pertinence of the fluents, which can be simply appended to each arc label (they are separated from action values by a colon). For instance, the complete Lin's suitcase domain could finally become the machine in figure 4.2.

As remarkable differences with respect to figure 4.1, notice that some double-direction arcs have been unfolded, since they vary now in their outputs (they yield different pertinence mappings).

[4]Notice that although we use here a situation index, it is not so difficult to imagine an extension for a temporal basis using real numbers.



toggle(1) toggle(2) : up(1) up(2) open

Figure 4.2: Finite state machine for Lin's suitcase.

Chapter 5

$\mathcal{P}$-language

Using finite state machines as the "programmer-level" representation of the system behavior has some important drawbacks. The most evident one is the intractability in the number of possible states, as far as the system representation grows up. But, perhaps, the most cumbersome one is the absence of modularity. Even when dealing with small systems, any update of the representation, like adding more actions or fluents, cannot be implemented as a small addition or transformation but, instead, it forces us to reconsider the whole machine. As a counterpart, however, they also present some advantages. They provide a straightforward description of the system evolution: solving temporal projection, explanation or planning problems just amounts to a simple analysis of the machine graph. In some way, the machine compiles all the possible answers for all the possible queries of these three basic types of problems. However, we claim that the machine based description (or any other one just based on a set of narratives) does not contain enough information for a *complete understanding* of the system behavior.

For instance, should we consider that two systems described by the same machine are "equivalent"? With respect to temporal projection, explanation or planning queries, it is clear that the answer is affirmative. However, the two systems may handle different causal relations, although they finally lead to the same set of answers. Think about our machine in figure 4.2. By just looking at the graph, we could not clearly establish whether fluent *open* is an indirect effect of toggling the locks or not. Thus, we could think about an "equivalent" system in which $up(1)$, $up(2)$ and *open* were all direct effects, without any causal relation among them. Of course, one could object: why to bother about the internal causal relations, if we are just interested in obtaining the same results for the same queries? The answer is that the underlying causal dependences may influence the elaboration of our system representation. As an example, if we add a new action for moving the locks, the machine for representing the new system will be clearly different depending on whether *open* is affected by the locks or not. So, the two systems should not be considered completely equivalent, since under the same modification, they lead to different machines[1].

In this section we describe a description language consisting of elementary causal constructions that are very similar to those from the well known $\mathcal{A}$-*language* [41], but will be later characterized as simple cases of the L^2 conditional. The basic causal language introduced here allows us not only to describe in a more compact way the transition and output relations for the finite state machine describing the system behavior, but also to represent in an explicit way

[1] In fact, a similar discussion has recently arisen[64] in Logic Programming, where two programs may provide the same answers but react in a different way with respect to addition of rules. When the programs are also equivalent in this last sense, they are said to be *strongly equivalent*.

the causal dependences present in the domain.

We will initially handle an important restriction: the set of rules will not contain cyclic references. Using this assumption, we provide a (deterministic) operational semantics consisting on a constructive algorithm for computing the successor state and the pertinence output. It is interesting to note that all the proposed formalizations we introduce in the next chapter actually coincide with the operational semantics when there are no cyclic references, leading in all cases to a deterministic system. Some of these formalizations, however, will use nondeterminism as a way of interpreting causal cycles.

5.1 $\mathcal{P}$-rule syntax

A $\mathcal{P}$*-rule* has the general shape:

$$E \textbf{ if } C \textbf{ after } D \tag{5.1}$$

where:

1. E is called the *effect* and corresponds to a fluent value fact $holds(f, v)$ or the constant $\perp$,
2. C is the *condition* and has the shape $C_1 \wedge \cdots \wedge C_k$, $k > 0$, where the C_i are facts of any kind (like $holds(p, v)$ or $pert(p, v)$),
3. D is the *(effect) precondition* and has the shape $D_1 \wedge \cdots \wedge D_m$, $m \geq 0$, where each D_i is a fluent value fact $holds(f, v)$.

With an empty effect precondition, $m = 0$, we simply write E **if** C. Besides, we sometimes use the abbreviation:

$$\{E_1, \ldots, E_k\} \textbf{ if } C \textbf{ after } D$$

to stand for the set of rules:

$$\begin{array}{ll} E_1 & \textbf{if } C \textbf{ after } D \\ & \vdots \\ E_k & \textbf{if } C \textbf{ after } D \end{array}$$

Intuitively, the rule is applicable when, being the precondition D true, the condition C is both true and pertinent. In its turn, once it is applied, it will make the effect E both true and pertinent.

As a pair of examples, the Yale shooting scenario can be represented by[2] $\mathcal{A}_y = \{load, shoot\}$, $\mathcal{F}_y = \{alive, loaded\}$ and the set R_y of rules:

$$\begin{array}{rll} loaded & \textbf{if} & load \\ \{\overline{alive}, \overline{loaded}\} & \textbf{if} & shoot \textbf{ after } loaded \end{array} \tag{5.2, 5.3}$$

[2]As we allow empty compound actions, action *wait* is not needed.

whereas the suitcase example can be represented by $\mathcal{A}_s = \{toggle(1), toggle(2)\}$, $F_s = \{up(1), up(2), open\}$ and the set of rules R_s:

$$\begin{array}{rll} up(1) & \textbf{if} & toggle(1) \textbf{ after } \overline{up(1)} \\ \overline{up(1)} & \textbf{if} & toggle(1) \textbf{ after } up(1) \\ up(2) & \textbf{if} & toggle(2) \textbf{ after } \overline{up(2)} \\ \overline{up(2)} & \textbf{if} & toggle(2) \textbf{ after } up(2) \\ open & \textbf{if} & up(1) \wedge up(2) \end{array} \quad \begin{array}{r} (5.4) \\ (5.5) \\ (5.6) \\ (5.7) \\ (5.8) \end{array}$$

Let F be either a conjunction of facts $\phi_1 \wedge \cdots \wedge \phi_k$ or a set of facts $\{\phi_1, \ldots, \phi_k\}$. Then, we denote:

$$symb(F) = \{symb(\phi_i) \mid i \in [1, k]\}$$

A rule that exclusively contains actions in the condition, $symb(C) \subseteq \mathcal{A}$, is called an *effect axiom* (for instance, rules (5.2)-(5.3) and (5.4)-(5.7)), whereas if it exclusively contains fluent symbols ($symb(C) \subseteq \mathcal{F}$) we call it *ramification rule* (e.g., rule (5.8)).

An effect axiom with $E = \bot$ receives the name of *qualification constraint*. Intuitively, its condition and precondition point out a configuration *not allowed to happen*, leading to the non existence of successor state, and so, to the nonexecutability of the performed action – the action is *disqualified*. To understand its utility, consider the following modification of the Yale shooting scenario, R'_y:

$$\begin{array}{rll} loaded & \textbf{if} & load \\ \{\overline{alive}, \overline{loaded}\} & \textbf{if} & shoot \\ \bot & \textbf{if} & shoot \textbf{ after } \overline{loaded} \end{array} \quad \begin{array}{r} (5.9) \\ (5.10) \\ (5.11) \end{array}$$

The effect precondition has disappeared from rule (5.3) to become, negated, a disqualification in (5.11). In this way, while R_y allowed a shot with an unloaded gun, leading to no resulting effects, with R'_y such a shot *cannot be performed* (there does not exist any successor state).

As another example of use of qualification constraints, we can also consider the encoding of STRIPS [34] operators, frequently handled in planning problems. A STRIPS operator a has an associated formula Pre called the precondition, plus two lists of facts: the add list $f_1, \ldots, f_k$ and the delete list $g_1, \ldots, g_m$. This can be directly represented as:

$$\begin{array}{rll} \{f_1, \ldots, f_k, \overline{g_1}, \ldots, \overline{g_m}\} & \textbf{if} & a \\ \bot & \textbf{if} & a \textbf{ after } Pre_1 \\ & \ldots & \\ \bot & \textbf{if} & a \textbf{ after } Pre_r \end{array} \quad \begin{array}{r} (5.12) \\ (5.13) \\ (5.14) \\ (5.15) \end{array}$$

where $Pre_1, \ldots, Pre_r$ is the set of conjunctive clauses obtained from the Disjunctive Normal Form of $\neg Pre$.

Analogously to effect axioms, a ramification rule with $E = \bot$ plays a similar role to a state constraint, ruling out some undesired configurations. However, it does not impose any restriction on the initial situation, since at that moment, no symbol is pertinent, and no rule would be applicable. To allow constraints at situation 0, we define an *initial constraint* as any propositional combination of fluent facts. A state σ_0 satisfies an initial constraint ϕ, denoted as $\sigma_0 \models \phi$ iff:

1. $\sigma_0 \models holds(f,v)$ iff $holds(f,v) \in \sigma_0$

2. $\sigma_0 \models \phi \vee \psi$ iff $\sigma_0 \models \psi$ or $\sigma_0 \models \psi$

3. $\sigma_0 \models \neg\phi$ iff $\sigma_0 \not\models \psi$

and the rest of propositional connectives are defined in terms of $\vee$ and $\neg$. Given any state constraint ϕ, we require that $\sigma_0 \models \phi$ for any allowed state $\sigma_0 \in S$.

To see the need for initial constraints, consider for instance the suitcase domain represented as R_s. If we do not add the constraint:

$$up(1) \wedge up(2) \supset open$$

then we would be considering as a possible initial situation one in which the suitcase has both locks up but it is closed.

Of course, we could impose that rules are interpreted as constraints in the initial situation. However, it is interesting to maintain both things separatedly[3]. For instance, we may actually interpret Lin's enunciate as a description on *how the suitcase can be opened* instead of *when it happens to be open*. In such a case, we could allow an initial state that does not satisfy the constraint (for instance, imagine that the suitcase is manufactured closed but with both locks up).

Given any domain represented by the set of rules R, we will also add the syntactic restriction of absence of cyclic references. The study of cycles is postponed until chapter 7.

Definition 31 (Direct dependence) Given a set of $\mathcal{P}$-rules R, we say that a fluent f *directly depends* on a symbol p (fluent or action) iff there exists a rule (5.1) in R with $symb(E) = f$ and $p \in symb(C)$. □

Definition 32 (Acyclic rules) A set of $\mathcal{P}$-rules R is *acyclic* iff we can establish a nonnegative integer mapping, $layer : \mathcal{A} \cup \mathcal{F} \longrightarrow \mathbb{N} \cup \{0\}$ such that, if a fluent f directly depends on a symbol p, then $layer(f) > layer(p)$. □

Notice the similarity between this *layer* function and the *level* function we defined for hierarchical logic programs. There may exist several possible *layer* functions satisfying this condition. We will be particularly interested in the following one:

1. $layer(a) = 0$ for any action a

2. $layer(f) = 0$ iff there exists no rule (5.1) in R with $symb(E) = f$.

3. $layer(f) = max\{layer(C) \mid (E \textbf{ if } C \textbf{ after } D) \in R, symb(E) = f\} + 1$

where the application of *layer* to a condition $C = C_1 \wedge \cdots \wedge C_k$ is defined as:

$$layer(C) \stackrel{\text{def}}{=} max_{i\in[1,k]}\{layer(symb(C_i))\}$$

Another way of establishing the acyclicity of a set of rules R is representing its corresponding *dependences graph*, a directed graph where each node f represents a fluent and each edge $f \to f'$ means that f' directly depends on f. As expected, a set of rules will be acyclic (under our

[3]The separation between constraints and causal rules is extensively discussed in [28].

previous definition) iff its corresponding graph is free of cycles. For instance, given the set of rules:

$$f \quad \textbf{if} \quad g \wedge h \tag{5.16}$$
$$\overline{f} \quad \textbf{if} \quad \overline{g} \tag{5.17}$$
$$i \quad \textbf{if} \quad h \wedge \overline{f} \tag{5.18}$$
$$j \quad \textbf{if} \quad \overline{h} \wedge g \wedge f \wedge i \tag{5.19}$$

the corresponding directed acyclic graph is represented in figure 5.1, where the vertical lines show the layer assigned to each fluent.

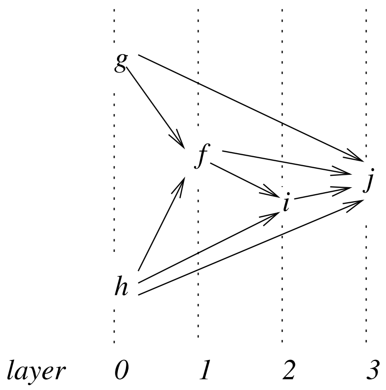


Figure 5.1: Dependences graph for rules (5.16)-(5.19).

5.2 Operational semantics

We want to establish now how to obtain the transition and output relations, ϱ and ω from the set of rules R. In fact, our transition system will be deterministic by now, and so, we want to compute the successor state (if there exists so) $\sigma_1 = \varrho(\sigma_0, \alpha_1)$ and pertinence mapping $\pi_1 = \omega(\sigma_0, \alpha_1)$, given any pair σ_0 and α_1.

The first intuitive idea is that rules whose conditions are true should be "applied," so that we include their effects in σ_1, and mark the affected fluents as pertinent in π_1. The problem is how to know when a rule condition C "is true," since the facts in C, in their turn, may be effects from other rules, or old values persisting from the predecessor state. Thanks to our acyclicity assumption, we can directly fix an ordering in this process of rule application by following the layer mapping.

We will construct an iterative algorithm that goes completing the transition, adding facts to σ_1 and π_1 layer by layer, until they are completely defined for all the fluents. Thus, at each moment, we will actually handle a partial transition.

Definition 33 (Partial transition of layer j) Let j be a nonnegative integer. We define a *partial transition* ν^j as a narrative of shape:

$$\sigma_0, \alpha_1, \sigma_1^j, \pi_1^j$$

where σ_1^j and π_1^j are complete for symbols up to layer j, but undefined for symbols of greater layers. □

The sets α_0 and π_0 are not represented, since they are not relevant for establishing the final complete transition. As a remark, note that π_1^j is only partial for fluents, since by condition C1 of definition 28, the pertinence mapping for actions is always complete.

The addition of new facts to σ_1^j and π_1^j will depend on the applicability of rules, which is defined as:

Definition 34 (Rule applicability) Given a layer number $j \geq 0$, a rule of shape (5.1) with $layer(symb(E)) = j$ is *applicable* in a partial transition $\nu^j = \sigma_0, \alpha_1, \sigma_1^j, \pi_1^j$ iff:

1. $\phi \in \sigma_0$, for all fact ϕ in D,
2. $\phi \in \sigma_1^j \cup \pi_1^j \cup \alpha_1$, for all fact ϕ in C,
3. $!symb(\phi) \in \pi_1^j$, for some fact ϕ in C

□

Note that condition 3 says that some symbol occurring in the condition C must be known to be pertinent. This actually corresponds to the practical implementation of postulate P5.

Given the state σ_0 and the compound action α_1, the algorithm in figure 5.2 describes how to compute the successor state σ_1 and pertinence mapping π_1 (i.e., how to complete the transition ν).

S0) $j := 0$
$Out^0 = \emptyset$
$Pers^0 = \{holds(f, v) \in \sigma_0 \mid layer(f) = 0\}$
$\sigma_1^0 = Pers^0$
$\pi_1^0 = \{!a \mid a \in symb(\alpha_1)\} \cup \{\overline{!a} \mid a \notin symb(\alpha_1)\} \cup \{\overline{!(f)} \mid layer(f) = 0\}$

S1) We define $\nu^j = \sigma_0, \alpha_1, \sigma_1^j, \pi_1^j$
If j is the maximum layer in $\mathcal{F}$ then finish with $\sigma_1 = \sigma_1^j$ and $\pi_1 = \pi_1^j$.

S2) $j := j + 1$
$Out^j = \{E \mid E$ is effect of some applicable rule in ν^{j-1} and $layer(symb(E)) = j\}$
If Out^j is inconsistent or contains $\bot$, then finish: α_1 is not executable in σ_0.

S3) Compute:
$Pers^j = \{holds(f, v) \in \sigma_0 \mid layer(f) = j, f \notin symb(Out^j)\}$
$\sigma_1^j = \sigma_1^{j-1} \cup Out^j \cup Pers^j$
$\pi_1^j = \pi_1^{j-1} \cup \{!p \mid p \in symb(Out^j)\} \cup \{\overline{!p} \mid p \in symb(Pers^j)\}$
and go to step S1.

Figure 5.2: Algorithm for computing the successor state for acyclic rules.

For each layer j, the algorithm computes the symbol values using two disjunct sets of facts: Out^j and $Pers^j$. The intuitive meaning of these sets is to point out the source for each fluent value: Out^j collects the effects of the applicable rules, whereas facts in $Pers^j$ are directly retrieved from the previous state σ_0 by persistence.

Proposition 1 *Under the operational semantics based on algorithm in figure 5.2, when the applied action is empty,* $\alpha_1 = \emptyset$*, the transition relation* ϱ *is reflexive and the output assigns nonpertinence to all the symbols. In other words: for all state* σ_0*,* $\langle \sigma_0, \emptyset, \sigma_0 \rangle \in \varrho$ *and* $\langle \sigma_0, \emptyset, \{\overline{!p} \mid p \in \mathcal{A} \cup \mathcal{F}\} \rangle \in \omega$*.*

Proof

It is straightforward since, pertinence of a fluent comes from a chaining of rule applications that must be originated by a performed action. When there are not performed actions, all the fluents are nonpertinent and so all the facts are obtained from $Pers^j$, which implies that the final $\sigma_1 = \sigma_0$. □

5.3 Examples

Let us first consider the suitcase problem in its original shape: we have $\sigma_0 = \{ovup(1), up(2), \overline{open}\}$ and we want to predict what happens after performing $\alpha = \{toggle(1)\}$ using the set of rules (5.4)-(5.8). Given $layer(up(1)) = layer(up(2)) = 1$ and $layer(open) = 2$, the next table shows the successive steps followed by the algorithm:

$$\begin{array}{ll} Out^0 = \emptyset & Pers^0 = \emptyset \\ Out^1 = \{up(1)\} & Pers^1 = \{up(2)\} \\ Out^2 = \{open\} & Pers^2 = \emptyset \end{array}$$

It can be easily seen that the final σ_1 and π_1 simply correspond to:

$$\begin{array}{lcl} \sigma_1 & = & \sigma_1^2 = \{up(1), up(2), open\} \\ \pi_1 & = & \pi_1^2 = \{!toggle(1), \overline{!toggle(2)}, !up(1), \overline{!up(2)}, !open\} \end{array}$$

Repeating this process for all the possible states and compound actions in this domain, we get the machine in figure 4.2.

To illustrate how the algorithm can be used for computing a narrative with several transitions, consider the typical Yale shooting problem, which consists of a sequence of loading, waiting and shooting:

$$\begin{array}{lcl} \sigma_0 & = & \{alive, \overline{loaded}\} \\ \alpha_1 & = & \{load\} \\ \alpha_2 & = & \emptyset \\ \alpha_3 & = & \{shoot\} \end{array}$$

We may apply the algorithm so that rather than considering transition $\alpha_0, \alpha_1, \sigma_1, \pi_1$ we handle instead $\sigma_i, \alpha_{i+1}, \sigma_{i+1}, \pi_{i+1}$, with $i \in [0, 2]$:

i=0) The first transition is computed as follows:

$$\begin{array}{ll} Out^0 = \emptyset & Pers^0 = \emptyset \\ Out^1 = \{loaded\} & Pers^1 = \{alive\} \end{array}$$

and so $\sigma_1 = \{loaded, alive\}$, $\pi_1 = \{!load, \overline{!shoot}, !loaded, \overline{!alive}\}$.

i=1) By proposition 1, the application of the empty action $\alpha_2 = \emptyset$ leads to the same state, $\sigma_2 = \sigma_1$.

i=2) Finally, the last transition leads to:

$$Out_0 = \emptyset \qquad Pers_0 = \emptyset$$

$$Out_1 = \{\overline{alive}, \overline{loaded}\} \quad Pers_1 = \emptyset$$

Note that, when $i = 2$, rule (5.3) becomes applicable because the condition is true and pertinent (action *shoot* has been performed) and the precondition is true ($loaded \in \sigma_0$). The final state is then $\sigma_3 = \{\overline{loaded}, \overline{alive}\}$ and $\pi_3 = \{!shoot, \overline{!load}, !alive, !loaded\}$.

The whole obtained narrative can be summarized in the already seen format:

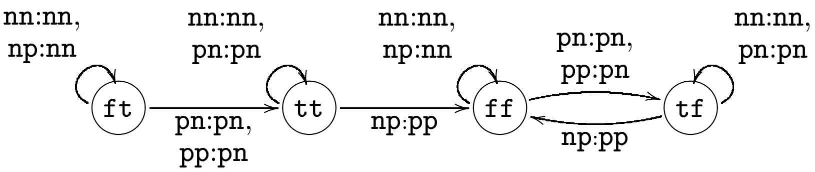
0) $alive, \overline{loaded}$

1) $load, loaded$

2)

3) $shoot, \overline{alive}, \overline{loaded}$

so that only pertinent facts are shown at each transition. In figure 5.3, we have represented the machine that corresponds to the complete transition relation for rules (5.2)-(5.3).



load shoot : *loaded alive*

Figure 5.3: Machine for the Yale Shooting scenario.

Finally, we will see one more example, as introduced in [86], for showing the behavior of pertinence in the condition of a rule.

Example 5 *(***The gong example)** A ballerina must start her choreography when the *gong* is struck. If the choreography is particularly short, she may even finish while the gong is still sounding. □

Let us define the sets of actions $\mathcal{A}_g = \{strike, finish\}$, fluents $\mathcal{F}_g = \{dancing, gong\}$ and rules, R_g:

$$gong \quad \textbf{if} \quad strike \tag{5.20}$$
$$dancing \quad \textbf{if} \quad gong \tag{5.21}$$
$$\overline{dancing} \quad \textbf{if} \quad finish \tag{5.22}$$

Assume that we perform action *finish*, but the gong is still sounding. Apparently, this would lead to an inconsistence, obtaining both *dancing* and $\overline{dancing}$. However, as condition of (5.21) is not pertinent (the gong remains sounding by inertia), the rule is not applicable. Intuitively, the ballerina relies on the gong stroke to begin her choreography, but not on the persistent sound afterwards.

Looking at the rules, fluent *gong* would be in layer 1 and fluent *dancing* in layer 2. Let $\sigma = \{dancing, gong\}$ and $\alpha = \{finish\}$. The successor state is computed as follows:

$$
\begin{array}{ll}
Out_0 = \emptyset & Pers_0 = \emptyset \\[1em]
Out_1 = \emptyset & Pers_1 = \{gong\} \\[1em]
Out_2 = \{\overline{dancing}\} & Pers_2 = \emptyset
\end{array}
$$

and so:

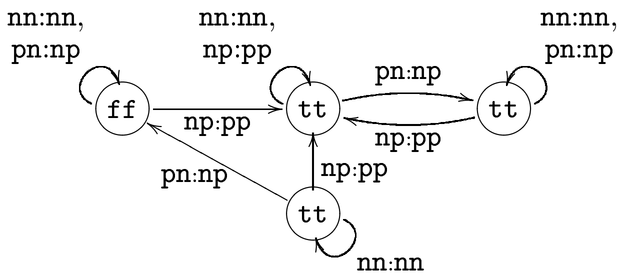
$$
\begin{array}{rcl}
\sigma_2 & = & \{gong, \overline{dancing}\} \\
\pi_2 & = & \{!finish, \overline{!strike}, \overline{!gong}, !dancing\}
\end{array}
$$

The complete transition and output relations is depicted in figure 5.4.

Of course, we could formulate the example by replacing rule (5.21) by an effect axiom:

$$dancing \quad \textbf{if} \quad gong$$

However, we could also easily imagine that there may exist different actions for making the gong sound, which is the actual cause to start dancing. In this way, when we use pertinence, we have somehow available a “dual” behavior for each fluent. On the one hand, the value of the fluent can be used as the inertial feature, like in this example, where the truth of *gong* points out that the gong remains sounding. On the other hand, the pertinence of the fluent behaves as an associated “non-intertial” fluent which represents some punctual event, like in the example, where the pertinence of *gong* points out that a new gong “note” (in musical terminology) has been initiated.



finish strike : *gong dancing*

Figure 5.4: Machine for the gong example.

Chapter 6

Pertinence Calculus: formalizing pertinence in action domains

Although the operational behavior for $\mathcal{P}$-rules allows solving the frame and ramification problems in a simple and natural way, we have not provided yet a real logical characterization of this behavior. To this aim, we may simply consider a propositional signature where we deal with the two types of atoms, $holds(p, v, i)$ and $pert(p, v, i)$, we had already defined for representing narratives. For commodity sake, however, we will allow a first order representation with quantification for fluents, actions or situations. We have called *pertinence calculus* to the resulting framework, because of its similarity to other well known action formalisms like Situation Calculus [81] or Event Calculus [53], which also use a predicate "*holds*" reifying the fluent name.

Pertinence calculus will be used as the monotonic framework on which to apply an additional nonmonotonic technique for capturing the inertia default. We propose different possibilities for representing inertia, using first well-known nonmonotonic techniques like circumscription and default logic, that are applicable to first order logic, and proposing afterwards a logic programming encoding (which relies on a previous theory grounding) interpreted under three different semantics: stable models, well founded semantics and completion. In all the cases, we obtain the correspondence with respect to the operational semantics seen in the previous chapter, although in the case of circumscription only after requiring an additional syntactic restriction.

6.1 Basic definitions and axioms

Pertinence calculus can be presented as a many-sorted first order logic axiomatization with four finite sorts: $\mathcal{A}$ (action names), $\mathcal{F}$ (fluent names), $\mathcal{V}$ (values), and the set of *situations*[1], nonnegative integers ranging in $[0, n]$. As before, we simply call *symbol* to any element of the set $\mathcal{A} \cup \mathcal{F}$, and for each symbol p, we have a subset of $\mathcal{V}$ called $range(p)$. Letters I, F, A, V and P will be used for situation, fluent, action, value and symbol variables respectively, whereas the corresponding lower case letters will be used for denoting constants. Pertinence calculus only handles two basic 3-ary predicates, the already mentioned $holds(p, v, i)$ and $pert(p, v, i)$, for which the following axioms are defined.

[1] Although this work is oriented to narrative domains, a simple variation of pertinence calculus can be directly proposed to deal with a branching structure of situations $do(A, S)$ as those used in Situation Calculus.

We include, for each sort, a pair of axioms called *domain closure* (DC) and *unique names assumption* (UNA). For instance, in the case of the actions sort, we would have:

$$\forall.\ A \bigvee_{a\in\mathcal{A}} A = a \tag{DC}$$

$$\bigwedge_{a,a'\in\mathcal{A},\ a\neq a'} \neg(a = a') \tag{UNA}$$

Since we will not handle structured terms nor function names, these axioms allow establishing a one-to-one correspondence between (syntactic) constant names and the possible (semantic) objects in the universe of any first order interpretation M. Thus, we will directly use $M[c] = c$, for any constant name $c \in \mathcal{A} \cup \mathcal{F} \cup \mathcal{V} \cup [0, n]$.

The following axioms respectively guarantee, at each situation, the uniqueness of value for each symbol and its existence (only for fluents[2]).

$$\forall P, I, V, V'.\ \big(V \neq V' \supset \neg(holds(P, V, I) \wedge holds(P, V', I))\big) \tag{6.1}$$

$$\forall F, I.\ \exists V.\ holds(F, V, I) \tag{6.2}$$

Analogously, uniqueness and existence of pertinence value are expressed as:

$$\forall P, I.\ \neg\big(pert(P, \mathtt{p}, I) \wedge pert(P, \mathtt{n}, I)\big) \tag{6.3}$$

$$\forall P, I.\ \big(pert(P, \mathtt{p}, I) \vee pert(P, \mathtt{n}, I)\big) \tag{6.4}$$

which, in fact, can be reexpressed altogether as:

$$\forall P, I.\ \big(pert(P, \mathtt{n}, I) \equiv \neg pert(P, \mathtt{p}, I)\big)$$

This formula clearly points out that reifying the pertinence value in predicate *pert* is not actually needed, although it will be later very convenient for implementing causal rules and inertia using logic programs. In this way, while we maintain the representation inside first order logic, we will sometimes allow the alternative representation of *pert* as a binary predicate $pert(P, I)$, defining the following straightforward correspondence:

$$\begin{aligned} pert(P, \mathtt{p}, I) &\stackrel{\text{def}}{=} pert(P, I) \\ pert(P, \mathtt{n}, I) &\stackrel{\text{def}}{=} \neg pert(P, I) \end{aligned}$$

The conditions C1, C2 and C3 we had imposed on narratives respectively become now the axioms:

$$\forall A, I.\ \big(pert(A, \mathtt{p}, I) \equiv \exists V.\ holds(A, V, I)\big) \tag{6.5}$$

$$\forall P.\ pert(P, \mathtt{n}, 0) \tag{6.6}$$

$$\forall F, V, I.\ \big(\ I > 0\ \wedge pert(F, \mathtt{n}, I) \wedge holds(F, V, I-1) \supset holds(F, V, I)\big) \tag{UFR}$$

[2]Remember that an action has not any associated value when it is not performed

The formula (UFR) represents the *universal frame axiom*, asserting that the value of any non-pertinent fluent must persist. We call AX to the set of axioms (DC), (UNA), (6.1)-(6.6), without including the universal frame axiom, (UFR).

The usual shape the universal frame axiom in other approaches is closer to:

$$\forall F, V, I.\ \big(\ I > 0\ \wedge pert(F, \mathtt{n}, I) \supset \big(holds(F, V, I-1) \equiv holds(F, V, I)\big)\big) \tag{6.7}$$

The following theorem shows that under pertinence calculus axiomatization, it is indeed equivalent to (UFR).

Lemma 2 *If axioms (6.1) and (6.2) are satisfied, then the following formulas are equivalent:*

$$\forall F, V, I, J.\ \big(holds(F, V, I) \supset holds(F, V, J)\big) \tag{6.8}$$
$$\forall F, V, I, J.\ \big(holds(F, V, I) \equiv holds(F, V, J)\big) \tag{6.9}$$

Proof
(See appendix A). □

Theorem 6 *If axioms (6.1) and (6.2) are satisfied, then the formula (UFR) is equivalent to (6.7).*
Proof
It is straightforward by observing that (UFR) can be reexpressed as:

$$\forall F, V, I.\ \big(\ I > 0\ \wedge pert(F, \mathtt{n}, I) \supset \big(holds(F, V, I-1) \supset holds(F, V, I)\big)\big)$$

and then replacing the inner implication by an equivalence, as a result of applying lemma 2. □

6.2 Propositional representation

This first order representation can be alternatively converted into a propositional one. Thanks to finiteness of all sorts and axioms (DC) and (UNA), we can consider each possible first order model M as a propositional one, using as signature the finite set of ground Herbrand atoms for *holds* and *pert*. In this way, any universally (resp. existentially) quantified sort variable can be replaced by a finite conjunction (resp. disjunction). Therefore, we can simultaneously see the previous axioms (6.1)-(6.6), (UFR) as the respective propositional patterns:

$$\neg holds(P, V, I) \wedge holds(P, V', I) \tag{6.10}$$
$$\bigvee_{V \in range(F)} holds(F, V, I) \tag{6.11}$$
$$pert(A, \mathtt{p}, I) \equiv \bigvee_{V \in range(A)} holds(A, V, I) \tag{6.12}$$
$$pert(P, \mathtt{n}, 0) \tag{6.13}$$
$$pert(F, \mathtt{n}, I') \wedge holds(F, V, I'-1) \supset holds(F, V, I') \tag{6.14}$$

where all the variables are replaced by their possible ground instances, with $V \neq V'$, $I \in [0, n]$ and $I' \in [1, n]$.

6.3 High level constructions

All the L^2 constructions can be incorporated into pertinence calculus in a very simple way, just understanding them as abbreviations. These abbreviations can be "unfolded" using the already seen rules (3.13)-(3.23) for nonatomic formulas, plus the transformations:

$$!\,holds(p,v,i) \longrightarrow pert(p,\mathtt{p},i) \tag{6.15}$$
$$!\,pert(p,v,i) \longrightarrow pert(p,\mathtt{p},i) \tag{6.16}$$

We also introduce here a new notation which will be interesting for the comparison to other causal approaches. We define the predicate $caused(p,v,i)$ to stand for $\mathbb{C}holds(p,v,i)$, that is:

$$caused(p,v,i) \stackrel{\text{def}}{=} holds(p,v,i) \wedge pert(p,\mathtt{p},i) \tag{6.17}$$

To see how to unfold high level constructions, let us consider:

$$\mathbb{C}\,(holds(up(1),\mathtt{t},I) \wedge holds(up(2),\mathtt{t},I))$$

This is successively transformed into:

$$\begin{array}{rcl} (holds(up(1),\mathtt{t},I) \wedge holds(up(2),\mathtt{t},I)) & \wedge & !(holds(up(1),\mathtt{t},I) \wedge holds(up(2),\mathtt{t},I)) \\ holds(up(1),\mathtt{t},I) \wedge holds(up(2),\mathtt{t},I) & \wedge & (!holds(up(1),\mathtt{t},I) \vee !holds(up(2),\mathtt{t},I)) \\ holds(up(1),\mathtt{t},I) \wedge holds(up(2),\mathtt{t},I) & \wedge & (pert(up(1),\mathtt{p},I) \vee pert(up(2),\mathtt{p},I)) \end{array}$$

Using these L^2 constructions we can now provide a translation of any $\mathcal{P}$-rule $r = (E$ **if** C **after** $D)$ into pertinence calculus, following the next steps. Let I be a fresh situation variable. We define the formulas:

1. C_I as C after replacing facts $holds(p,v)$ by $holds(p,v,I)$ and pertinence facts $pert(p,v)$ by $pert(p,v,I)$,
2. E_I as E after replacing fact $holds(f,v)$ by $holds(f,v,I)$
3. D_{I-1} is D after replacing facts $holds(f,v)$ by $holds(f,v,I-1)$, or $\top$, if D is empty.

Then, the translation of r, denoted as $t(r)$, is defined as the formula:

$$\forall I. \quad \big(\, I>0 \;\wedge D_{I-1} \supset (E_I \Leftarrow C_I)\big)$$

Notice that the situation variable I is quantified for all the situations excepting 0. There exist two reasons for this. First, when D is not empty, situation $I-1$ must be defined, and so $I > 0$. Second, at situation 0 everything is nonpertinent (axiom (6.6)) and so, any rule would never be applicable. When there is no confusion, we will usually omit the quantification for I in rule translations, directly writing:

$$D_{I-1} \supset (E_I \Leftarrow C_I)$$

or, what is equivalent:

$$D_{I-1} \wedge \mathbb{C}C_I \supset \mathbb{C}E_I \tag{6.18}$$

Although in this expression, (6.18), we have used material implication $\supset$, we will bear in mind that it may be replaced by an inference rule (in the case of default logic) or by a program rule conditional, in the case of logic programming.

As for the translation itself, it is interesting to note how D is simply required to be true at situation $I-1$, while condition C is required to be *caused* at situation I (i.e., not only to hold but also to be pertinent). In the same way, the resulting effect must also be caused at I. This captures the effect of "pertinence propagation" explained in the operational semantics.

As an example, consider the translation of rule (5.8):

$$open \quad \textbf{if} \quad up(1) \wedge up(2)$$

from the suitcase domain, R_s. First, we would have:

$$\begin{aligned} C_I &\stackrel{\text{def}}{=} holds(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I) \\ D_{I-1} &\stackrel{\text{def}}{=} \top \\ E_I &\stackrel{\text{def}}{=} holds(open, \mathtt{t}, I) \end{aligned}$$

(remember that in the rule, we used abbreviation f to denote $holds(f, \mathtt{t})$). Then, the translation of (5.8) would be:

$$\top \wedge \mathbb{C}(holds(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I)) \supset \mathbb{C}holds(open, \mathtt{t}, I)$$

which, after applying the transformation rules, finally becomes:

$$\begin{aligned} &holds(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I) \wedge (pert(up(1), \mathtt{p}, I) \vee pert(up(2), \mathtt{p}, I)) \\ &\qquad \supset holds(open, \mathtt{t}, I) \wedge pert(open, \mathtt{p}, I) \end{aligned} \tag{6.19}$$

It is easy to see that, using the *caused* notation, this rule can be also expressed as the conjunction of the formulas:

$$caused(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I) \supset caused(open, \mathtt{t}, I) \tag{6.20}$$

$$holds(up(1), \mathtt{t}, I) \wedge caused(up(2), \mathtt{t}, I) \supset caused(open, \mathtt{t}, I) \tag{6.21}$$

As another example, rule (5.4) would be successively transformed into:

$$holds(up(1), \mathtt{f}, I-1) \wedge \mathbb{C}holds(toggle(1), \mathtt{t}, I) \supset \mathbb{C}holds(up(1), \mathtt{t}, I) \tag{6.22}$$

$$\begin{aligned} &holds(up(1), \mathtt{f}, I-1) \wedge holds(toggle(1), \mathtt{t}, I) \wedge !holds(toggle(1), \mathtt{t}, I) \supset \\ &\qquad holds(up(1), \mathtt{t}, I) \wedge !holds(up(1), \mathtt{t}, I) \end{aligned} \tag{6.23}$$

$$\begin{aligned} &holds(up(1), \mathtt{f}, I-1) \wedge holds(toggle(1), \mathtt{t}, I) \wedge pert(toggle(1), \mathtt{p}, I) \supset \\ &\qquad holds(up(1), \mathtt{t}, I) \wedge pert(up(1), \mathtt{p}, I) \end{aligned} \tag{6.24}$$

which, since performed actions are always pertinent (axiom (6.5)), can be further simplified into:

$$holds(up(1), \mathtt{f}, I-1) \wedge holds(toggle(1), \mathtt{t}, I) \supset holds(up(1), \mathtt{t}, I) \wedge pert(up(1), \mathtt{p}, I) \tag{6.25}$$

This last simplification can be enunciated as the following property:

Property 15 *Let E* **if** *C* **after** *D be any $\mathcal{P}$-rule where C contains a conjunct like $holds(a, v)$ or $pert(p, \mathtt{p})$, for some action a or some symbol p. Then, the translation of C is equivalent to the formula:*

$$D_{I-1} \wedge C_I \supset \mathbb{C} E_I$$

Proof

The only difference with respect to (6.18) is that C_I is not required to be caused, but just to be true instead. In other words, we do not require explicitly that C_I is pertinent. As C_I is a conjunction of atoms, C_I is pertinent whenever one of its atoms is so. Assume one of these atoms is $holds(a, v, I)$. When the atom is true, by axiom (6.5) we get that $pert(a, \mathtt{p}, I)$ is true, and so $!C_I$ must also be true. On the other hand, if one of the atoms is directly $pert(p, \mathtt{p}, I)$, then requiring it to be true directly implies $!C_I$. So, in both cases we get $C_I \supset !C_I$ and we can omit $!C_I$ in the translation. □

Using this property, the rest of rule translations for R_s are:

$$holds(up(1), \mathtt{t}, I-1) \wedge holds(toggle(1), \mathtt{t}, I) \supset holds(up(1), \mathtt{f}, I) \wedge pert(up(1), \mathtt{p}, I) \quad (6.26)$$
$$holds(up(2), \mathtt{f}, I-1) \wedge holds(toggle(2), \mathtt{t}, I) \supset holds(up(2), \mathtt{t}, I) \wedge pert(up(2), \mathtt{p}, I) \quad (6.27)$$
$$holds(up(2), \mathtt{t}, I-1) \wedge holds(toggle(2), \mathtt{t}, I) \supset holds(up(2), \mathtt{f}, I) \wedge pert(up(2), \mathtt{p}, I) \quad (6.28)$$

Finally, in order to obtain a compact representation of observations, one more construction is introduced. Assume we want to enunciate a temporal projection problem consisting of an initial situation σ_0 plus a sequence of action executions $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$. If we just represent the observations as atoms:

$$atoms(\sigma_0) \cup atoms(\alpha_1) \cup \cdots \cup atoms(\alpha_n)$$

we will be just asserting which elementary actions are performed, but not which actions *are not* performed. To avoid this, several solutions can be proposed. The simplest one is to explicitly include a set of negative pertinence literals, asserting which actions are not performed. When we want to express that the action execution is completed in this way, we will use the following abbreviation:

$$\mathbf{do}\ (\alpha, i) \stackrel{\text{def}}{=} \left(\bigwedge_{holds(a,v) \in \alpha} holds(a, v, i) \right) \wedge \left(\bigwedge_{a \in \mathcal{A} - symb(\alpha)} pert(a, \mathtt{n}, i) \right)$$

For instance:

$$\mathbf{do}\ (\{load\}, 1) \stackrel{\text{def}}{=} holds(load, \mathtt{t}, 0) \wedge pert(shoot, \mathtt{n}, 0)$$

that is, we assert that the only performed action is *load*.

Given a sequence of compound actions $\vec{\alpha} = \alpha_1, \ldots, \alpha_n$, we denote:

$$\mathbf{do}\ (\vec{\alpha}) \stackrel{\text{def}}{=} \bigwedge_{i \in [1,n]} \mathbf{do}\ (\alpha_i, i)$$

Of course, a more general solution relying on some kind of default reasoning can be proposed. For instance, we could minimize the occurrence of actions using some simple circumscription policy like CIRC$[atoms(\alpha_i); holds]$. However, due to the shape of the minimized formulas and to pertinence calculus axioms related to action occurrences, this circumscription always amounts to the formula **do** $(\alpha, 1)$, and so, we do not get any real advantage of using a more elaborated definition. Minimizing the occurrence of actions could be interesting, perhaps, not for temporal projection, but for other kind of problems like minimal planning, where this minimization is not so trivial.

6.4 Circumscribing pertinence

Now, let T_s be the pertinence calculus theory consisting of the set of axioms AX $\cup$ (UFR) plus the translations (6.19)-(6.28) of $\mathcal{P}$-rules for the suitcase domain. The following question is straightforward: do the models of T_s describe the set of narratives we obtained in figure 4.2 using the operational behavior? In other words, does our pertinence calculus encoding provide the right semantics for the algorithmic description?

The answer is clearly negative yet, as we can see with the following counterexample. Assume we just want to toggle lock 1 when the suitcase is closed and both locks are down. This would correspond to:

$$holds(up(1), \mathtt{f}, 0) \wedge holds(up(1), \mathtt{f}, 0) \wedge holds(open, \mathtt{f}, 0) \wedge \ \mathbf{do}\ (\{toggle(1)\}, 1) \tag{6.29}$$

The result should be clear: lock 1 must be caused to be up, whereas the rest should persist by inertia. In fact, this is the solution provided by the operational semantics. In figure 4.2, there exists a unique successor state corresponding to $\sigma_1 = \{up(1), \overline{up(2)}, \overline{open}\}$ with π_1 making $toggle(1)$ and $up(1)$ pertinent. The theory $T_s \cup (6.29)$, however, just allows concluding that $up(1)$ becomes true – nothing is entailed about $up(2)$ nor $open$. In fact, we get seven models which only vary in the valuations of $holds$ and $pert$ for fluents $up(2)$ and $open$ in situation 1. Using L^2 notation for representing $holds$ and $pert$ values, these models correspond to:

	$up(2)$	$open$
M_1	fn	fn
M_2	fn	fp
M_3	fp	fn
M_4	fp	fp
M_5	fn	tp
M_6	fp	tp
M_7	tp	tp

The expected model is M_1, where both $up(2)$ and $open$ persist false unaffected. However, our representation is still too weak: we have not said anywhere that $up(2)$ and $open$ cannot become pertinent by only performing $toggle(1)$. Thus, we have the other six possibilities which are clearly counterintuitive. The reason for these extra models is that, as explained in the introduction, a logical description of a dynamic system requires not only the formulas that express the changes, but also some kind of mechanism that allows concluding that unaffected facts do not change: the *inertia law*. Of course, we could try to add more formulas for this purpose (effect axioms), but their shape would vary depending on the whole domain knowledge, leading to the already explained frame problem. Instead, we will explain how to incorporate the common sense law of inertia into pertinence calculus, using in this section a models minimization technique.

As we had seen, the inertia default can be stated as follows:

everything persists, under no evidence on the contrary

In our case, since the universal frame axiom (UFR) already asserts that nonpertinent fluents persist, we could rephrase inertia as:

everything is not pertinent, under no evidence on the contrary

which also copes with the requirement of minimizing unneeded pertinence. This intuitive behavior can be simply achieved by selecting those models with less pertinent atoms. The question now is how to measure "less pertinence" between two given models. To this aim, we can use now the binary representation for the pertinence predicate, $pert(P, I)$, bearing in mind that both $pert(P, \mathtt{p}, I)$ and $pert(P, \mathtt{n}, I)$ are the already seen abbreviations of $pert(P, I)$ and $\neg pert(P, I)$, respectively. In this way, we can simply try, as a first attempt, to select those models of our theory T with less extent for $pert(P, I)$ regardless the extent of $holds(P, V, I)$. As we had seen in section 2.1, this ordering relation is denoted as $\leq^{pert;holds}$ and corresponds to the circumscription:

$$\mathrm{CIRC}[T; pert; holds]$$

This models minimization solves our previous scenario, since we would get M_1 as unique minimal model (it is the only one without pertinent atoms), but it is still too naive and does not cover all the cases for the system automaton. As a counterexample, consider that we toggle lock 1 when it was down, lock 2 was up and the suitcase was closed:

$$holds(up(1), \mathtt{f}, 0) \wedge holds(up(2), \mathtt{t}, 0) \wedge holds(open, \mathtt{f}, 0) \wedge \ \mathbf{do}\ (toggle(1), 1) \tag{6.30}$$

Note that this was the original scenario introduced by Lin. Following machine in figure 4.2, the unique successor state should be that lock 1 is caused to be up, and the suitcase caused to be open:

$$\sigma_1 = \{up(1), up(2), open\} \tag{6.31}$$
$$\pi_1 = \{!up(1), \overline{!up(2)}, !open\} \tag{6.32}$$

However, the theory $T_s \cup (6.30)$ has five models that vary for the following fluents at situation 1:

	$up(2)$	$light$
M_8	fp	fn
M_9	fp	fp
M_{10}	fp	tp
M_{11}	tn	tp
M_{12}	tp	tp

and so, we get two $\leq^{pert;holds}$-minimal models, M_8 and M_{11}. Notice that we have arrived to the surprising model explained in the introduction: while M_{11} provides the expected behavior (the suitcase results open), M_8 prefers the persistence of *open* false, "moving down" lock 2.

To overcome this difficulty, many change based approaches [65, 47, 103] apply a *filter preferential entailment* [96] technique, consisting in minimizing only a part of the theory. More concretely, these action approaches minimize the change predicate in the whole theory *excepting* the universal frame axiom, which must be satisfied afterwards. An analogous process can be established in pertinence calculus, using instead the ordering relation, $\leq^{pert}$, so that we require *holds* fixed before minimizing the extent of *pert*. Note that this relation is more restrictive than $\leq^{pert;holds}$, since two models with a different extent for *holds* are not comparable now. Given this modified ordering relation, we define the selected models as follows.

Definition 35 (Selected model) Given a set R of $\mathcal{P}$-rules, let the pertinence calculus theory T be $t(R) \cup \mathrm{AX} \cup Obs$ where Obs is a set of observations. Then, a model M of T is said to be *selected* iff:

i) M is a $\leq^{pert}$-minimal model of T,

ii) M satisfies the universal frame axiom: $M \models$ (UFR).

In other words, the selected models of T are the models of the circumscribed theory:

$$\mathrm{CIRC}[t(R) \cup \mathrm{AX} \cup Obs; pert] \cup (\mathrm{UFR})$$

□

Coming back to our example, let T_s contain the rule translations (6.19)-(6.28) plus axioms AX and observations (6.30). There exist ten models of T_s, corresponding to M_8-M_{12} plus those that, at situation 1, vary in these fluent valuations:

	$up(2)$	$open$
M_{13}	fn	fn
M_{14}	fn	fp
M_{15}	fn	tn
M_{16}	fp	tn
M_{17}	fn	tp

Notice that these five extra models appear now because we are not requiring (UFR) – in all of them, some fluent is nonpertinent while it has changed its truth value with respect to σ_0. We select now those models with less pertinence, fixing the truth mapping. Thus, we can classify the models by each truth combination:

	$up(2)$	$open$
M_9	fp	fp
M_8	fp	fn
M_{14}	fn	fp
M_{13}	fn	fn
M_{10}	fp	tp
M_{16}	fp	tn
M_{17}	fn	tp
M_{15}	fn	tn
M_{12}	tp	tp
M_{11}	tn	tp

obtaining three $\leq^{pert}$ minimal models, M_{13}, M_{15} and M_{11}. Finally, since M_{13} and M_{15} do not satisfy (UFR), the unique selected model is M_{11}, which is the expected one (lock 2 persists up, whereas the suitcase results open).

6.4.1 Correspondence to operational semantics

Although the applied minimization technique has solved the suitcase problem in its usual terms, we cannot be actually sure that similar problems will not appear for other scenarios, or even for other transitions of the suitcase domain. In fact, this objection is sometimes applicable to most works in the literature[3], where the proof for showing that there are no "surprising" results with indirect effects is simply left to an study of a particular domain and a particular action execution. In this way, the usual methodology amounts to a trial and error process where it is always possible to look for a new scenario that leads to "unnatural" results.

In our case, there exists a considerable advantage: we have available the operational interpretation of rules which plays the role of "expected behavior." Thus, rather than constructing a proof for each imaginable scenario, we will look for a general result of correspondence with respect to the operational interpretation. Unfortunately, it is not possible to obtain a full correspondence when using the already explained circumscription policy, as shown with the following counterexample.

Example 6 (The alarm scenario) An alarm system detects if somehow people enter a building. We assume that there may be many different ways to enter the building, that is, there are many actions bringing someone in. We use the fluents *in* (stating that someone is inside), *active* (the alarm system is active) and *ring* (the alarm bell is ringing). While the system is active, anyone entering the building triggers the alarm: if *in* becomes true when *active* is already true, *ring* becomes true. However, activating the alarm system when someone is already in the building is not supposed to cause the bell to ring. □

This problem was proposed in [15] to emphasize the difference between causal rules, which talk about change propagations, and state constraints like:

$$in \wedge active \supset ring$$

which would not behave correctly when we activate the alarm while someone is inside.

For simplicity sake, we consider an action *enter* that causes *in* to become directly true. Of course, in order to leave open the possibility of other ways of making *in* true, we avoid the use of action *enter* in the rule for *ring*. We will also handle two actions, *connect* and *disconnect* for respectively activating or deactivating the alarm. Finally, as we will allow concurrent actions, we assume that activating the alarm while a person is entering does not cause the ring to bell. This assumption was not clarified in the original problem, but it is interesting for our purposes.

Under these assumptions, the set R_a of $\mathcal{P}$-rules would be:

$$\begin{array}{rll} active & \textbf{if} & connect \\ \overline{active} & \textbf{if} & disconnect \\ in & \textbf{if} & enter \\ ring & \textbf{if} & in \wedge active \wedge \overline{!active} \end{array} \qquad \begin{array}{r} (6.33) \\ (6.34) \\ (6.35) \\ (6.36) \end{array}$$

Notice that the rule for *ring* requires that *active* is not only true but also persistent. In other words, we want the alarm to be active, but *not caused* to be so. It is easy to see that, if we use the algorithm in figure 5.2 to compute the successor state for the transition:

$$\begin{array}{rcl} \sigma_0 & = & \{\overline{in}, active, \overline{ring}\} \\ \alpha_1 & = & \{enter\} \end{array}$$

[3]As the most remarkable attempt, if not the only one, of a systematic assessment of action approaches, see [97].

we obtain:

$$\begin{aligned}\sigma_1 &= \{in, active, ring\}\\ \pi_1 &= \{!in, \overline{!active}, !ring\}\end{aligned}$$

that is, as expected, the bell begins ringing.

Unfortunately, the circumscriptive policy for the pertinence calculus encoding does not allow to conclude that the bell rings. The translation $t(R_a)$ would be the set of formulas:

$$holds(connect, \mathtt{t}, I) \supset holds(active, \mathtt{t}, I) \wedge pert(active, \mathtt{p}, I) \quad (6.37)$$

$$holds(disconnect, \mathtt{t}, I) \supset holds(active, \mathtt{f}, I) \wedge pert(active, \mathtt{p}, I) \quad (6.38)$$

$$holds(enter, \mathtt{t}, I) \supset holds(in, \mathtt{t}, I) \wedge pert(in, \mathtt{p}, I) \quad (6.39)$$

$$holds(in, \mathtt{t}, I) \wedge holds(active, \mathtt{t}, I) \wedge pert(active, \mathtt{n}, in) \wedge pert(in, \mathtt{p}, I) \supset holds(ring, \mathtt{f}, I) \wedge pert(ring, \mathtt{p}, I) \quad (6.40)$$

whereas the set of observations, Obs, corresponds to:

$$holds(in, \mathtt{f}, 0) \wedge holds(active, \mathtt{t}, 0) \wedge holds(ring, \mathtt{f}, 0) \wedge \ \mathbf{do}\ (\{enter\}, 1)$$

Clearly, any model of $t(R_a) \cup \mathrm{AX} \cup Obs$ satisfies $holds(in, \mathtt{t}, 1) \wedge pert(in, \mathtt{p}, 1)$ (fluent in is caused true). However, we get *two* different selected models, which vary at situation 1 for:

	active	*ring*
M_1	tn	tp
M_2	tp	fn

Which is the explanation for this uncertainty? As we saw before, model M_1 is the expected one, where persistence of *active* has been preferred before deciding *ring*. In the other model, persistence of *ring* has been preferred "first." Then, formula (6.40) has been applied *by contraposition* allowing to conclude $pert(active, \mathtt{p}, 1)$. The problem here is that there is no intuitive explanation for obtaining pertinence of *active*. Notice that the two rules that may affect *active* are not applicable, since neither *connect* nor *disconnect* are performed. In this way, models M_2 clearly violates postulate P1: *active cannot be* affected by the performed actions. Allowing this behavior would imply assuming that ramifications can be obtained by reasoning from effects to causes[4].

Therefore, circumscribing our pertinence calculus translation does not allow us to conclude that the bell begins ringing, since there exists a model in which *ring* is true and a model in which it is false. Furthermore, this counterexample also shows that, when we use the proposed circumscription policy, the induced transition relation is *nondeterministic*.

Fortunately, this deviation from the operational interpretation is not the most frequent case. In fact, it is possible to establish a syntactic constraint in the $\mathcal{P}$-rules that guarantees the correspondence with respect to the operational behavior.

Definition 36 (Definiteness) A set R of $\mathcal{P}$-rules is said to be *definite* iff no rule condition contains any negative pertinence fact: $\overline{!p}$. □

[4] Although this behavior is clearly counterintuitive for temporal projection, it has been applied for the case of diagnosing dynamic systems, as proposed in [84].

As we can see, the alarm example is not definite, since we require testing that fluent *active* is not pertinent while fluent *in* is caused true. Of course, restricting the representation to definite $\mathcal{P}$-rules means an important limitation with respect to our initial goals since, in this way, we cannot freely handle pertinence in the same way as fluent values. However, the need for testing nonpertinence of a fluent (or that it has not been caused) is not so frequent, at least in most of the action domain examples used in the literature[5].

Definiteness becomes interesting because it allows us to apply a transformation which is very similar to Clark's completion for logic programs, but adapted to pertinence in $\mathcal{P}$-rules.

Definition 37 (Pertinence Completion of $\mathcal{P}$-rules) Let R be a set of $\mathcal{P}$-rules, and $t(R)$ be their translations into pertinence calculus. Then, we define the *pertinence completion* of R, written PCOMP$[R]$, as the union of $t(R)$ and the set of pertinence calculus formulas:

$$pert(f, \mathtt{p}, I) \quad \equiv \quad \bigvee_k \mathbb{C}C_I^k \wedge D_{I-1}^k \tag{6.41}$$

for each fluent f and all the rules E **if** C^k **after** D^k with $symb(E) = f$. □

Notice that, in fact, $t(R)$ already contains the right-to-left direction of (6.41), and so, the only real addition is the left-to-right implication. Now, under the assumption of definite $\mathcal{P}$-rules, we can replace circumscription by pertinence completion, as stated by the following lemma:

Lemma 3 *For any definite acyclic set R of $\mathcal{P}$-rules:*

$$\mathrm{CIRC}[t(R); pert] \equiv \mathrm{PCOMP}[R]$$

Proof
(See appendix A). □

As an example of pertinence completion, consider again Lin's suitcase scenario, with the set of $\mathcal{P}$-rules R_s=(5.4)-(5.8). The pertinence completion PCOMP$[R_s]$ would include the translation of the rules $t(R_s)$ ((6.19), (6.25)-(6.28)) plus the additional formulas:

$$pert(up(1), \mathtt{p}, I) \equiv \big(holds(up(1), \mathtt{f}, I-1) \wedge holds(toggle(1), \mathtt{t}, I)\big) \vee \big(holds(up(1), \mathtt{t}, I-1) \wedge holds(toggle(1), \mathtt{t}, I)\big) \tag{6.42}$$

$$pert(up(2), \mathtt{p}, I) \equiv \big(holds(up(2), \mathtt{f}, I-1) \wedge holds(toggle(2), \mathtt{t}, I)\big) \vee \big(holds(up(2), \mathtt{t}, I-1) \wedge holds(toggle(2), \mathtt{t}, I)\big) \tag{6.43}$$

$$pert(open, \mathtt{p}, I) \equiv holds(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I) \wedge (pert(up(1), \mathtt{p}, I) \vee pert(up(2), \mathtt{p}, I)) \tag{6.44}$$

[5]As an interesting discussion, see Thielscher's comment and the authors' answer at the online review of Denecker et al's paper [28]:
`http://www.ida.liu.se/ext/etai/ra/rac/009/rppf.html#004`

In fact, thanks to pertinence calculus axioms for fluent values, formulas (6.42) and (6.43) can be further simplified into the respective equivalences:

$$\begin{aligned} pert(up(1), \mathtt{p}, I) &\equiv holds(toggle(1), \mathtt{t}, I) \\ pert(up(2), \mathtt{p}, I) &\equiv holds(toggle(2), \mathtt{t}, I) \end{aligned}$$

Now, the main theorem of this section guarantees that, under the syntactic limitations of definiteness and acyclicity, the circumscription we have defined for our pertinence calculus encoding obtains the same results than the operational semantics presented in the previous chapter:

Theorem 7 *Let R be a definite set of acyclic causal rules and let σ_0 be some initial state and $\vec{\alpha}$ a sequence of actions $\alpha_1, \ldots, \alpha_n$. Then, ν is a narrative for σ_0 and $\vec{\alpha}$ under the operational semantics iff the interpretation $atoms(\nu)$ is a model of the theory:*

$$\mathrm{CIRC}[t(R) \cup \mathrm{AX} \cup atoms(\sigma_0) \cup \ \mathbf{do}\ (\vec{\alpha}); pert] \cup (UFR)$$

Proof
(See appendix A). □

In fact, even when some rules in R contain negative pertinence facts in their conditions, any narrative of the operational semantics is still a selected model in the circumscribed theory. However, the other direction does not necessary apply, since the presence of facts like $\overline{!p}$ may lead to more selected models, apart from the one corresponding to the narrative obtained by the rule application algorithm, as we saw with the alarm example.

6.5 Encoding Pertinence into Default Logic

As we have just observed, a suitable pertinence circumscription allows avoiding most problems with contraposition of material implication. Theorem (7) guarantees that the circumscriptive policy we have introduced captures the behavior of the operational description, for which it can be trivially checked that no effect can be obtained by contraposition. However, we have had to pay a price for obtaining the correspondence result: rule conditions must not contain negative pertinence facts. Otherwise, as shown with the alarm example, there could be models of the circumscribed theory that are not narratives of the operational description. In fact, these "non-covered" models are obtained by contraposition of (the translation of) some causal rule.

Of course, a radical solution could be to forbid rule conditions with negative pertinence facts. In this way, we would understand that the circumscriptive encoding is the *right semantics* for causal rules, and that references to nonpertinence have some special meaning not captured by the operational behavior. However, this asymmetric treatment of pertinence does not seem very natural and moves away from one of our goals, that is, to provide a uniform treatment of pertinence information, similar to the one for fluent values. Besides, we would be somehow "breaking" the methodology: once we decided to look for a semantics for the operational description, we should rather try to capture its exact behavior. Other possibilities are neither better nor worse, but describe a different thing.

In this section we study how to avoid contraposition by reinforcing pertinence calculus with inference rules, while maintaining the pertinence minimization. To this purpose, we use a well known nonmonotonic formalism, *default logic* [94], which is, in fact, the most suitable among the usual ones for dealing with inference rules. Default logic is also important from the practical point of view, as it allows exploiting the following property: a default theory that exclusively deals with atoms corresponds to a logic program under the stable models semantics [38]. This property is interesting from the point of view of implementation, as there exist several practical efficient tools [105, 32] for computing the stable models of a logic program. In this way, we follow here similar steps for pertinence calculus as done in [109] for the action language $\mathcal{AC}$, providing first an embedding into default logic and, in the next sections, into logic programming.

If we look at the nature of most of the representational problems already explained (the Yale shooting, the suitcase and the alarm examples, for instance), there exists in all of them a causal violation where we obtain conclusions by applying some conditional expression in the "wrong" direction. That is, it seems that the intended meaning of some of the conditionals we have used does not fit with a material implication, but with an inference rule instead. For instance, from the suitcase and the alarm examples it seems clear that causal rules should be represented using inference rules. However, this would not be enough for dealing with the Yale shooting problem. In that case, the problem arose because of applying backwards the universal frame axiom (UFR). After considering this, we present the encoding, which is actually divided into three clearly differentiated parts.

Definition 38 (D(R)) Let R be a set of causal rules. We define the default theory $D(R)$ as the four sets of default rules:

1. $D(R)_1$ contains the formulas in AX, i.e., axioms (DC), (UNA) and (6.1)-(6.6),
2. $D(R)_2$ contains the inference rule schemata
$$\frac{pert(F, \mathtt{n}, I) \wedge holds(F, V, I-1)}{holds(F, V, I)} \tag{6.45}$$
for any fluent F and $I \in [1, n]$, plus the inference rule schemata
$$\frac{D_{I-1} \wedge \mathbb{C} C_I}{\mathbb{C} E_I} \tag{6.46}$$
for each causal rule E **if** C **after** D and $I \in [1, n]$,
3. $D(R)_3$ contains the default rule schemata
$$\frac{: pert(P, \mathtt{n}, I)}{pert(P, \mathtt{n}, I)} \tag{6.47}$$
for any symbol P and $I \in [1, n]$,

□

As interesting remarks, note that each set contains different types of rules. The set $D(R)_1$ exclusively contain classical formulas that correspond to the basic set of pertinence calculus axioms. The set $D(R)_2$ is the corpus of inference rules. It is interesting to note that all of them show a clear *temporal directionality*, since their consequents always refer to a situation I which is *greater or equal* than the situations in the antecedents. In other words, we never apply an inference rule to obtain conclusions from future to past. Finally, the set $D(R)_3$ consists of normal defaults that clearly express the minimization of pertinence: whenever some symbol P can be assumed to be nonpertinent, it is concluded to be so. This minimization, in conjunction with the universal frame axiom (6.45), implements the commonsense law of inertia.

6.6 Encoding pertinence into logic programming

The default logic encoding has some serious drawbacks from a practical point of view. The most important one is the use of logically closed theories which, in principle, imply dealing with sets of an infinite number of formulas. However, for our purposes, the full expressivity of default logic is not actually needed. A first example of this is that we only use a particular type of normal defaults (rule (6.47)). Another interesting observation is that all the consequents in $D(R)$ are atoms (like $holds(F,V,I)$ in (6.45)) or conjunctions of atoms (like $\mathbb{C}E_I$ in (6.46)). Motivated by this last feature, we can study the feasibility of imposing the following strong restriction:

consider theories and rules in which all the propositional formulas are atoms.

As we saw in the background, a default theory under that restriction corresponds to a logic program under the stable models semantics [38], whose definition is simpler and for which there exist efficient implementations [105, 32]. For this reason, we will not provide a correspondence theorem for the default logic encoding with respect to the operational semantics. Instead, we will explain how the already defined default theories can be suitably rearranged as logic programs and, afterwards, provide the correspondence with respect to these programs.

Given any set R of causal rules, we define the program $P(R)$ consisting of three sets of program rules, $P(R)_1$, $P(R)_2$ and $P(R)_3$, analogous to the sets defined for $D(R)$. The translations of $P(R)_2$ (the corpus of inference rules) and $P(R)_3$ (the pertinence minimization) are straightforward. The set $P(R)_2$ contains, for any fluent F and $I \in [1,n]$, the program rules:

$$holds(F,V,I) \leftarrow pert(F,\mathtt{n},I),\ holds(F,V,I-1) \tag{6.48}$$

plus, for each causal rule $holds(f,v)$ **if** C **after** D in R, and for each symbol p occurring in C, the rules:

$$\begin{array}{rcl} holds(f,v,I) & \leftarrow & D_{I-1}, C_I, pert(p,\mathtt{p},I) \\ pert(f,\mathtt{p},I) & \leftarrow & D_{I-1}, C_I, pert(p,\mathtt{p},I) \end{array}$$

assuming $I \in [1,n]$ and that $\wedge$'s are replaced by commas in D_{I-1} and C_I.

The explanation for this translation is simple. The expression $D_{I-1} \wedge \mathbb{C}C_I$ we used before as antecedent cannot be represented as a conjunction of atoms. However, we can reexpress the formula into its disjunctive normal form, and then create a different program rule with each clause as condition. Since the formula is equivalent to $D_{I-1} \wedge C_I \wedge !C_I$, and both D_{I-1} and C_I are conjunctions, we only have to decompose $!C_I$ which is the disjunction of $pert(p,\mathtt{p},I)$ for all the atoms occurring in C. Besides, for each one of these conditions, we get two rules, since the effect must be caused, i.e., must become both true and pertinent.

The set $P(R)_3$ is a direct translation of $D(R)_3$, that is, for any symbol P and $I \in [1,n]$, it contains the single rule:

$$pert(P,\mathtt{n},I) \leftarrow not\ pert(P,\mathtt{p},I) \tag{6.49}$$

Finally, in order to encode the basic axioms AX, we understand $\bot$ as a special atom in the Herbrand base, so that we will reject later those stable models containing this atom. The set $P(R)_1$ contains the rules:

$$\bot \leftarrow holds(P,V,I),\ holds(P,V',I) \tag{6.50}$$

$$\bot \leftarrow pert(P,\mathtt{p},I),\ pert(P,\mathtt{n},I) \tag{6.51}$$

$$pert(A,\mathtt{p},I) \leftarrow holds(A,V,I) \tag{6.52}$$

with $I \in [0, n]$. Rules (6.50)-(6.52) respectively correspond to axioms (6.1), (6.3), and (6.5). The axioms (DC) and (UNA) are not needed since we already handle Herbrand models, whereas axiom (6.6) has also been omitted, since there are no rules with head $pert(P, \mathtt{p}, 0)$ and so (6.49) makes everything nonpertinent at situation 0. Note also that we do not need to include the axioms for existence of value, (6.2) and (6.4), and that in fact, rule (6.52) only represents one of the directions of the double implication of (6.5).

As an example of formulation, consider again rule *open* **if** $up(1) \wedge up(2)$. As we had seen:

$$\begin{aligned} D_I &\stackrel{\text{def}}{=} \top \\ C_I &\stackrel{\text{def}}{=} holds(up(1), \mathtt{t}, I) \wedge holds(up(2), \mathtt{t}, I) \\ !C_I &\stackrel{\text{def}}{=} pert(up(1), \mathtt{p}, I) \vee pert(up(2), \mathtt{p}, I) \end{aligned}$$

and so, we would have the logic program rules:

$$\begin{aligned} holds(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(1), \mathtt{p}, I) \\ pert(open, \mathtt{p}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(1), \mathtt{p}, I) \\ holds(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(2), \mathtt{p}, I) \\ pert(open, \mathtt{p}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(2), \mathtt{p}, I) \end{aligned}$$

The final logic program for Lin's suitcase domain contains the rule translations:

$$\begin{aligned} holds(up(N), \mathtt{t}, I) &\leftarrow holds(up(N), \mathtt{f}, I-1),\ holds(toggle(N), \mathtt{t}, I) \\ holds(up(N), \mathtt{f}, I) &\leftarrow holds(up(N), \mathtt{t}, I-1),\ holds(toggle(N), \mathtt{t}, I) \\ holds(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(1), \mathtt{p}, I) \\ pert(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(1), \mathtt{p}, I) \\ holds(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(2), \mathtt{p}, I) \\ pert(open, \mathtt{t}, I) &\leftarrow holds(up(1), \mathtt{t}, I),\ holds(up(2), \mathtt{t}, I),\ pert(up(2), \mathtt{p}, I) \end{aligned}$$

for any $I \in [1,n]$ and $N \in \{1,2\}$, plus the axiom translations:

$$
\begin{array}{rcl}
holds(up(N),\mathtt{t},I') & \leftarrow & holds(up(N),\mathtt{t},I'-1),\ pert(up(N),\mathtt{n},I') \\
holds(up(N),\mathtt{f},I') & \leftarrow & holds(up(N),\mathtt{f},I'-1),\ pert(up(N),\mathtt{n},I') \\
holds(open,\mathtt{t},I') & \leftarrow & holds(open,\mathtt{t},I'-1),\ pert(open,\mathtt{n},I') \\
pert(up(N),\mathtt{f},0) & & \\
pert(toggle(N),\mathtt{f},0) & & \\
pert(open,\mathtt{f},0) & & \\
pert(toggle(N),\mathtt{t},I) & \leftarrow & holds(toggle(N),\mathtt{t},I) \\
\bot & \leftarrow & holds(open,\mathtt{t},I),\ holds(open,\mathtt{f},I) \\
\bot & \leftarrow & holds(up(N),\mathtt{t},I),\ holds(up(N),\mathtt{f},I) \\
\bot & \leftarrow & pert(open,\mathtt{p},I),\ pert(open,\mathtt{n},I) \\
\bot & \leftarrow & pert(up(N),\mathtt{p},I),\ pert(up(N),\mathtt{n},I) \\
\bot & \leftarrow & pert(toggle(N),\mathtt{p},I),\ pert(toggle(N),\mathtt{n},I) \\
pert(up(N),\mathtt{f},I) & \leftarrow & not\ pert(up(N),\mathtt{t},I) \\
pert(open,\mathtt{f},I) & \leftarrow & not\ pert(open,\mathtt{t},I) \\
pert(toggle(N),\mathtt{f},I) & \leftarrow & not\ pert(toggle(N),\mathtt{t},I)
\end{array}
$$

for $I \in [0,n]$, and $I' \in [1,n]$.

6.6.1 Correspondence to operational semantics

Of course, in order to describe the program behavior, we must choose one of the possible logic programming semantics. In this way, while stable models allow obtaining the same results than default logic (when we restrict the study to atoms), other logic programming semantics like well-founded semantics or Clark's completion, may lead to different results. However, the next theorem will be very important for comparison purposes with respect to the operational behavior, since it shows that for an acyclic set R of $\mathcal{P}$-rules, the resulting program $P(R)$ is *hierarchical*, and so, by property (9) seen in the background, the three logic programming semantics will actually coincide.

Theorem 8 *Given a set of acyclic causal rules R, the program $P(R)$ is hierarchical.*
Proof (See appendix A). □

Corollary 5 *Let R be an acyclic set of causal rules, σ_0 an initial state σ_0 and $\vec{\alpha}$ a sequence of compound actions. Then, the program $P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$ is hierarchical.*
Proof
Trivial, since $atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$ are a collection of atoms (we are assuming that the conjunction of atoms obtained from the **do** abbreviation is represented as a set of logic program facts). □

The previous results allow a straightforward application of proposition (9), so that we always obtain a unique stable model, which is also the unique supported model and, furthermore, corresponds to the (complete) WFM. Therefore, in order to show the correspondence with respect to the operational behavior, it will suffice to use for this purpose any of the three logic programming semantics, thanks to the absence of cycles.

Theorem 9 *Let R be an acyclic set of causal rules, σ_0 an initial state and $\vec{\alpha} = \alpha_1 \dots \alpha_n$ a sequence of compound actions. Then, ν is a narrative for σ_0 and $\vec{\alpha}$ under the operational semantics iff $atoms(\nu)$ is the complete WFM of the program $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$.*
Proof
(See appendix A). ☐

6.6.2 Using program cycles for planning and temporal explanation

Although theorem 9 has shown that it is possible to capture the operational behavior using logic programs (interpreted under any of the three semantics we have considered), these logic programs have a disadvantage: they are only thought for temporal projection problems. In other words, we actually construct *a different program* for each initial state and sequence of actions, obtaining a unique model which describes the resulting narrative (remember we are considering, by now, a deterministic transition relation). However, we know that under some semantics, like stable models or Clark's completion, a logic program may be satisfied by several models and not just by only one. The question is, could we construct a *single* program so that its models correspond to the set of all possible narratives?

We will show how a suitable use of logic program negative cycles may allow the generation of all the possible narratives for any sequence of transitions. This is very interesting since, in this way, we can solve temporal explanation and planning problems by just adding observations (as facts) to the program, rather than constructing all the possible programs for all the possible narratives.

Definition 39 (P_{gen}) Given the set of actions $\mathcal{A}$, fluents $\mathcal{F}$ and a fixed narrative length n, we define the program P_{gen} as the set of rules:

$$holds(F, v, 0) \quad \leftarrow \quad not\ holds(F, v_1, 0), \dots, not\ holds(F, v_m, 0)$$

for any fluent F, any $v \in range(F)$ and $\{v_1, \dots, v_m\} = range(F) - \{v\}$, plus the rules:

$$holds(A, u, I) \quad \leftarrow \quad not\ holds(A, u_1, I), \dots, not\ holds(A, u_m, I), not\ pert(A, \mathtt{n}, I)$$

for any $I \in [1, n]$, any action A, and any $u \in range(A)$ and $\{u_1, \dots, u_m\} = range(A) - \{u\}$. ☐

In other words, if no other value for fluent F at 0 can be proved, then we assign value v. In the case of actions, this is done at any situation greater than 0, but we add one more possibility $not\ pert(A, \mathtt{n}, I)$, allowing also the case in which the action is not performed (and so, no value can hold for it).

The following result shows the utility of P_{gen} when we interpret our logic programs either under stable models or under Clark's completion semantics.

Theorem 10 *Let R be an acyclic set of causal rules and n some fixed narrative length. Then the set of stable models (resp. supported models) of the program $P(R) \cup P_{gen}$ correspond to the stable models (resp. supported models) of the multiple programs $P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$ for any possible initial state σ_0 and any possible sequence of actions $\vec{\alpha} = \alpha_1, \dots, \alpha_n$.*
Proof
(See appendix A). ☐

Notice that this covers the case of an acyclic R. Thus, a unique program $P(R) \cup P_{gen}$ is able to provide the set of all narratives captured by the finite state machine we would obtain applying the operational semantics. Besides, this result is in fact more general, since if we provide partial observations about σ_0 or $\vec{\alpha}$, the program will compute all the narratives resulting from completing this information. In this way, we can use program $P(R) \cup P_{gen}$ to solve temporal explanation problems and so, if the system is deterministic, to solve planning problems.

As we can see, there is not a similar result for WFS. This be no surprise, since, as we had already explained, any logic program has always a *unique* well-founded model. Thus, when we consider WFS, it does not make sense to talk about a *set* of models for the logic program for trying to capture the desired narratives. For instance, the negative cycles we introduced in P_{gen}, rather than generating multiple scenarios, they would actually constitute a problem in WFS, leaving undefined all the facts for the initial situation and action executions. From a practical point of view, this means that solving a postdiction or a planning problem using WFS involves an extra generation of possible scenarios which is not part of the program interpretation itself, but for which efficient strategies could be obtained. The main disadvantage, however, relies in the theoretical understanding, since planning and postdiction would be tasks "outside" our logical formalization.

Although it is outside the scope of this dissertation, two possible solutions to this problem can be outlined: the *disjunctive* and the *abductive* extensions for logic programming. *Disjunctive logic programming* (see [67] for a survey) consists in extending the shape of program rules to cope with disjunctive heads:

$$H_1 \mid \ldots \mid H_m \;\leftarrow\; L_1, \ldots, L_n$$

where H_i are atoms and L_i are program literals. Intuitively, the idea is that we may obtain different models for the program (even in WFS) depending on the selected choice of rule consequence. As an example, consider the program:

$$p \mid q \;\leftarrow\; not\ r$$

Since r cannot be proved, we should be able to apply the program rule. However, as the head is disjunctive, we should obtain two (complete) models: $\{p\}$ and $\{q\}$. The stable models semantics has been naturally extended for disjunctive programs [40], and in fact, one of the currently available tools [32] for efficiently computing the stable models semantics allows disjunctive constructions. Moreover, it is possible to reformulate P_{gen} using disjunctive heads instead of negative cycles[6]:

$$holds(F, v_1, 0) \mid \ldots \mid holds(F, v_m, 0) \tag{6.53}$$

$$holds(A, u_1, I) \mid \ldots \mid holds(A, u_n, I) \mid pert(A, \mathtt{n}, I) \tag{6.54}$$

for any $range(F) = \{v_1, \ldots, v_m\}$ and $range(A) = \{u_1, \ldots, u_n\}$. Unfortunately, there is no general agreement about the disjunctive version of WFS: there exist several approaches like *stationary semantics* [93], D-WFS [29], GDWFS [11] which in most cases do not even take into account the coherence problem.

Another alternative is using *abductive logic programming* (see [51] for a survey), which relies on applying abductive reasoning to a logic program. Abduction is a logical mechanism that

[6] In general, a negative cycle $(p \leftarrow not\ q), (q \leftarrow not\ p)$ is not equivalent to $(p|q)$. However, in this case, the presence of constraints (6.1) and (6.3) guarantee this equivalence.

allows meta-reasoning about some given inference relation, $\vdash$. Consider, for instance, some pair of related formulas $\phi \vdash \psi$. While deduction allows concluding which consequences ψ may follow from the fixed hypothesis ϕ, abduction allows *proposing the possible hypotheses* ϕ which could lead to the fixed consequence ψ. When abduction is applied to logic programming, the generation of hypotheses is usually limited to addition of atoms for a fixed set of predicates called *abducible*. This mechanism could be directly used in our case, by understanding as abducible the atoms for the initial situation and for the actions execution. In this way, we would obtain a logical counterpart of our practical "generation of possible cases." It must also be noticed that, contrarily to disjunctive logic programming, using abduction would allow real planning, even when the transition relation was nondeterministic, since the obtained abductions *guarantee* that the goal is finally obtained.

Chapter 7

Causal cycles

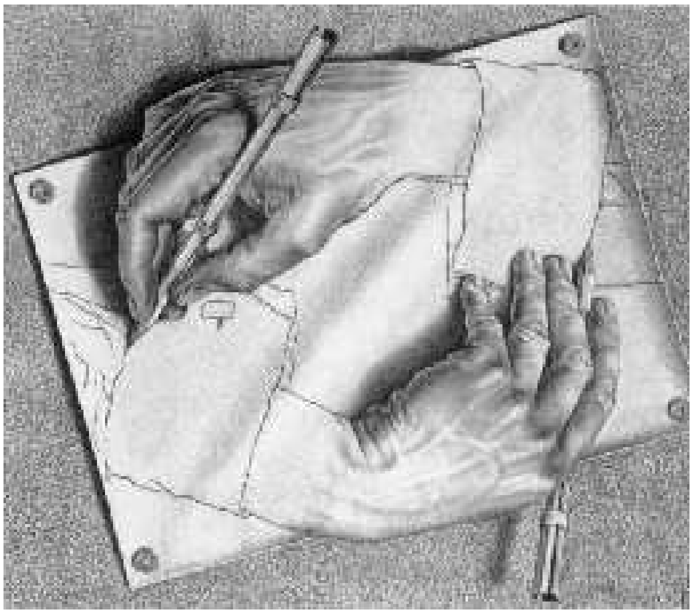


Figure 7.1: "*Drawing hands*" by M. C. Escher.

The previous chapter has shown how to encode the operational behavior of $\mathcal{P}$-rules into pertinence calculus by either using logic programming (theorem 9) or circumscription (theorem 7), although for the latter we had to additionally assume definiteness. An important feature that allowed the agreement of the different semantics for the case of logic programming (LP) was requiring acyclicity of the set of $\mathcal{P}$-rules. In fact, the operational interpretation was not defined when the set of rules contained cycles. Nevertheless, the presented logical formulations are still perfectly defined for causal cycles, but provide different results depending on the non-monotonic technique we use: circumscription, or logic programming, which also varies under Clark's completion, stable models or WFS.

The main motivation for studying causal cycles is to provide a coherent interpretation for our causal rule representation. Of course, we could impose the restriction of acyclicity on R, but this would mean a lack of flexibility when trying to represent the system in an elaboration tolerant way. Imagine updating a large system dealing with a considerable amount of rules. It is not so unusual that the addition of a new rule leads to a cyclic reference which, perhaps, is never a problem in practice, but simplifies the representation. On the other hand, although not so frequently, we can also find some domains in which the causal cycle itself has a natural meaning.

7.1 Characterizing the effects of causal cycles

The generalization of the $\mathcal{P}$-language to cope with causal cycles yields different effects depending on the nonmonotonic technique we apply to pertinence calculus. It is interesting to note that in all the four cases – circumscription, completion, stable models and WFS – we obtain new features in the behavior that are exclusively due to the presence of cycles. Without trying to be exhaustive, we informally proceed to relate the most relevant among these new features by studying simple examples of causal cycles. Many of these studied features were already commented for LP. The interest is that, although the logic programs for encoding $\mathcal{P}$-rules seem to be very simple (default negation is exclusively used for the normal default for minimizing pertinence), they are however complex enough for presenting all the effects derived from LP cycles.

In all the examples of this section, we assume a set of boolean fluents $\mathcal{F} = b, c, d$, an action $\mathcal{A} = \{a\}$ and that all the narratives have length $n = 1$ (i.e., we consider single transitions).

7.1.1 Positive vs. negative cycles

As $\mathcal{P}$-rules can be encoded into logic programs, the first question that may arise when studying causal cycles is whether they can be classified as positive/negative LP cycles or not. As we saw in chapter 2, this distinction is very useful for classifying the effects of LP cycles under different LP semantics, and so, it may indirectly help to fix the behavior of $\mathcal{P}$-rules. For instance, we know that the differences between stable models and WFS exclusively rely on negative LP cycles (property 11). Unfortunately, it is easy to see that, in fact, practically any causal cycle (i.e. a cycle in the $\mathcal{P}$-rules) leads to some *negative* cycle in its logic programming encoding. Consider the following simple example:

Example 7 Let R_1 be the singleton $\mathcal{P}$-rule:

$$b \textbf{ if } b \tag{7.1}$$

□

At a first glimpse, this example should just contain a positive cycle. However, in order to establish the type of LP cycle, we must not excessively rely on the fluent values. In fact, the rule:

$$\overline{b} \textbf{ if } \overline{b}$$

should behave in a completely analogous way. Instead, we must analyze the translation of R_1 into a logic program, $P(R_1)$ which contains, among other, the rules:

$$holds(b,\mathtt{t},1) \leftarrow holds(b,\mathtt{t},1),\ pert(b,\mathtt{p},1) \tag{7.2}$$
$$pert(b,\mathtt{p},1) \leftarrow holds(b,\mathtt{t},1),\ pert(b,\mathtt{p},1) \tag{7.3}$$
$$pert(b,\mathtt{n},1) \leftarrow not\ pert(b,\mathtt{p},1) \tag{7.4}$$
$$holds(b,\mathtt{t},1) \leftarrow holds(b,\mathtt{t},0),\ pert(b,\mathtt{n},1) \tag{7.5}$$

where (7.2) and (7.3) are the translation of (b **if** b), (7.4) is the rule (6.49) for fluent b, and (7.5) is the frame axiom for fluent b and value $\mathtt{t}$. Positive cycles are evident in (7.2) and (7.3) where the heads are also included in the bodies. However, we have also the following negative cycle:

$$pert(b,\mathtt{n},1) \stackrel{-}{\longleftarrow} pert(b,\mathtt{p},1) \stackrel{+}{\longleftarrow} holds(b,\mathtt{t},1) \stackrel{+}{\longleftarrow} pert(b,\mathtt{n},1)$$

following rules (7.4), (7.3) and (7.5) backwards.

Another interesting remark is that whereas in LP some cycles are not "problematic" due to the program shape, when we use $\mathcal{P}$ rules this will usually depend on the initial state and the performed actions. For instance, the logic program:

$$\begin{array}{lcl} p & \leftarrow & not\ p \\ p & & \end{array}$$

contains a negative cycle that can be ignored thanks to the program fact p. However, we will see that in most of the examples we will study, the same causal cycle may be sometimes ignored like the LP cycle above (for instance, the initial state contains some fact like p that directly solves the problem), but may lead to problems when we study a different transition. In other words, we must actually bear in mind that the representation of a domain as $\mathcal{P}$-rules is actually a *set of logic programs*, one for each possible transition. Thus, a "static" study of the cycle, like in LP, is not so easy, as we must bear in mind all the possible transitions that may become affected.

7.1.2 Self-supportedness

A first important effect that may follow from the interpretation of a causal cycle is *self-supportedness*, that is, using LP terminology, the non-satisfaction of *well-supportedness*. As we saw in chapter 2, we informally say that a model of a logic program is well-supported iff for any concluded atom p, we can construct a rule-application chain starting from facts and which *does not contain* p itself. Thus, self-supportedness means that we may obtain models which, for explaining some p, we have used p itself or, in other words, p is auto-justified. We also saw in chapter 2 that stable models and WFS are well-supported. Therefore, we will specially focus here on circumscription and Clark's completion, analyzing examples of occurrence of self-supportedness induced by a causal cycle.

As a first example, we may use the already introduced R_1. Consider the case $\sigma_0 = \{\overline{b}\}$ and $\alpha_1 = \emptyset$, that is, we let pass one situation without performing any action. Three semantics lead to the same result: circumscription, stable models and WFS. In these three cases, fluent b persists unchanged. The explanation, in the case of stable models and WFS, is that although R_1 contains a negative cycle, it actually depends on a positive one: $pert(b,\mathtt{p},1)$ always depends on $pert(b,\mathtt{p},1)$ itself, and so may never become well-supported. This "solves" the negative cycle by making $pert(b,\mathtt{p},1)$ always false, and so, $pert(b,\mathtt{n},1)$ always true, due to rule (7.4).

The difference appears when using Clark's completion which, apart from the model we obtained with the other three semantics, it also yields one more model in which b becomes true and pertinent: $holds(b,\mathtt{t},1)$ and $pert(b,\mathtt{p},1)$. The reason for this is that, under Clark's completion, we may first freely assume that b is pertinent and true. This makes the rule bodies for (7.2) and (7.3) true, and then we obtain again that b is pertinent and true (this informal reasoning corresponds to the selection of fixpoints of operator T_P).

This kind of reasoning is analogous to the understanding of the picture that illustrated the beginning of this chapter (figure 7.1). Escher's drawings are famous because of showing impossible configurations (either spatial or, like in this case, causal). Somehow, both hands "come to life" although each one is drawn (caused) by the other.

Coming back to our example, notice that the completion interpretation of R_1 directly violates postulate P1, since there is no external intervention that justifies the pertinence of b. In fact, the absence of pertinent atoms is the reason why circumscription does not lead in this case to self-supportedness. Unfortunately, the example can be easily modified to achieve a similar situation for the circumscription case.

Example 8 Let R_2 be the set of $\mathcal{P}$-rules:

$$b \textbf{ if } b \wedge a \tag{7.6}$$

□

Apparently, executing $\alpha_1 = \{a\}$ in $\sigma_0 = \{\overline{b}\}$ should behave in a similar way. However, note that now, action a "triggers" the pertinence of the causal rule condition. Thanks to property (15), the translation of 7.6 amounts to:

$$\begin{array}{rcl} holds(b, \mathtt{t}, 1) & \leftarrow & holds(b, \mathtt{t}, 1),\ holds(a, \mathtt{t}, 1) \\ pert(b, \mathtt{p}, 1) & \leftarrow & holds(b, \mathtt{t}, 1),\ holds(a, \mathtt{t}, 1) \end{array}$$

From α_1 and σ_0 we include the program facts $holds(a, \mathtt{t}, 1)$ and $holds(b, \mathtt{f}, 0)$. While both stable models and WFS provide again the same answer (b persists false), both completion and circumscription lead to an additional model in which b is caused to be true. Notice that, although in this case postulate P1 is not violated (there is an external intervention due to action a), an atom is still not self-supported, since $holds(b, \mathtt{t}, 1)$ is needed in the condition of the rule we used to derive it.

7.1.3 Nonexistence of successor state

From a causal point of view, self-supportedness does not seem to be very interesting and, in fact, is one of the main disadvantages of circumscription and Clark's completion techniques, when applied to pertinence calculus. For this reason, we center the rest of the discussion on stable models and WFS, although some of the effects we analyze from now on also occur in completion and circumscription. For instance, in the case of stable models, one of these singular effects is the absence of successor state, due to the absence of stable model for some logic program.

Consider again R_2 but using now the initial state $\sigma_0 = \{b\}$, that is, we include the observation $holds(b, \mathtt{t}, 0)$ instead of $holds(b, \mathtt{f}, 0)$. At a first glimpse, this cycle does not seem to lead to any problem: when b was true, we should be able to execute a "overriding" the value of b. However, due to the postulates of pertinence, allowing such behavior *would not leave clear* whether b persisted or was caused true (note that, in order to become pertinent, we must assume first that b persists true). As a result, the corresponding program has no stable model. It is easy to see that, due to the particular initial state and action execution, we have finally forced a negative LP cycle in the pertinence of `b`.

Just as a remark, the answer for this transition in the cases both of completion and circumscription is that we obtain a unique model: b is caused true, but becomes self-supported.

The whole problem would disappear if we had, instead:

Example 9 Let R_3 be the set of $\mathcal{P}$-rules:

$$b \quad \textbf{if} \quad b \wedge a \tag{7.7}$$
$$b \quad \textbf{if} \quad a \tag{7.8}$$

□

In this case, the new rule would make b to be directly pertinent whenever a is performed, and so, there would be no interference with inertia.

As pointed out by example R_2, we may have now a *new possible reason for nonexecutability of actions*: the nonexistence of model. When R was acyclic, the nonexistence of successor state was always due to obtaining an inconsistent set of facts, either by assigning different values to the same fluent or by applying a rule with $\perp$ head. We had a more explicit representation of impossible action executions. In other words, we could always explain to a programmer why the action is not executable: either some particular constraint has been violated, or some fluent, due to some particular rules, is assigned two different values. Once cycles are allowed, the stable models interpretation (and in fact, the same applies for circumscription and Clark's completion) may lead to the nonexistence of successor state due to an "ill-defined" construction conflicting with inertia. Thus, the nonexecutability is somehow hidden and it can be objected that it is more difficult to be detected. For instance, note that R_2 does not contain any $\perp$ head, and that it only causes b to have one value ($\mathtt{t}$), not the other ($\mathtt{f}$).

Of course, we can consider that the introduction of a cycle like R_2 actually had an intentioned purpose, trying to represent that a is nonexecutable when b was true. But there exist more reasons to think about it as a wrong formalization. Had the intention been that one, it would have been far more clear to formulate the rule as:

$$\perp \quad \textbf{if} \quad a \textbf{ after } b$$

since, as we have seen, (b **if** $b \wedge a$) behaves in different ways depending on the previous state or on the presence of other rules. Besides, from the programmer's point of view, if we look for the explanation for a nonexecutable action and we deal with a large system, we will not know a priori whether this was due to a constraint we introduced, the assignment of different values to the same fluent or a cyclic reference, which may be almost hidden, affecting a great number of rules and fluents. It seems more probable that, in such cases, the occurrence of the cyclic reference was something unexpected constituting a mistake in our representation, rather than an intentioned way of expressing nonexecutability.

The orientation followed by WFS is, in fact, closer to this last idea. When $\sigma_0 = \{b\}$ we face the same problem as in the stable models case: it is not possible neither to assume pertinence nor persistence of b. However, the result is different: instead of having no model, we obtain an incomplete WFM where only those atoms actually involved (or affected) in the ill-defined cycle are left undefined. Thus, while atoms $holds(b, \mathtt{t}, 1)$, $pert(b, \mathtt{p}, 1)$ and $pert(b, \mathtt{n}, 1)$ are left undefined, the truth of $holds(a, \mathtt{t}, 1)$ is guaranteed, as well as the falsity of $holds(b, \mathtt{f}, 1)$ (we know, at least, for sure, that b cannot become false). From a programmer's point of view, the well founded model offers a more detailed description of the anomalous cyclic dependence.

7.1.4 Nondeterminism

A very important feature that may appear as a result of causal cycles is the availability of different successor states for a same transition. In fact, we already obtained nondeterminism when we studied completion and circumscription for examples R_2 and R_3. We will see now that this may also happens for stable models, but not for WFS, which again makes use of undefined atoms.

Example 10 Let R_4 be the set of $\mathcal{P}$-rules:

$$b \quad \textbf{if} \quad \overline{c} \wedge a \tag{7.9}$$
$$c \quad \textbf{if} \quad \overline{b} \wedge a \tag{7.10}$$

□

The resulting program $P(R_4)$ contains, among other, the program rules:

$$holds(b,\mathtt{t},1) \leftarrow holds(c,\mathtt{f},1),\ holds(a,\mathtt{t},1) \tag{7.11}$$
$$pert(b,\mathtt{p},1) \leftarrow holds(c,\mathtt{f},1),\ holds(a,\mathtt{t},1) \tag{7.12}$$
$$holds(c,\mathtt{t},1) \leftarrow holds(b,\mathtt{f},1),\ holds(a,\mathtt{t},1) \tag{7.13}$$
$$pert(c,\mathtt{p},1) \leftarrow holds(b,\mathtt{f},1),\ holds(a,\mathtt{t},1) \tag{7.14}$$
$$pert(b,\mathtt{n},1) \leftarrow not\ pert(b,\mathtt{p},1) \tag{7.15}$$
$$pert(c,\mathtt{n},1) \leftarrow not\ pert(c,\mathtt{p},1) \tag{7.16}$$
$$holds(b,\mathtt{t},1) \leftarrow holds(b,\mathtt{t},0),\ pert(b,\mathtt{n},1) \tag{7.17}$$
$$holds(b,\mathtt{f},1) \leftarrow holds(b,\mathtt{f},0),\ pert(b,\mathtt{n},1) \tag{7.18}$$
$$holds(c,\mathtt{t},1) \leftarrow holds(c,\mathtt{t},0),\ pert(c,\mathtt{n},1) \tag{7.19}$$
$$holds(c,\mathtt{f},1) \leftarrow holds(c,\mathtt{f},0),\ pert(c,\mathtt{n},1) \tag{7.20}$$

Again, if we execute action a, different effects are obtained depending on the initial state. For instance, when the initial state is $\sigma_0 = \{b, c\}$, there is no interference between inertia and causal rules, since we do not have any reason for assuming b or c false. In fact, the four semantics coincide in this case: we simply obtain that b and c persist true and this directly disables both rules. Assume now that at the initial situation, one of the fluents was false, for instance $\sigma_0 = \{b, \overline{c}\}$. We still have no reason for using rule (7.10), and so c may persist false, making b true and pertinent. Again, the four semantics provide this unique model. However, when $\sigma_0 = \{\overline{b}, \overline{c}\}$, it is impossible to assume the persistence of both fluents, since this would make rules (7.9) and (7.10) to be applied and we would get as a result that b and c are pertinent. However, we can consider each of the two cases separatedly. Preferring first the persistence of any of the two fluents implies that the other becomes true and pertinent. These two possible models are obtained in all the cases, excepting WFS. The well founded model for the corresponding program leaves all the atoms referring to b and c undefined. The explanation for this is that WFS always provides a unique model (i.e., a deterministic answer) and so, it cannot choose a preference for the persistence of b nor c.

As happened with the possible use of nonexistence of model to represent nonexecutability, we can think that the existence of multiple models may be used to represent nondeterministic systems. In this case, nondeterminism is a new property we have not provided for the operational semantics, nor in the shape of causal rules. So, this would be interesting, for it means an increase in expressivity. However, similar objections may be risen as we did with the nonexistence of model. A first important problem is that the nondeterministic behavior of causal cycles *depends on the previous situation*, as we saw with all the possibilities for R_4. This seems to make difficult the representation of a nondeterministic effect which does not depend on the previous state. A second problem is again the possible appearance of a cycle in a large set of rules. Notice that the cycle may be "well-behaved" in most of the transitions. In fact, it may be the case that, due to the constraints or the shape of the rules, it never leads to nondeterminism. However, when several successor states appear, it is quite improbable that it was something intentionally expected, but seems instead an "uncontrolled case" we had not expected to occur.

7.1.5 Cumulativity and cycle ordering

Another feature that may occur in the presence of cycles is due to non-cumulativity of the stable models semantics which, as we explained in chapter 2, seems to be related to a unordered (with respect to the level function) interpretation of cycles. Consider the following example (analogous to program P_5 in chapter 2):

Example 11 Let R_5 be the set of $\mathcal{P}$-rules:

$$b \quad \textbf{if} \quad \overline{c} \wedge a \tag{7.21}$$
$$c \quad \textbf{if} \quad \overline{b} \wedge a \tag{7.22}$$
$$d \quad \textbf{if} \quad \overline{c} \wedge a \tag{7.23}$$
$$d \quad \textbf{if} \quad \overline{d} \wedge \overline{b} \wedge a \tag{7.24}$$

□

If we consider the initial state σ_0 in which all fluents are false and we perform $\alpha_1 = \{a\}$, we obtain a unique stable model in which both b and d become caused true, whereas c persists false. Notice that there exist two cycles in this domain: one between b and c due to rules (7.21) and (7.22); and the other for d in the single rule (7.24). In order to solve the cycles, one could expect that a layering criterion could be applied: as c and b do not depend on d, we could try to solve first the cycle (7.21)-(7.22), and then proceed to reason for d, which depends on b and c. The stable models interpretation somehow "violates" this ordering: informally, we first use cycle (7.24) to conclude that $\overline{b}$ cannot be true (if so, we get into a problem with d). Then, we use this constraint to prune one of the two possible stable models that would arise from cycle (7.21)-(7.22) stand-alone (as we saw with R_4).

The real problem of this behavior is that the addition of apparently harmless rules may lead to strange effects. Thus, since d is obtained from performing a in σ_0, one could expect that adding a rule that expresses this relation explicitly should not vary the obtained result.

Example 12 Let R_6 be the union of R_5 plus the $\mathcal{P}$-rule:

$$d \quad \textbf{if} \quad a \textbf{ after } \overline{b} \wedge \overline{c} \wedge \overline{d} \tag{7.25}$$

□

However, we obtain now for the same transition *two different stable models*: the one obtained before plus one in which both c and d are caused true, whereas b persists false.

Contrarily to stable models, the WFS interpretation is modular and ordered with respect to the dependences among fluents. In this way, in order to solve R_5, we can first decide cycle (7.21)-(7.22) that, as in R_4, leaves b and c undefined. After that, as d depends on b and c

and it is also involved in a cycle, it is left undefined. In this way, WFS already warns about a strange construction in the cycles of R_5. As for R_6, the addition of (7.25) immediately solves the undefinition for d, but the cycle between b and c remains as before. As a result, we get b and c undefined, but d becomes now caused true.

7.1.6 Coherence

From the previous examples, it seems that the most reasonable interpretation for cycles is provided by WFS, since when a cycle is not "solvable," at least the WFM points out which fluents (and furthermore, which fluent values) originated the problem. However, apart from the fact of leaving some fluents undefined, which does not have a clear correspondence with respect to a real domain, WFS may also lead to unnatural results due to the *coherence problem* commented in section 2.3.6, unless we use the variation of WFSX.

To illustrate how the coherence problem arises also in $\mathcal{P}$-rules, consider first the following example.

Example 13 Let R_7 be the set of $\mathcal{P}$-rules:

$$\begin{array}{lll} b & \textbf{if} & b \wedge a \\ c & \textbf{if} & b \wedge a \\ d & \textbf{if} & c \wedge a \end{array}$$

$\Box$

If we use initial state $\sigma_0 = \{b, c, d\}$ and perform action $\alpha_1 = \{a\}$, the solution for this example is straightforward, provided that we already studied cycle (7.26) in example R_2 for a similar transition. Thus, b is left undefined and so, since c depends on b via (7.26) and, in its turn, d on c via rule (7.26), we get that all the fluents become undefined. Now, let us modify R_7 by adding one more rule:

Example 14 Let R_8 be the union of R_7 plus the $\mathcal{P}$-rule:

$$\overline{c} \quad \textbf{if} \quad a$$

$\Box$

If we consider the same transition, the addition of this rule directly fixes fluent c to be caused true. As a result, fluent d should also be solved, since we are sure now that rule (7.26) *cannot be applied*, because its condition *is false*. However, as usual WFS does not establish any connection between atoms c and $\overline{c}$, we actually obtain that $\overline{c}$ is founded but c remains undefined.

This problem is directly solved if we consider WFSX instead of usual WFS. The main feature introduced by WFSX is the application of coherence: whenever an atom is fixed to be true its explicit negation is fixed as false. Thus, condition of rule (7.26) becomes false, and d persists true.

It is interesting to remind that we have shown that WFSX can be characterized as a set of rewriting rules that includes all those needed for WFS, as presented in [17], plus the two transformations we called coherence failure and coherence reduction. We recall their definitions:

- *Coherence failure* $\stackrel{\mathtt{C}}{\longmapsto}$: delete all rules containing the positive literal $\overline{p}$ such that p is a fact.

- *Coherence reduction* $\stackrel{\mathtt{R}}{\longmapsto}$: delete all literals *not* $\overline{p}$ such that p is a fact.

Coherence reduction is not actually needed for our logic programs for pertinence: the only default negations *not* $pert(P, \mathtt{p}, I)$ are in the body of (6.49), but the opposite atom, $pert(P, \mathtt{n}, I)$, is not head of any rule, excepting (6.49) itself.

The problem of WFSX is that it is exclusively thought for boolean atoms, whereas fluents and actions can be multivalued. In this way, we must actually adapt the WFSX transformation rules to cope with multiple values. Fortunately, this adaptation is quite straightforward, simply redefining:

- *Coherence failure* $\stackrel{\mathtt{C}}{\longmapsto}$: delete all rules containing $holds(p, v, i)$ (resp. $pert(p, v, i)$) such that $holds(p, v', i)$ (resp. $pert(p, v', i)$) is a program fact, where $v \neq v'$.

7.1.7 Summary

To sum up, we have shown that both the circumscriptive approach and the interpretation based on completion do not satisfy well-supportedness, allowing that some effect becomes the only cause for its own occurrence. Only stable models and WFS satisfy well-supportedness. However, the stable models focusing may provide multiple models or nonexistence of model. We claim that both situations seem to point out a lack of representativity rather than a way for expressing nondeterminism or nonexecutability. It must be clearly understood that we are not claiming that these effects of negative cycles are not interesting for their use in logic programming. In fact, by an appropriated formulation of cycles with stable models, it is possible to construct logic programs with a comfortable constraint solving orientation, as shown in section 6.6.2. The problem here is that these effects *arise from the translation of causal rules*, and not as an intended work of representation.

It can be objected that the problem actually relies on the translation $P(R)$ and that perhaps, with another encoding into stable models, these effects for causal cycles disappear. This could be possible, but note that: (1) the translation seems perhaps the most natural one, using inference rules in most cases and leaving the default negation exclusively for minimizing pertinence; and (2), as a hint of being in the right direction, the encoding $P(R)$ corresponds in a direct way to the operational semantics when the set of causal rules is acyclic.

Finally, we could think about a radical solution to these problems: consider that these "strange cycles" never occur when representing a real world domain. However, the next section presents a pair of domains where cyclic dependences seem to be present in the physical system.

7.2 Domains with cyclic dependences

Thinking about a cyclic causal dependence may seem unnatural. However, we will show, with a pair of examples, that depending on the level of abstraction we use for representing the problem, causal cycles may correspond to a natural representation of a real system.

A typical example of a causal cycle is the domain presented in [72, 28] dealing with a pair of gear wheels.

Example 15 The gear wheels We have a gear mechanism with a pair of wheels. Turning (resp. stopping) one wheel makes also move (resp. stop) the other one. □

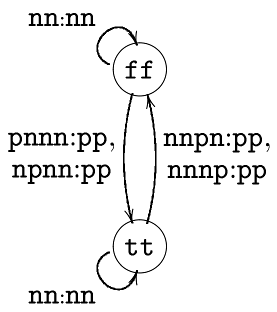
This domain can be represented using the actions $\mathcal{A}_w = \{start(1), start(2), stop(1), stop(2)\}$, fluents $\mathcal{F}_w = \{turn(1), turn(2)\}$ and causal rules R_w:

$$turn(1) \quad \textbf{if} \quad start(1) \tag{7.26}$$
$$turn(2) \quad \textbf{if} \quad start(2) \tag{7.27}$$
$$\overline{turn(1)} \quad \textbf{if} \quad stop(1) \tag{7.28}$$
$$\overline{turn(2)} \quad \textbf{if} \quad stop(2) \tag{7.29}$$
$$turn(1) \quad \textbf{if} \quad turn(2) \tag{7.30}$$
$$turn(2) \quad \textbf{if} \quad turn(1) \tag{7.31}$$
$$\overline{turn(1)} \quad \textbf{if} \quad \overline{turn(2)} \tag{7.32}$$
$$\overline{turn(2)} \quad \textbf{if} \quad \overline{turn(1)} \tag{7.33}$$

We assume nonconcurrent actions and that we rule out undesired initial states with the constraint:

$$turn(1) \equiv turn(2)$$

If we interpret the rules (7.26)-(7.33) using stable models, WFS or the circumscriptive encoding, we obtain the machine depicted in figure 7.2, which at a first glimpse seems to capture the intuitive behavior. However, using Clark's completion, we obtain a different transition relation, which, in fact, is nondeterministic. Thus, for instance, when both wheels are not turning, if we perform no action, we obtain two possible successor states: one in which everything remains unchanged (as happened with the other semantics), plus an additional one in which both wheels begin turning without external intervention! The explanation of this effect is again due to the lack of well-supportedness: the wheels movements mutually justify one each other.



$start(1)\ start(2)\ stop(1)\ stop(2) : turn(1)\ turn(2)$

Figure 7.2: Gear wheels example.

Of course, we can still solve the problem inside the Clark's completion interpretation. For instance, we can consider one of the fluents (e.g., turn(1)) as primary, defining the other fluent as a simple synonym. In this way, only $turn(1)$ would follow inertia, whereas $turn(2)$ would simply get its truth value from $turn(1)$, using the constraint: $turn(2) \equiv turn(1)$. However, it is easy to see that this solution does not reflect the physical organization of the wheels, and we can always further elaborate the system description so that this equivalence is not valid. To emphasize the difference between both fluents, consider that we can now couple and uncouple the wheels, handling a mechanism represented in figure 7.3. When both wheels are uncoupled, they can be managed independently, whereas only when they are coupled, the causal cycle is really effective.

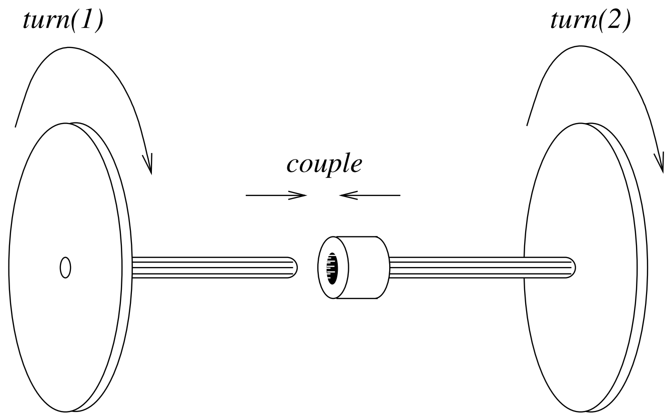


Figure 7.3: A device for coupling gear wheels.

We define $\mathcal{A}_c = A_w \cup \{couple, uncouple\}$ and $\mathcal{F}_c = \mathcal{F}_w \cup \{coupled\}$, using, as a first temptative, the set of causal rules R_c:

$$\begin{array}{lll}
turn(1) & \textbf{if} & start(1) \\
turn(2) & \textbf{if} & start(2) \\
\overline{turn(1)} & \textbf{if} & stop(1) \\
\overline{turn(2)} & \textbf{if} & stop(2) \\
coupled & \textbf{if} & couple \\
\overline{coupled} & \textbf{if} & uncouple \\
turn(1) & \textbf{if} & turn(2) \wedge coupled \\
turn(2) & \textbf{if} & turn(1) \wedge coupled \\
\overline{turn(1)} & \textbf{if} & \overline{turn(2)} \wedge coupled \\
\overline{turn(2)} & \textbf{if} & \overline{turn(1)} \wedge coupled
\end{array}$$

These rules lead to a problem when coupling the wheels when at least one of them was turning. For instance, when $\sigma_0 = \{turn(1), \overline{turn(2)}, \overline{coupled}\}$ and $\alpha_1 = \{couple\}$ we get no stable model, whereas the WFM leaves all the fluent values undefined (excepting truth for *coupled*, which is founded, and falsity for $turn(1)$, which is unfounded). The intuitive explanation is that if we assume, for instance, that $turn(1)$ persists true, we can apply rule (7.34) and get $turn(2)$ true and pertinent. But then, rule (7.34) would be applied making $turn(1)$ not persistent, which is the opposite of what we had assumed. Something similar happens assuming first that $turn(2)$ persists false.

It seems clear that the problem arises because a lack of formalization: we have not specified how to couple the wheels at that situation. We may consider, for instance, that both wheels are caused to stop when coupled, adding the rules:

$$\overline{turn(1)} \quad \textbf{if} \quad couple \tag{7.34}$$

$$\overline{turn(2)} \quad \textbf{if} \quad couple \tag{7.35}$$

Another option is to require instead, as action precondition, that the wheels must be stopped before coupling:

$$\begin{array}{lll} \bot & \textbf{if} & \mathit{couple}\ \textbf{after}\ turn(1) \\ \bot & \textbf{if} & \mathit{couple}\ \textbf{after}\ turn(2) \end{array}$$

Note that, now, we disqualify action *couple* by explicitly provoking an inconsistence, rather than by leading to the nonexistence of stable model. Unfortunately, this solution needs extra formalization, since we still have problems when coupling two stopped wheels. In such a case, we could neither assume persistence of any wheel (rules (7.34) and (7.34) would finally make both $turn(1)$ and $turn(2)$ pertinent) nor pertinence (since it would always depend on itself, being not well-supported), and so, there is no stable model (and so, the WFM is incomplete). This additional problem arises because it seems that rules (7.34)-(7.34) should only be applicable due to some wheel movement and not just by coupling the wheels. A possible solution is, as we did in the alarm problem, to consider that *coupled* must not only be true, but also persistent. Thus, we reinforce the rules by explicitly requiring nonpertinence of *coupled*:

$$\begin{array}{lll} turn(1) & \textbf{if} & turn(2) \wedge \mathit{coupled} \wedge \overline{!\mathit{coupled}} \\ turn(2) & \textbf{if} & turn(1) \wedge \mathit{coupled} \wedge \overline{!\mathit{coupled}} \\ \overline{turn(1)} & \textbf{if} & \overline{turn(2)} \wedge \mathit{coupled} \wedge \overline{!\mathit{coupled}} \\ \overline{turn(2)} & \textbf{if} & \overline{turn(1)} \wedge \mathit{coupled} \wedge \overline{!\mathit{coupled}} \end{array}$$

Many other possible solutions are imaginable. For instance, we could choose to give one of the wheels more priority than the other so that, in the exact moment in which they are coupled, the former governs the latter. In any case, the possibility for all these options seems to point out that the nonexistence of stable model (or the incompleteness of the WFM) for the initial set of rules was due to a lack in the formalization: we had not clearly specified the system behavior in all the possible cases.

It is still possible to imagine a solution for this problem using completion. In chapter 8 we will study a generalization of Clark's completion called the Logic of Causal Explanation. A possible solution relying on this logic[1] is to define the rule $(turn(1) \equiv turn(2)) \leftarrow \mathit{coupled}$, using the equivalence as the head of the rule. However, it is easy to think about new elaborations of the system in which the need of an actual causal cycle becomes evident. For instance, it could be the case that we are initially said that only one wheel controls the other, but later we must update the description so that one affects each other. It would not be very reasonable to be forced to decide, for each new possible causal cycle, whether the old rules must be replaced by a new rule with an equivalence as head or not.

As a second example of "cyclic scenario," consider the circuit[2] in figure 7.4 introduced by Shanahan in [103]. The circuit contains a lamp, a relay and three switches. The interest of this circuit is that it shows a vicious cyclic dependence which corresponds to a real domain. For instance, when we close switch 1 at the state depicted in the figure, the relay must become activated but this, in its turn, should open switch 2, and so, the cause for activating the relay is retracted. Something similar happens when closing switch 2 while switch 1 is closed, so that the action becomes nonexecutable. Of course, one may argue that if we consider intermediate delays, reducing the level of abstraction, the vicious cycle disappears. In this way, we would

[1]Suggested by Lifschitz during a discussion

[2]This circuit is in fact a modification of one proposed by Thielscher [107] we will study later in chapter 8.

be replacing causal knowledge by a chain of delayed effects [3]. However, the interest of causal knowledge relies precisely in that *it provides a powerful abstraction* that avoids a detailed study of the real physical interactions. This abstraction is particularly useful for commonsense reasoning, but paying the price of being unable to cover some unusual configurations, like the one in the figure. In this case, for instance, although the circuit is perfectly possible in the real world, from a "causal" point of view, its purpose is not very clear, and seems to correspond to a faulty design.

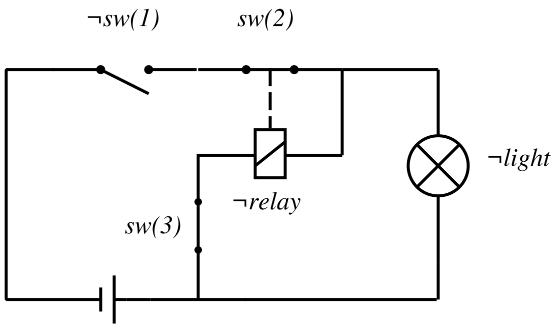


Figure 7.4: Shanahan's relay.

To formalize this domain, we define the set of actions would correspond to:

$$\mathcal{A}_r = \{toggle(1), toggle(2), toggle(3)\}$$

the fluents are:

$$\{sw(1), sw(2), sw(3), relay, light\}$$

and the causal rules would be R_r:

$$\begin{array}{rll}
sw(N) & \textbf{if} & toggle(N) \textbf{ after } \overline{sw(N)} \qquad (7.36)\\
\overline{sw(N)} & \textbf{if} & toggle(N) \textbf{ after } sw(N) \qquad (7.37)\\
light & \textbf{if} & sw(1) \wedge sw(2) \qquad (7.38)\\
\overline{light} & \textbf{if} & \overline{sw(1)} \qquad (7.39)\\
\overline{light} & \textbf{if} & \overline{sw(2)} \qquad (7.40)\\
relay & \textbf{if} & sw(1) \wedge sw(2) \wedge sw(3) \qquad (7.41)\\
\overline{relay} & \textbf{if} & \overline{sw(1)} \qquad (7.42)\\
\overline{relay} & \textbf{if} & \overline{sw(2)} \qquad (7.43)\\
\overline{relay} & \textbf{if} & \overline{sw(3)} \qquad (7.44)\\
\overline{sw(2)} & \textbf{if} & relay \qquad (7.45)
\end{array}$$

for all $N \in \{1, 2, 3\}$. When we perform action $\alpha_1 = \{toggle(1)\}$ after state:

$$\sigma_0 = \{\overline{sw(1)}, sw(2), sw(3), \overline{light}, \overline{relay}\}$$

[3]This is idea is in fact very common in some interpretations of causality. For instance, we may cite Sandewall's *transition cascade semantics* [98], Pinto's approach [90, 91] or even Thielscher's causal rules [107], although in this case, the causal transitions are handled at a different level than the state transitions.

we get that the resulting program has not any stable model, whereas the well founded model is incomplete, leaving all the atoms for $sw(2), relay$ and $light$ undefined. This intuitively points out that there exist a problem involving these three fluents. Assuming that $sw(2)$ persists true means that the $relay$ is activated and that in its turn $sw(2)$ gets open, refuting the assumption. Notice that the well founded model guarantees that $sw(1)$ is made true and pertinent. This is important, because it allows, at least, to fix which fluent values can be determined without being affected by the vicious cycle. Thus, we can see how $light$ is left undefined because it depends on the cycle affecting $sw(2)$, whereas $sw(1)$ can be fixed because it has no causal dependence on the other fluents.

As happens with stable models, completion and circumscription also lead to the nonexistence of any selected model. Notice again how the well founded model provides a more detailed description of the origin of the problem.

Chapter 8

Comparison to other action approaches

This chapter contains a comparative study with respect to a wide group of action approaches, including not only the most representative ones, but also some specific solutions which show interesting features for our purposes. Although not all the presented approaches have been explicitly named as causal, most of them actually deal with a predicate for expressing change, and so can be classified as change-based causality, as we discussed in the introduction.

The study will be particularly focused on how inertia and causality are respectively represented and combined in all these approaches. In fact, most of them have similarities with pertinence postulates, but do not satisfy them completely. As a result, it is often not possible to use the change predicate to express accurately that the fluent has persisted or not, since there is no clear distinction between inertia and causation.

In order to present the original formulations[1], we will indistinctly handle narrative and time-branching approaches, but this difference is not essential for our purposes. We will use both letters S or I to denote situations. In most cases, fluents are boolean and the *holds* predicate has only two arguments, so that, with respect to our notation:

$$\begin{aligned} holds(F, S) &\equiv holds(F, \mathtt{t}, S) \\ \neg holds(F, S) &\equiv holds(F, \mathtt{f}, S) \end{aligned}$$

Besides, the reification of formulas inside the *holds* predicate will be sometimes used:

$$\begin{aligned} holds(\neg\phi, S) &\stackrel{\text{def}}{=} \neg holds(\phi, S) \\ holds(\phi \wedge \psi, S) &\stackrel{\text{def}}{=} holds(\phi, S) \wedge holds(\psi, S) \\ holds(\phi \vee \psi, S) &\stackrel{\text{def}}{=} holds(\phi, S) \vee holds(\psi, S) \end{aligned}$$

As examples for effect axioms and ramification constraints, we will extract formulas from two well known already studied scenarios: Yale Shooting and Lin's suitcase. Together with these two typical domains, we will also handle the account balance example presented in the introduction (example 2) in order to emphasize the interest of pertinence postulates, helping to detect which ones are violated by each approach.

[1] For coherence sake, we have changed the use of capital letters to denote variables, as in the rest of this thesis.

8.1 Change as abnormality

The first case of change-based predicate under study is the abnormality predicate, ab, extensively used in the first formalizations of situation calculus. Although it does not correspond to a real causal approach, nor is used for dealing with the ramification problem, it is interesting because it constitutes the root of the first representations of inertia. For instance, in Baker's classical work [8] (one of the first solutions to the Yale shooting problem) we can find the following expressions:

$$\neg ab(F, A, S) \supset \big(holds(F, do(A, S)) \equiv holds(F, S)\big) \tag{8.1}$$

$$holds(loaded, S) \supset \neg holds(alive, do(shoot, S)) \tag{8.2}$$

which correspond respectively to the universal frame axiom and an example of effect axiom. It can be easily seen that this formalization does not satisfy postulate P2: sometimes, facts can be explained both by inertia and by a causal process. For instance, if we shoot a loaded gun when the turkey is already dead, there is no need to conclude $ab(alive, shoot, S)$ (predicate ab is minimized). In this way, we can explain the fact $\neg holds(alive, do(shoot, s_0))$ both as the result of (8.2) and as the application of (8.1), since $alive$ is not abnormal in this situation.

As we can see, this formalization understands abnormality exclusively as a change of truth value in a fluent. Of course, for the original aim of this formulation, the actual extent of predicate ab was not relevant at all. The only interest was focused on the resulting extent of $holds$. However, it is interesting to note that, in the initial formulation of the Yale Shooting scenario [48], the effect axiom is actually represented as:

$$holds(loaded, S) \supset \neg holds(alive, do(shoot, S)) \land ab(alive, shoot, S) \tag{8.3}$$

which does not eventually affect the extent of $holds$, but seems an attempt of preserving postulate P2 and using ab as a "causal" predicate. Under this second formulation, the shot to a dead turkey would lead to $ab(alive, shoot, S)$ and so *only the effect axiom* would explain the fact $\neg holds(alive, do(shoot, s_0))$.

8.2 Lin's *caused* predicate

Lin's approach [65] constituted the first successful solution to the frame and ramification problems that relied on a minimization technique for a change-based predicate: $caused(F, V, S)$. Since $caused$ contains the fluent truth value reified, the following two axioms are added:

$$caused(F, \mathtt{t}, S) \supset holds(F, S) \tag{8.4}$$

$$caused(F, \mathtt{f}, S) \supset \neg holds(F, S) \tag{8.5}$$

The universal frame axiom has the shape[2]:

$$\neg caused(F, \mathtt{t}, S) \land \neg caused(F, \mathtt{f}, S) \supset [holds(F, do(A, S)) \equiv holds(F, S)] \tag{8.6}$$

the effect axioms are expressed as:

$$holds(up(L), S) \quad \supset \quad caused(up(L), \mathtt{f}, do(toggle(L), S)) \tag{8.7}$$

$$\neg holds(up(L), S) \quad \supset \quad caused(up(L), \mathtt{t}, do(toggle(L), S)) \tag{8.8}$$

[2]For a better comparison, we have omitted the $Poss$ predicate.

for each lock $L \in \{1, 2\}$, and finally, as example of ramification rule, we have:

$$holds(up(1), S) \wedge holds(up(2), S) \supset caused(open, \mathtt{t}, S) \tag{8.9}$$

In order to avoid unnatural models, Lin conceived the following minimization policy, based on filter preferential entailment (minimizing only a part of the theory):

1. Let T' be the result of minimizing *caused* in the whole theory T *excepting the universal frame axiom.*

2. Add to T' the universal frame axiom (8.6).

As we can see, this is exactly the technique we have followed in section 6.4 for circumscribing our pertinence predicate, *pert*, and, in fact, this same idea was also applied to other circumscription-based approaches [47, 103].

There exists a strong resemblance between Lin's formulation and pertinence calculus, specially when using the circumscriptive encoding. It is interesting to see, for instance, how Lin's *caused* predicate satisfies some of the pertinence postulates like:

P2) when a fluent value is obtained, it is either due to a causal rule or to inertia, but not both (the frame axiom (8.6) has as explicit prerequisite that the fluent is not caused)

P3) any change in a fluent value is always due to some causal chain (this corresponds exactly to the contrapositive reading of (8.6))

P4) *caused* does not necessarily mean that the fluent value has changed

The main differences are, however:

(a) Unlike *caused*, the pertinence predicate does not refer to the truth value of a fluent.

(b) Pertinence is also defined for actions, and furthermore, for complex formulas.

(c) In Lin's approach, conditions of causal rules, like (8.9), are just required to be true. They do not necessarily contain *caused* atoms (postulates P5 is not satisfied and, as a result, neither P1).

The first difference, (a), means that the inclusion of the fluent value inside the *caused* predicate is not really necessary, since pertinence calculus obtains an equivalent effect without using that representation. Notice that, although for its logic programming use, we have reified the pertinence value, $V \in \{\mathtt{p}, \mathtt{n}\}$, inside pertinence atoms, we had seen how we may actually use instead a binary predicate $pert(P, V)$. In any case, pertinence atoms *do not refer* to the fluent value at all, being predicate *holds* exclusively used to that purpose. All this explanation suggests that pertinence calculus provides a more economic representation, in the sense that we do not need to repeat the fluent value (which could be nonboolean) both in *caused* and *holds*, and so axioms (8.4) and (8.5) are not necessary any more in pertinence calculus. Furthermore, we had already seen (6.17) how *caused* can be defined in terms of *pert* inside pertinence calculus:

$$caused(p, v, i) \stackrel{\text{def}}{=} holds(p, v, i) \wedge pert(p, \mathtt{p}, i)$$

Of course, the opposite can also be done, defining *pert* inside Lin's approach:

$$pert(F, \mathtt{p}, I) \stackrel{\text{def}}{=} caused(F, \mathtt{t}, I) \vee caused(F, \mathtt{f}, I)$$

or, generally:

$$pert(F, \mathtt{p}, I) \stackrel{\text{def}}{=} \exists V.\, caused(F, V, I)$$

for any $V \in range(F)$.

The second difference (b) is not fundamental, but provides a more comfortable description of causal rules. The definition of pertinence for actions is particularly interesting because of handling a narrative approach allowing concurrent actions. Notice that without this feature, the nonoccurrence of an action a at a situation i would need to be represented as the negation of all its possible values:

$$\forall V.\, \neg holds(a, V, i)$$

Thanks to axiom (6.5), we can simply use negative pertinence $pert(a, \mathtt{p}, i)$ to point out nonocurrence of the action. Lin's formulation was specifically thought for situation calculus, and so, it would need to be carefully adapted for concurrent actions[3].

To see the utility of pertinence of actions, consider the typical example (from [9]) for testing a appropriated representation of effects from concurrent actions:

Example 16 (**The soup bowl**) *Whenever Mary tries to lift the bowl with one hand, she spills the soup. When she uses both hands, she does not spill the soup. We know that the soup is not spilled initially.*

On the one hand, in order to represent the domain in a modular way, we should avoid referring to all the performed actions in any causal rule. On the other hand, we need to represent that a given action has not occurred: for example, when we lift the right side *but not* the left side, or vice versa. Assuming we are just interested in actions $\mathcal{A} = \{lift(left), lift(right)\}$ and fluents $\mathcal{F} = \{spilled\}$, this domain can be represented using the $\mathcal{P}$-language rules:

$$\begin{array}{lll} spilled & \textbf{if} & lift(left) \wedge \overline{!lift(right)} \\ spilled & \textbf{if} & lift(right) \wedge \overline{!lift(left)} \end{array}$$

The most important part of difference (b) is the definition of pertinence for complex formulas, not only for fluents or actions. As we had already seen when studying L^2 logic, this provides a very concise high level representation which, in most cases, allows expressing causal rules without explicitly referring to *pert* predicate (or its abbreviation '!'). In fact, this feature is crucial for an adequate implementation of postulate P5, which says that causal rules must be applicable if only if their conditions (which are complex formulas) are pertinent.

Last, but not least, difference (c) is in fact the most fundamental one, since it means that Lin's approach does not necessarily satisfy P5, that is, in the typical encoding of a ramification rule, the condition does not refer to *caused* predicate, but depends exclusively on *holds*. This has an advantage for implementation: in each formula of the circumscribed portion of the theory, all the references to *caused* are either exclusively negative (it is in the antecedents of axioms (8.4) and (8.5)) or exclusively positive (it is in the consequents of effect axioms and ramification rules). Then, by applying standard results in circumscription[4] it is *always* possible to circumscribe *caused* by using Clark's completion.

[3]For studies about concurrent actions inside situation calculus, see for instance [9, 80].

[4]See [58], propositions 3.1.1 and 3.3.1

As a direct example of difference (c), consider the ramification rule (8.9) provided in Lin's original work. It is easy to see that its condition just relies on fluent values (it only contains *holds* atoms). As a result of this violation of P5, there may exist situations in which postulate P1 does not apply. For instance, if both switches are up, each time we execute an action *wait* without effects, we would have that the suitcase *would be caused* to be open, but not as a result of any direct or indirect effect of the action[5] (violation of P1). To avoid this, we should make the rule applicable not only when both locks are up, but also when such a conjunction has been caused. Notice how we need to represent that a conjunction (not a single atom) has been caused. In this sense, it is important to observe that the representation of:

"$holds(up(1), S) \wedge holds(up(2), S)$ is caused"

cannot be simply done by requiring:

$$caused(up(1), \mathtt{t}, S) \wedge caused(up(2), \mathtt{t}, S) \tag{8.10}$$

as one could be tempted to use in a first attempt, since this would have a completely different meaning (as observed by Turner in [110], formula (5.76) page 180) forcing always to move up both switches simultaneously in order to open the suitcase. Instead, the suitcase is actually caused open whenever both locks are up and, either lock 1, or lock 2 or both have been caused to be up. That is, we should replace (8.9) by:

$$holds(up(1), S) \wedge holds(up(2), S) \wedge (caused(up(1), \mathtt{t}, S) \vee caused(up(2), \mathtt{t}, S)) \supset caused(open, \mathtt{t}, S) \tag{8.11}$$

which is equivalent to the conjunction of (6.20) and (6.21) we obtained as translation of the $\mathcal{P}$-rule:

$$open \ \mathbf{if} \ up(1) \wedge up(2)$$

Thus, the definition of pertinence of a conjunction makes the representation easier and more comfortable, while preserving postulates P1 and P5.

To deepen in the interest of postulate P5, let us analyze now the account balance example using Lin's causal expressions. If we used the rule:

$$holds(balance(X), S) \wedge holds(transac(Y), do(A, S)) \supset caused(balance(X+Y), \mathtt{t}, do(A, S))$$

to compute the addition of values[6], the application of a waiting action would cause the last (persisting) value of *transac* to be improperly added to *balance*. The solution to this problem is simply to formulate the rule as:

$$holds(balance(X), S) \wedge caused(transac(Y), \mathtt{t}, do(A, S)) \supset caused(balance(X+Y), \mathtt{t}, do(A, S))$$

so that the value of *transac* is taken into account only if it has been caused right now. The problem is that, as we are not said how *transac* is computed, this solution would not work *unless we can guarantee* that $caused(transac(Y), \mathtt{t}, do(A, S))$, *in its turn*, is only true when there exist

[5]It could be objected that this behavior may point out the execution of a so-called "natural" action or delayed effect. However, it is clear that this scenario is not actually related to this feature.

[6]For simplicity sake, we have omitted the necessary additional axioms to avoid simultaneous different values for fluents *transac* and *balance*.

some causal propagation from the performed actions. In other words, the pertinence postulates must be satisfied *by all the involved causal rules.*

It must be observed that this problem (under the point of view of pertinence postulates) of Lin's formulation is mainly due to its freedom for expressing causal rules, rather than on a limitation on their shape. This is because the general shape of a causal rule was actually defined in [65] as a formula like:

$$\Phi(holds, S) \wedge caused(F_1, V_1, S) \wedge \cdots \wedge caused(F_n, V_n, S) \supset caused(F, V, S) \tag{8.12}$$

where *caused* atoms can be also used in the condition. As an example for using the general shape, Lin proposes adding to the suitcase scenario a new fluent, *down*, which is the antonym of *up*, so that one of them is caused to be true iff the other is caused to be false, and vice versa:

$$\begin{aligned} caused(up(L), \mathtt{t}, S) &\supset caused(down(L), \mathtt{f}, S) & (8.13)\\ caused(up(L), \mathtt{f}, S) &\supset caused(down(L), \mathtt{t}, S) & (8.14)\\ caused(down(L), \mathtt{t}, S) &\supset caused(up(L), \mathtt{f}, S) & (8.15)\\ caused(down(L), \mathtt{f}, S) &\supset caused(up(L), \mathtt{t}, S) & (8.16) \end{aligned}$$

for each lock $L \in \{1, 2\}$.

This general shape of rule has the disadvantage that, when cyclic references for *caused* atoms occur, it prevents the use of Clark's completion [7]. Besides, it is difficult to find a natural reason why this example must be formulated as (8.13)-(8.16) instead of:

$$\begin{aligned} holds(up(L), S) &\supset caused(down(L), \mathtt{f}, S) & (8.17)\\ \neg holds(up(L), S) &\supset caused(down(L), \mathtt{t}, S) & (8.18)\\ holds(down(L), S) &\supset caused(up(L), \mathtt{f}, S) & (8.19)\\ \neg holds(down(L) S) &\supset caused(up(L), \mathtt{t}, S) & (8.20) \end{aligned}$$

which follows the same "pattern" as the ramification rule (8.9) we had before. The only explanation for the change in representation seems to be that this encoding would unadequatedly affect the minimization for $up(L)$. To see this, consider the initial state:

$$\neg holds(up(1), s_0) \wedge \neg holds(up(2), s_0) \wedge \neg holds(open, s_0)$$

trying to predict what happens in situation $s_1 = do(toggle(1), s_0)$ (we toggle the first lock). It is clear that $caused(up(1), \mathtt{t}, s_1)$ becomes true as a direct effect of $toggle(1)$, and so we get $holds(up(1), s_1)$ and $caused(down(1), \mathtt{f}, s_1)$ by respectively applying (8.4) and (8.17). However, we actually get several selected models depending on the truth values for the second lock. One of these models is obtained by assuming $\neg holds(up(2), s_1)$ and $holds(down(2), s_1)$ when fixing the extension of *holds*. These assumptions, together with (8.18) and (8.19), respectively imply $caused(down(2), \mathtt{t}, s_1)$ and $caused(up(2), \mathtt{f}, s_1)$, and so, both fluents are not affected by the

[7] In Lin's work it was asserted that, at least, the theory is still definite for *caused*, and so, its circumscription leads to a unique minimal model. This is not accurately true, since we actually get a unique minimal model per valid combination of *holds* predicate (remember we are fixing the extent of *holds*). As a result, Lin's approach leads also to nondeterminism and, in fact, is not well-supported. For example, when representing the gear wheels example (without using *caused* in the rule conditions) we get *two* minimal models, independently of the initial state: one in which both wheels are caused to turn one each other, and one in which they are caused not to turn one each other.

afterwards application of the frame axiom. As a result, we get a model in which the second switch is moved up without any evident reason.

As explained in [65], this problem of affecting the minimization for the original fluents $up(1), up(2)$ does not occur when formulas (8.13)-(8.16) are used instead. So, the reason in this case for using *caused* in the condition seems to be motivated by a purely technical purpose (to avoid undesired effects with circumscription) rather than by any particular representational aspect. As a result, there is no homogeneous method establishing when the condition of a causal rule must depend on *caused*. The process of adding new rules becomes in this way a nonmodular task: we must always decide whether the new rules may affect or not to the minimization for the previously defined fluents.

8.3 Occlusion

Among the so-called *standard* approaches for Reasoning about Actions, one of the closest to pertinence is perhaps Sandewall's *Occlusion* [97], specially in its use for solving the ramification problem inside Temporal Action Logic (TAL) [47, 33]. The essential idea of occlusion is to allow handling elementary expressions like:

$$\mathbf{X}\ f$$

read "occluded f", where f is some fluent. In Sandewall's words ([97], pages 234, 235) :

> "It is intended that $\mathbf{X}\ f$ shall be true at time t iff there is no preference for the fluent f to retain its value in the transition from θt to t. ...The occlusion predicate can be seen as a special-purpose abnormality predicate, where the reason for the abnormality has been made precise. Notice that occlusion does not *imply* change; it merely allows it."

This last sentence is nothing else but postulate P4 expressed for occlusion. In fact, occlusion satisfies postulates P1 through P4, being by this reason the closest approach to pertinence. The main difference, however, is still centered on postulate P5, which is not fully satisfied and may imply obtaining less occluded fluents than those actually needed from the point of view of pertinence. To illustrate this, let us center the analysis in TAL formulas for dealing with indirect effects. A possible representation of the suitcase scenario using TAL surface language, called $\mathcal{L}(\mathcal{SD})$, could be:

$$[I, I+1]\ toggle(L) \rightsquigarrow ([I]up(L) \supset [I+1]\neg up(L)) \quad (8.21)$$

$$[I, I+1]\ toggle(L) \rightsquigarrow ([I]\neg up(L) \supset [I+1]up(L)) \quad (8.22)$$

$$\forall I.\ [I]\big((up(1) \wedge up(2)) \gg [I]open\big) \quad (8.23)$$

These formulas are unfolded into a basic formalism which uses just two predicates [8]: $holds(F, I)$ and $occlude(F, I)$. Thus, the expression $\mathbf{X}\ F$ will be true at time I iff $occlude(F, I)$ is satisfied. Note that the predicate representation is also closer to pertinence calculus than that of Lin's *caused* predicate, since occlusion does not refer to the fluent truth value. In this way, we can establish the straightforward correspondence:

$$occlude(F, I) \quad \equiv \quad pert(F, \mathtt{p}, I)$$

[8] For comparison sake, we have reversed the original ordering of the predicate arguments.

or, when pertinence value is not reified, simply as:

$$occlude(F, I) \quad \equiv \quad pert(F, I)$$

The expressions for direct effects, like (8.21) and (8.22) are actually understood in $\mathcal{L}(\mathcal{SD})$ as macros, so that they are previously instantiated with each action occurrence explicitly observed. Without entering into detail about the translation, the most interesting part is the final representation of (8.23), which leads to the pair of formulas:

$$holds(up(1) \wedge up(2), I) \quad \supset \quad holds(open, I) \tag{8.24}$$

$$holds(up(1) \wedge up(2), I+1) \wedge \neg holds(up(1) \wedge up(2), I) \quad \supset \quad occlude(open, I+1) \tag{8.25}$$

Unfolding the formula reification inside *holds*, we get:

$$holds(up(1), I) \wedge holds(up(2), I) \supset holds(open, I) \tag{8.26}$$

$$holds(up(1), I+1) \wedge holds(up(2), I+1) \wedge \neg(holds(up(1), I) \wedge holds(up(2), I)) \supset occlude(open, I+1) \tag{8.27}$$

Together with this rule representation, the semantics consists in a filter preferential entailment, circumscribing the *occlude* atoms (while fixing *holds*) and conjoining afterwards the so-called *Nochange Axiom*:

$$holds(F, I) \oplus holds(F, I+1) \supset occlude(F, I+1) \tag{8.28}$$

(with $\oplus$ standing for exclusive disjunction), which is nothing else but the contraposition of the usual Universal Frame Axiom:

$$\neg occlude(F, I+1) \supset [holds(F, I) \equiv holds(F, I+1)] \tag{8.29}$$

Thus, like the pertinence calculus circumscriptive encoding, the minimization method coincides again with Lin's technique, in which it seems to be inspired, although formulas are actually more similar to pertinence calculus, due to the already explained correspondence between predicates *pert* and *occlude*.

Back to the representation of ramification rules, it is easy to see that (8.24) and (8.25) can be reexpressed altogether as the single formula:

$$holds(up(1), I+1) \wedge holds(up(2), I+1) \wedge \neg(holds(up(1), I) \wedge holds(up(2), I)) \supset holds(open, I+1) \wedge occlude(open, I+1) \tag{8.30}$$

which is very similar to the final representation of this same rule in pertinence calculus (6.19). Apart from the use of *occlude* versus *pert*, we directly observe that here, *occlude* is only present in the consequent, and not in the antecedent (violation of postulate P5). However, the condition is slightly more elaborated than Lin's rules, since it is not enough with requiring both switches to be up, but also that at least one of them was down before. The general pattern is, in fact, that the whole formula used as rule condition must have experimented a change of truth value, passing from false in I to true in $I+1$. In this way, we somehow replace the idea of requiring pertinence of $up(1) \wedge up(2)$ by the idea of requiring its *change of truth value*.

Some advantages, with respect to Lin's notation, are obtained from this formulation. The first and perhaps most interesting one is that, contrarily to Lin's approach, postulate P1 is always satisfied. For example, consider the *wait* action with an open suitcase, while both locks are up. It is easy to see that, since the formula $up(1) \wedge up(2)$ *remains* true from the previous situation to the next one, there is no need to consider *open* to be occluded, and so, it persists unaffected. Generally speaking, whenever we obtain a fact $occlude(F, I)$ it will be always due to some causal chain directly or indirectly initiated by an action execution, not by the mere application of inertia. A second advantage, which has more to do with the technical implementation rather with a real representation subject, is that, as *occlude* only occurs in the consequents, the circumscription can always be computed by using Clark's completion.

Although, as we have seen, occlusion avoids forcing unnecessary *occluded* atoms when just inertia has been applied, we will see next that it is too weak for other examples, falling short in the set of fluents that should be actually caused. This is because postulate P5 is just applied in its "if" direction, but not in the "only if." In this way, requiring the change of truth value will always imply that some fluent has become occluded/pertinent (postulate P3 is guaranteed by axiom (8.28)) but there may be cases in which the condition is pertinent whereas its value has not changed (postulate P4). In such cases, the causal rule is not applied, when it should actually be taken into account.

As an example of this behavior, consider again the account balance scenario and assume that we are given two consecutive transactions by the same amount. It is clear that *both numbers* must be taken into account to compute the average, and that this situation is different from the one in which the last transaction has just persisted. The occlusion rules for computing the balance would finally look like:

$$holds(transac(Y), I+1) \wedge \neg holds(transac(Y), I) \wedge holds(balance(X), I) \supset \\ holds(balance(X+Y, I+1) \wedge occlude(balance(X+Y), I) \quad (8.31)$$

which it is easy to see that would not be applicable when the value of *transac* is caused to be the same one as before. So, the *balance* would simply persist, while it should have been updated.

Finally, another interesting similarity between occlusion and pertinence is that the former is also defined for complex formulas, although for a different purpose. The occlusion of a propositional fluent formula corresponds to the occlusion of *all* the fluents occurring in it. Notice how this is exactly the dual concept of pertinence: *at least one* of the involved fluents is required to be pertinent. The reason for this difference seems to rely on the way in which both definitions are used. On the one hand, occlusion of a complex formula is thought for forcing the *consequent* of a causal rule to become an exception to inertia. In this way, all the occurring fluents are occluded, when perhaps it could be enough with occluding just some of them in order to allow the change of truth value of the whole formula. On the other hand, pertinence of a complex formula is mainly thought for an easier representation of the *antecedent* of a causal rule, precisely where occlusion is never used in TAL. As we saw in the previous section (formula (8.10)), requiring the pertinence of all the fluents occurring in the rule condition would not capture the appropriated meaning for pertinence postulates.

8.4 Event Calculus

The Event Calculus [53] is another example of well known *standard* action approach which, although not explicitly labeled as causal, relies on the use of change-oriented predicates. As we did with occlusion, we will particularly focus the study on the solution to the ramification problem which, in fact, has been first studied in a recent work by Shanahan [103]. That solution is very interesting because it consists in defining additional special-purpose noninertial fluents, that is, adding new *events*, to point out when an effect must be considered to be "new". In other words, thinking about the account balance example, Shanahan's proposal informally corresponds to the already explained solution of defining a special fluent $new\_transac$.

As explained in that paper, the Event Calculus notation used in [103] was directly obtained from chapter 16 in [101]. Event Calculus has the particularity of handling *events* instead of actions. Informally speaking, we can see an event as a noninertial fluent whose value is understood as momentary. When the value of an event is directly specified as a fact it behaves as an usual action. However, we can also reason about events, adding conditions for deriving their values. In this way, they may behave as "derived actions." The basic Event Calculus formalism is more complicated than Lin's or occlusion solutions, using a greater amount of formulas and predicates. The reason for this is that formulas seem to be oriented to an interval based representation, rather than exclusively relying on situation indices. We will first present exactly the original notation and proceed later to an equivalent simplification for comparison purposes.

8.4.1 Original formulation

The basic predicates in an Event Calculus representation are:

- $initiates(A, F, I)$ (respectively $terminates(A, F, I)$) to express that fluent F starts (respectively ceases) to hold after action A at time I,
- $releases(A, F, I)$ means fluent F is not subject to inertia after A at I,
- $initially_P(F)$ (resp. $initially_N(I)$) means that fluent F holds (resp. does not hold) from time 0,
- $clipped(I_1, F, I_2)$ (resp. $declipped(I_1, F, I_2)$) means that fluent F is not subject (resp. is subject) to inertia along interval $[I_1, I_2]$,
- $happens(A, I)$ to express that action A occurs at time I,
- and finally, $holds(F, I)$ with the standard meaning.

These predicates are used to represent effect axioms Σ (using *initiates*, *terminates* and *releases*) plus another set of formulas Δ containing the sequence of actions (using *happens*) and the initial state (using $intially_P$ and $initially_N$). All these formulas are combined with a set of general axioms in order to derive the resulting set of $holds(F, I)$ atoms:

$$initially_P(F) \wedge \neg clipped(0, F, I) \supset holds(F, I) \quad (8.32)$$

$$initially_N(F) \wedge \neg declipped(0, F, I) \supset \neg holds(F, I) \quad (8.33)$$

$$happens(A, I_1) \wedge initiates(A, F, I_1) \wedge I_1 < I_2 \wedge \neg clipped(I_1, F, I_2) \supset holds(F, I_2) \quad (8.34)$$

$$happens(A, I_1) \wedge terminates(A, F, I_1) \wedge I_1 < I_2 \wedge \neg declipped(I_1, F, I_2) \supset \neg holds(F, I_2) \quad (8.35)$$

$$\begin{aligned} &clipped(I_1, F, I_3) \equiv \\ &\quad \exists A, I_2.\ \big(happens(A, I_2) \wedge I_1 < I_2 \wedge I_2 < I_3 \wedge (terminates(A, F, I_2) \vee releases(A, F, I_2))\big) \end{aligned} \quad (8.36)$$

$$declipped(I_1, F, I_3) \equiv$$
$$\exists A, I_2.\ \big(happens(A, I_2) \wedge I_1 < I_2 \wedge I_2 < I_3 \wedge (initiates(A, F, I_2) \vee releases(A, F, I_2))\big) \quad (8.37)$$

Let us call EC to this set of axioms (8.32)-(8.37) and UNA to the unique names axioms for fluents and actions. The final theory is circumscribed as follows:

$$\text{CIRC}[\Sigma; initiates, terminates, releases] \wedge \text{CIRC}[\Delta; happens] \wedge \text{ EC } \wedge \text{ UNA}$$

The solution to ramification problem presented in [103] requires additional formulation. Besides effect axioms (used to represent direct effects of actions), new causal constraints of the shape:

$$\textbf{initiating } \phi \textbf{ causes } \psi$$

are introduced, where ϕ is any propositional combination of fluent names and ψ is a fluent name or its negation. These causal constraints are later understood as abbreviations of formulas using the following four additional predicates:

- $started(F, I)$ (resp. $stopped(F, I)$) meaning that either F holds (resp. does not hold) at I or an event occurs at I that initiates (resp. terminates) F,
- $initiated(F, I)$ (resp. $terminated(F, I)$) that means that F has been started (resp. stopped) at I but no event occurs at I that terminates (resp. initiates) F.

The meanings of these predicates become perhaps clearer by looking at the new general axioms:

$$\begin{aligned} initiated(F, I) &\equiv started(F, I) \wedge \neg\exists A.\ \big(happens(A, I) \wedge terminates(A, F, I)\big) \quad (8.38) \\ terminated(F, I) &\equiv stopped(F, I) \wedge \neg\exists A.\ \big(happens(A, I) \wedge initiates(A, F, I)\big) \quad (8.39) \end{aligned}$$

Then, each causal constraint like:

$$\textbf{initiating } \phi \textbf{ causes } F$$

is simply translated as a pair of formulas:

$$\begin{aligned} \phi' \wedge started(F, I) \supset happens(A', I) \\ terminates(A', F, I) \end{aligned}$$

where A' is a new action symbol and ϕ' is the result of replacing in ϕ (expressed in a normal form) positive fluent literals F by $initiated(F, I)$ and negative fluent literals $\neg F$ by $terminated(F, I)$. Analogously, a constraint like:

$$\textbf{initiating } \phi \textbf{ causes } \neg F$$

is translated as:

$$\begin{aligned} \phi' \wedge stopped(F, I) \supset happens(A', I) \\ initiates(A', F, I) \end{aligned}$$

As an example, Lin's suitcase would be represented as follows. The direct effects are simply expressed as:

$$\begin{aligned} \neg holds(up(L), I) &\supset initiates(toggle(L), I) \\ holds(up(L), I) &\supset terminates(toggle(1), I) \end{aligned}$$

for any lock number L, whereas the constraint:

$$\textbf{initiating } up(1) \wedge up(2) \ \textbf{ causes } open$$

becomes the pair of formulas:

$$initiated(up(1), I) \wedge initiated(up(2), I) \wedge started(open, I) \supset happens(got\_open, I)$$
$$initiates(got\_open, open, I)$$

8.4.2 Adapting the notation

As we can see, this formulation of Event Calculus relies on a great amount of intermediate predicates that, in many cases, are simple definitions of elaborated formulas, and so, can be replaced by them. Besides, the notation is sometimes redundant because of using different predicate names for each truth value. Finally, the frame axioms have a different shape since they do not seem to be oriented to a transition-based system, but they rely on interval definitions instead. To simplify the comparison, we present some slight modifications that will help in emphasizing the actual main differences with respect to pertinence calculus and, in moreover, to the rest of approaches.

As a first change, we will only rely on three different predicates:

1. $happens(A, I)$, which maintains its previous shape.
2. $holds(F, V, I)$, so that we reify the truth value for the fluent (we assume the inclusion of axioms (6.1) and (6.2)).
3. $set(A, F, V, I)$, that replaces both *initiates* and *terminates* also by reifying the fluent truth value

Besides, we replace:

$$initially_P(F) \stackrel{\text{def}}{=} holds(F, \mathtt{t}, 0) \tag{8.40}$$

$$initially_N(F) \stackrel{\text{def}}{=} holds(F, \mathtt{f}, 0) \tag{8.41}$$

We will also omit predicate *releases* since we are not interested in fluents that may become non-inertial along the narrative. If we apply these notational changes, it is easy to see that the axioms (8.32)-(8.37) can be expressed more compactly as:

$$holds(F, V, 0) \wedge$$
$$\neg \exists A, I_2, V'. \ \big(0 < I_2 \wedge I_2 < I_3 \wedge V \neq V' \wedge happens(A, I_2) \wedge set(A, F, V', I_2)\big) \supset$$
$$holds(F, V, I_3) \tag{8.42}$$

$$\begin{aligned}&happens(B, I_1) \wedge set(B, F, V, I_1) \wedge \\ &\qquad \neg\exists A, I_2, V'.\ \big(I_1 < I_2 \wedge I_2 < I_3 \wedge V \neq V' \wedge happens(A, I_2) \wedge set(A, F, V', I_2)\big) \supset \\ &\hfill holds(F, V, I_3)\end{aligned} \tag{8.43}$$

where we have simply replaced *clipped* and *declipped* by their definitions. After this change, it is easy to see that the only predicate to be minimized is *set*, while maintaining fixed *happens* and *holds*, and requiring (8.42) and (8.43) *outside* the minimization:

$$\mathrm{CIRC}[\Sigma; set] \wedge \mathrm{CIRC}[\Delta; happens] \wedge (8.42) \wedge (8.43) \wedge\ \mathrm{UNA}$$

The difference in the formulation of frame axioms (8.42) and (8.43) with respect to the ones we have seen until now – (UFR), (8.1), (8.6) or (8.29) – is something more than a simple change in notation: the Event Calculus formulas have the advantage of being *applicable to non-discrete time*. In fact, we could also reformulate (UFR) in pertinence calculus to adopt a similar fashion, so that we could use real numbers for situations in the narrative:

$$holds(F, V, I_1) \wedge \forall I_2.\ \big(I_1 < I_2 \wedge I_2 \leq I_3 \wedge \neg pert(F, I_2)\big) \supset holds(F, V, I_3)$$

For comparison sake, however, we will do the opposite, that is, reformulate the Event Calculus frame axioms (for the case of integer time) to obtain an equivalent shape closer to the narrative approaches we have seen so far:

Theorem 11 *If we handle discrete time, the conjunction* (8.42) $\wedge$ (8.43) *is equivalent to the conjunction of:*

$$\begin{aligned}&\neg\exists A, V'.\ \big(V \neq V' \wedge happens(A, I) \wedge set(A, F, V', I)\big) \supset \\ &\hfill \big(holds(F, V, I) \equiv holds(F, V, I+1)\big)\end{aligned} \tag{8.44}$$

$$\exists A.\ \big(happens(A, I) \wedge set(A, F, V, I)\big) \supset holds(F, V, I+1) \tag{8.45}$$

Proof
(See appendix A). □

Note how (8.44) is more similar now to the universal frame axioms seen so far, whereas (8.45) can be compared to Lin's axioms (8.4) and (8.5). Besides, we can also rephrase ramification rules to express them in terms of *holds*, *happens* and *set*. For instance, (8.40)-(8.40) would correspond to:

$$\begin{aligned}&\Big(holds(up(1), \mathtt{t}, I) \vee \exists A.\ \big(happens(A, I) \wedge set(A, up(1), \mathtt{t}, I)\big)\Big) \quad \wedge \\ &\qquad\qquad \neg\exists A.\ \big(happens(A, I) \wedge set(A, up(1), \mathtt{f}, I)\big) \quad \wedge \\ &\Big(holds(up(2), \mathtt{t}, I) \vee \exists A.\ \big(happens(A, I) \wedge set(A, up(2), \mathtt{t}, I)\big)\Big) \quad \wedge \\ &\qquad\qquad \neg\exists A\big(happens(A, I) \wedge set(A, up(2), \mathtt{f}, I)\big) \quad \wedge \\ &\Big(holds(open, \mathtt{f}, I) \vee \exists A.\ \big(happens(A, I) \wedge set(A, open, \mathtt{f}, I)\big)\Big) \quad \supset \quad happens(got\_open, I)\end{aligned} \tag{8.46}$$

$$set(got\_open, open, \mathtt{t}, I) \tag{8.47}$$

In words, we require for each lock that it is either up or about to be set up, but not about to be set down. Similarly, we require that *open* was false or about to be set false. When all these conditions are true, we finally force the special event *got_open* to occur, which, due to (8.47), finally sets the value of *open* to true.

8.4.3 Relation to pertinence

An important observation is that predicate *set*, which plays the role of *pert*, *occluded*, or *caused*, actually contains more information than the all these predicates. While we had $pert(F, I)$ and $occlude(F, I)$, Lin's $caused(F, V, I)$ included the reified fluent value and finally $set(A, F, V, I)$ also includes the performed action. Furthermore, we additionally *need to refer* to the action occurrence $happens(A, I)$. In this way, predicate *set* just points out the possibility of setting the value, but this possibility is not finally established until the action actually occurs. This notation seems, at a first sight, not very comfortable when we want to deal with indirect effects, since the direct reference to the action is precisely what we want to avoid in order to solve the ramification problem. Note how, in fact, in the Event Calculus formulas (both the original and the adapted ones) action names are in most cases quantified: we do not have a real interest in the particular action that sets the fluent value!

In sight of this feature, it would be easy to include one more notational change, that would make closer the Event Calculus formulation to Lin's approach, for instance. Consider the definition of a predicate $caused(F, V, I)$ as follows:

$$caused(F, V, I) \stackrel{\text{def}}{=} \exists A.\ \big(happens(A, I) \wedge set(A, F, V, I)\big)$$

This predicate is identical to Lin's *caused* (excepting that its effects for *holds* will be placed one situation after: $I + 1$). Then, axioms (8.44) and (8.45) respectively become:

$$\begin{aligned} \neg\exists V'.\ \big(V \neq V' \wedge caused(F, V', I)\big) &\supset \big(holds(F, V, I) \equiv holds(F, V, I+1)\big) \\ caused(A, V, I) &\supset holds(F, V, I+1) \end{aligned}$$

which correspond exactly to Lin's axiomatization, excepting for the delay of one situation.

As for the ramification rule, notice that the *actual purpose* of defining the special action *got_open* is exclusively due to obtain at least some action (regardless its name) that both occurs and sets open to be true. That is, the consequent of the ramification rule could simply be:

$$\exists A.\ \big(happens(A, I) \wedge set(A, open, \mathtt{t}, I)\big)$$

but this is corresponds to $caused(open, \mathtt{t}, I)$. So the final shape of the rule could simply be:

$$\begin{aligned} (holds(up(1), \mathtt{t}, I) \vee caused(up(1), \mathtt{t}, I)) &\quad \wedge \\ \neg caused(up(1), \mathtt{f}, I) &\quad \wedge \\ (holds(up(2), \mathtt{t}, I) \vee caused(up(2), \mathtt{t}, I)) &\quad \wedge \\ \neg caused(up(2), \mathtt{f}, I) &\quad \wedge \\ (holds(open, \mathtt{f}, I) \vee caused(open, \mathtt{f}, I)]) &\quad \supset \quad caused(open, \mathtt{t}, I) \end{aligned} \tag{8.48}$$

which is now much more familiar with respect to the notation we have followed. Looking at this formula, it could be thought that perhaps postulate P5 is satisfied, since the *caused* predicate is present in the rule antecedent. Unfortunately, the condition can be made true without requiring any *caused* atom to be true, just relying on values for *holds*. Besides, a change of value for the fluent in the consequent is required (*open* is required to be false before applying the rule)

Let us consider some examples of transitions. For instance, waiting on an just opened suitcase leads to the correct result: all the fluents persist and no *caused* atom is made true. This is because, after applying the waiting action, *open* was not false, as required in the antecedent (8.48). In a similar way, if we had that *open* was true, $up(1)$ false and $up(2)$ true, and we perform $toggle(1)$, *open persists true*. This is a difference w.r.t. occlusion or pertinence that, in both cases, lead to *open caused true* for this transition.

As for the account balance example, we would formulate the ramification rule as:

$$\begin{array}{rcl} holds(balance(X), \mathtt{t}, I) & \wedge & \\ (holds(transac(Y), \mathtt{t}, I) \vee caused(transac(Y), \mathtt{t}, I)) & \wedge & \\ \neg caused(transac(Y), \mathtt{f}, I) & \wedge & \\ (holds(balance(X+Y), \mathtt{f}, I) \vee caused(balance(X+Y), \mathtt{f}, I)]) & \supset & caused(balance(X+Y), \mathtt{t}, I) \end{array}$$

that is, under the precondition that fluent *balance* had value X, if *transac* has value Y or is about to get value Y (but not about to get another value), and we have that *balance* has not or is not about to get a different value from $X+Y$ then *balance* gets the value $X+Y$. It can be seen that when *transac* is not caused, that is, it has persisted (performing a waiting action, for instance), the rule is applicable. In this way, we would incorrectly "take into account" as many values in the balance as elapsed situations.

8.5 Inductive causation

Like all the previous approaches, Denecker et al's work [28], also uses a change predicate (called *caus* in this case) to represent causal rules, but has the interest of analyzing different options for the nonmonotonic technique to be applied for this fixed representation. This is in fact the methodology we have followed in this thesis: separating the representation of change itself (in our case, using the idea of pertinence) from the nonmonotonic inference technique to be applied to that representation. In other words, [28] studies a mixed approach which combines, at different levels, both orientations to causality: the change-based representation and the inferential approach.

The inference techniques mainly studied in [28] are completion and inductive definitions (which actually correspond to well-founded semantics), although stable models are also briefly commented. Besides, the comparison is focused on the behavior for causal cycles. In fact, great part of the discussion about cycles done in this thesis in inspired by Denecker et al's work, and similar conclusions are reached. For instance, as we also observed with pertinence calculus, completion has the problem of not being well supported. Similarly, their work talks about a *negative self-supportedness* to describe the effect of applying stable models semantics. As happened with pertinence calculus, the stable models interpretation for causal rules still has the problem of leading to nondeterminism, which is also interpreted as an unintended result, rather than a way of representing causal uncertainty:

> "nondeterminism should not arise due to tricky interactions of deterministic rules, but should be modeled explicitly when intended." ([28], section 5.2, page 21)

Despite of this closeness, both in the methodology and in the results of analysis for cycles, some important differences remain. On the one hand, the inductive definition for causation corresponds to well founded semantics, but nothing is said in [28] about the coherence problem. Thus, some examples (like example 13) would lead to more undefined atoms than actually needed. On the other hand, which is more important, the representation of change and causation does not completely fulfill the pertinence postulates, as happened with the other change-oriented approaches, although an interesting construction is defined which resembles a lot to the definition of pertinence for a complex formula. We will particularly focus in this section on these differences in the representation of change with respect to pertinence postulates.

Denecker et al's approach is transition oriented, exclusively dealing with two consecutive situations, let us call them $I-1$ and I. Given any fluent literal L (that is, a fluent name or its negation), the notation in [28] relies on three basic constructions:

1. $caus(L)$ to express that L becomes caused in the successor situation I
2. $holds(L)$ to represent that L is true in the initial situation $I-1$
3. $init(L)$ which is used as a shorthand notation for $holds(\neg L) \wedge caused(L)$

For comparison sake, we could bear in mind the following correspondences:

$$\begin{array}{rcl} caus(F) & \stackrel{\text{def}}{=} & caused(F, \mathtt{t}, I) \\ caus(\neg F) & \stackrel{\text{def}}{=} & caused(F, \mathtt{f}, I) \\ holds(F) & \stackrel{\text{def}}{=} & holds(F, \mathtt{t}, I-1) \\ holds(\neg F) & \stackrel{\text{def}}{=} & holds(F, \mathtt{f}, I-1) \end{array}$$

A causal (ramification) rule is represented as an expression like:

$$\textbf{initiating } \phi \textbf{ causes } L \textbf{ if } \psi$$

where *phi* and *psi* are propositional combinations of fluent symbols and L is a fluent literal. When ψ (the precondition) is trivially true, then ψ condition is omitted. For instance, the ramification rule for Lin's suitcase has the shape:

$$\textbf{initiating } up(1) \wedge up(2) \textbf{ causes } open \tag{8.49}$$

This rule is also represented as:

$$caus(open) \leftarrow init(up(1) \wedge up(2))$$

where, as we can see, *init* is applied to a complex formulas. The general effect of this will be commented later. By now, we just focus on the final unfolding of (8.49) that consists of the implications:

$$\begin{array}{rcl} caus(open) & \leftarrow & init(up(1)) \wedge holds(up(2)) \wedge \neg init(\neg up(2)) \\ caus(open) & \leftarrow & init(up(2)) \wedge holds(up(1)) \wedge \neg init(\neg up(1)) \\ caus(open) & \leftarrow & init(up(1)) \wedge init(up(2)) \end{array}$$

Since $init(up(L)) \stackrel{\text{def}}{=} caus(up(L)) \wedge holds(\neg up(L))$, we may first notice that postulate P5 is satisfied in its "if" direction: the antecedents require that some lock must have been caused, in order to reach the desired position. As a result, postulate P1 also holds which, in fact, is explicitly rephrased in Denecker et al's paper:

> "...an effect is never triggered by the absence of other effects alone, e.g. the suitcase will not be open just because nothing happens (which would be a serious violation of inertia" ([28], section 5.3, page 24).

It is easy to see that, once we have just opened the suitcase, performing a *wait* action does not make true any $init(up(L))$ for any lock, and so, no rule would be applied to obtain $caus(open)$. The suitcase would remain open, but as a result of inertia. In this sense, the meaning of $caus(open)$ would fully coincide with the intended meaning for pertinence.

However, as happened with occlusion in TAL, the conditions are excessively strong, since they require not only that some fluent in the condition must have been caused, but also that it has experimented *an actual change of value* (as explicitly stated in the definition of *init*). In this way, this approach does not satisfy the "only if" direction of postulate P5.

To understand the problem consider, as always, the account balance example:

$$\textbf{initiating } transac(Y) \textbf{ causes } balance(X+Y) \textbf{ if } holds(balance(Y))$$

assuming we have included constraints to avoid multiple simultaneous values for *transac* and *balance*. This rule is translated as:

$$caus(balance(X+Y)) \leftarrow holds(balance(X)) \wedge caus(transac(Y)) \wedge holds(\neg transac(Y))$$

which is only differs with respect to the pertinence calculus version in that conditions have been strengthened with $holds(\neg transac(Y))$. In this way, we not only require causation of *transac*, but we simultaneously *force a real change of value* in $transac(Y)$ with respect to its previous one, disabling the possibility of two repeated values that are consecutively caused. In other words, as a piece of example, the balance of the sequence of transactions $50, 50, 50, 30$ would be 80.

8.5.1 Initiation vs. pertinence of a formula

Although, as we have seen, causal rules in [28] do not follow exactly the pertinence postulates due to an excessive strengthening of rule conditions, there exists however a very interesting feature of [28] which is, to the best of our knowledge, the closest concept to our definition of pertinence of a formula that can be found in the literature. We are, of course, talking about the *initiation of complex formulas* [9]. Essentially, this feature explains how predicate *init* can be applied to a reified formula by a series of transformation steps very similar to the ones followed in L^2: (3.13)-(3.23).

The original definitions rely on the concept of *supporting set* for a complex formula ϕ which is equivalent to the idea of (consistent) three-valued model for ϕ. For instance, considering three fluents $\{up(1), up(2), open\}$, the formula $up(1) \wedge up(2)$ has three supporting sets:

$$\begin{aligned} S_0 &= \{up(1), up(2)\} \\ S_1 &= \{up(1), up(2), open\} \\ S_2 &= \{up(1), up(2), \neg open\} \end{aligned}$$

Clearly, a formula will be true iff all the literals in (at least) one of its supporting sets are true. Using this idea, the *initiation* of a formula ϕ is described as:

[9] Covered in section 5.3 in [28].

1. ϕ is not already true
2. For some supporting set S of ϕ, there exist S_i and S_p, subsets of S, $S_i \subseteq S$, $S_p = S - S_i$ such that all the literals in S_i are initiated and all the ones in S_p are true and not terminated.

Let us see a formal definition. Given a literal L, we write $\overline{L}$ to denote the opposite literal: $\overline{p} = \neg p$; $\overline{\neg p} = p$. For any set of literals S, we define the abbreviations:

$$\begin{aligned} \overline{S} &\stackrel{\text{def}}{=} \{\overline{L} : L \in S\} \\ caus(S) &\stackrel{\text{def}}{=} \{caus(L) : L \in S\} \\ holds(S) &\stackrel{\text{def}}{=} \{holds(L) : L \in S\} \\ \overline{caus(S)} &\stackrel{\text{def}}{=} \{\neg caus(L) : L \in S\} \end{aligned}$$

Then, given a general rule:

initiating ϕ **causes** L **if** ψ

its translation can be defined as the set of formulas:

$$caus(L) \leftarrow caus(S_i) \wedge holds(S_p) \wedge \overline{caus(\overline{S_p})} \wedge holds(\neg\phi) \wedge holds(\psi)$$

for each supporting set $S_i \cup S_p$ of ϕ, being $S_i \neq \emptyset$. The fact of requiring $S_i \neq \emptyset$ is very important, since otherwise, we would allow application of rules that do not depend on any positive causation. Therefore, this requirement corresponds to the implementation of postulate P5 (the condition must be pertinent), in its "if" direction. Remember that pertinence of a propositional formula corresponds to requiring that at least one of its atoms is pertinent.

To see how complex initiation works, consider again (8.49). We must deal with the three supporting sets studied before: S_0, S_1 and S_2. The rules from S_0 would be:

$$\begin{aligned} caus(open) &\leftarrow caus(up(1)) \wedge holds(up(2)) \wedge \neg caus(\neg up(2)) \wedge holds((\neg up(1) \wedge up(2))) \\ caus(open) &\leftarrow caus(up(2)) \wedge holds(up(1)) \wedge \neg caus(\neg up(1)) \wedge holds((\neg up(1) \wedge up(2))) \\ caus(open) &\leftarrow caus(up(1)) \wedge caus(up(2)) \wedge holds((\neg up(1) \wedge up(2))) \end{aligned}$$

Note that we would also generate rules for S_1 and S_2, although it seems clear that they would not be really needed. In fact, this is a problem of this definition, from the practical point of view, since the number of these ground rules combinatorily explodes when we add more fluents (we get more irrelevant supporting sets).

Looking at the resulting rules, it is easy to see that they have stronger conditions than the rules that resulted from pertinence (6.20) and (6.21). As we can see, the real difference is that initiation of formulas forces the condition to be previously false, which may lead to problems (as we saw with the balance example). Apart from this, the two decompositions actually yield the same result, but pertinence transformation is simplier. The reason for this simplicity is due to differences in the predicate notation. Thus, in [28], as predicate *holds* is exclusively used for the previous situation, *there is no way* to represent that a fluent value just holds in the resulting state (without saying anything about its causation) *unless* we refer to the previous situation $holds(F)$ and explicitly say that the fluent has persisted $\neg caus(\neg F)$. The possibility of referring to *holds* in the successor state is essential for obtaining a more reduced set of ground rules since, in this way, we do not need to explicitly unfold *all* the combinations caused/persisted for all the literals in the supporting set.

8.6 Abnormality and Logic Programming

In [12] another interesting approach was proposed. Although not explicitly classified as causal, it can be seen as another mix of the inferential and the change-based orientations. The change predicate is the already seen $ab(L, A, S)$, with the only difference that the first argument L is a fluent literal, rather than a fluent name. In this way, this notation can actually be seen as a truth reification, as in Lin's *caused* predicate. However, as an important difference to Lin's causation, in order to obtain directionality of inference rules and to minimize the extent of ab, the applied nonmonotonic technique this time is Logic Programming (under the answer sets semantics) rather than a circumscription policy. As a matter of fact, Baral's paper was focused on dealing with defeasible causal rules, for which two levels of abnormality predicates were defined, but we will omit here this distinction and pay more attention to the treatment of undefeasible ramification rules.

Under Baral's formalization, the inertia law corresponds to the program rules:

$$holds(F, do(A, S)) \leftarrow holds(F, S) \wedge not\ ab(F, A, S) \quad (8.50)$$

$$\neg holds(F, do(A, S)) \leftarrow \neg holds(F, S) \wedge not\ ab(\neg F, A, S) \quad (8.51)$$

whereas effect axioms and ramification rules would have the shape:

$$\neg holds(up(N), do(toggle(N), S)) \leftarrow holds(up(N), S) \quad (8.52)$$

$$ab(up(N), toggle(N), S) \leftarrow holds(up(N), S) \quad (8.53)$$

$$holds(up(N), do(toggle(N), S)) \leftarrow holds(up(N), S) \quad (8.54)$$

$$ab(\neg up(N), toggle(N), S) \leftarrow \neg holds(up(N), S) \quad (8.55)$$

$$holds(open, do(A, S)) \leftarrow holds(up(1), do(A, S)), holds(up(2), do(A, S)) \quad (8.56)$$

$$ab(\neg open, A, S) \leftarrow holds(up(1), do(A, S)), holds(up(2), do(A, S)) \quad (8.57)$$

First of all, it must be noticed that the idea of "F is caused true in $do(A, S)$" is represented here in an indirect way, by asserting that the opposite value is abnormal: $ab(\neg F, A, S)$. Similarly, "F caused false" would correspond to $ab(F, A, S)$. The general idea is that, when we cause a fluent to get a value, we should break the inertia for the other value (or values, thinking about a multi-valued fluent). Then, the inertia (8.50)-(8.51) requires that, for obtaining any fluent value, it both held before and it is not abnormal. Since abnormality is applied to one fluent value at each case, a first important consequence is that a fluent value may be obtained by applying simultaneously inertia and a causal rule (violation of postulate P2), if the fluent happened to have the same value before. Besides, postulate P5 is again not satisfied, since conditions of (nondefeasible) causal rules do not include references to ab.

Thinking about our examples, if we wait after opening the suitcase, we obtain:

$$ab(\neg open, wait, S)$$

which, although it does not yield any strange effect, would point out that fluent *open* was caused with some value different from false. In other words, although inertia should follow for all the fluents, we obtain some true atoms for ab. As for the balance example, it is easy to see that rule conditions should include references to ab. Otherwise, a construction like:

$$holds(balance(X + Y), do(A, S)) \leftarrow holds(transac(Y), do(A, S)), holds(balance(X), S)$$

$$ab(\neg balance(X + Y), A, S) \leftarrow holds(transac(Y), do(A, S)), holds(balance(X), S)$$

would always consider as many values as elapsed situations, even when no values for *transac* are provided, but they are obtained from inertia instead.

8.7 Thielscher's approach

Another well known causal approach is the so-called *causal relationships*, introduced in [107]. We could also classify this approach as a mixed orientation because it combines a distinction between caused and non-caused effects with a special inference method for applying causal rules. In fact, this approach can be considered the most 'procedural' one, since it does not provide a classical interpretation for causal rules nor an explicit representation of inertia but, instead, its semantics is based on an iterative process. The idea is to fire repeatedly the applicable rules at each stage, in a similar way to the operational semantics for $\mathcal{P}$-language, but with the particularity that it allows different changes of value for a same fluent in a same situation.

8.7.1 Basic definitions and a pair of examples

A *causal relationship* is a construction of the shape:

$$A \textbf{ causes } B \textbf{ if } \phi \tag{8.58}$$

where A and B are fluent literals, whereas ϕ is a fluent formula. Intuitively, literal A is the "triggering effect", literal B is the resulting effect and formula ϕ is a condition that must currently hold (in the resulting situation) but which is not needed to be caused. It must be observed that causal relationships *do not exist independently*, but are always associated to some state constraint, which has the shape of a propositional fluent formula.

For instance, the ramification for Lin's suitcase is represented as the pair of causal relationships:

$$up(1) \textbf{ causes } open \textbf{ if } up(2) \tag{8.59}$$

$$up(2) \textbf{ causes } open \textbf{ if } up(1) \tag{8.60}$$

associated to the state constraint:

$$up(1) \wedge up(2) \supset open$$

To decide the applicability of one of these rules, the inference process maintains a pair of sets of fluent literals (S, E), where S (which is maximal and consistent) represents the "current state" and E (which is possibly inconsistent) points out the current triggering effects. Any rule (8.58) is applicable w.r.t (S, E) iff:

1. $S \models \phi \wedge A \wedge \neg B$ using propositional satisfiability
2. $A \in E$

that is, the rule condition holds, the effect A holds and is among the triggering effects and the consequent is not true. As a result, we get the new pair:

$$((S - \{\overline{B}\} \cup \{B\}), E \cup \{B\})$$

For a set of causal relationships R, we denote $(S, E) \longrightarrow_R (S', E')$ to express that (S, E) is the result of applying some rule in R to (S, E). Then, given an initial state S, a delete-list C (or precondition) and an add-list E (or set of direct effects), the *successor state* S' is any state satisfying:

1. $((S - C) \cup E, E) \stackrel{*}{\longrightarrow}_R (S', E')$ for some E'

2. S' satisfies all the state constraints

Note that the process is non-deterministic: we may apply the rules in any sequence order, stopping at any state satisfying the constraints (even not necessarily the first to appear in the sequence). Observe also that, along this process, we generate a sequence of intermediate states in which fluents are continuously changing their values and this may affect the triggering of other rules. This means that to establish the successor state, we describe a chain of low level intra-state transitions. An also important feature is that the triggering effects set, E, never decreases and may be inconsistent. To illustrate these definitions, Thielscher proposes in [107] the following example:

Example 17 (**Thielscher's relay**) Consider a variation of the lamp circuit (example 1) by introducing a relay that controls switch 2, as depicted in figure 8.1. □

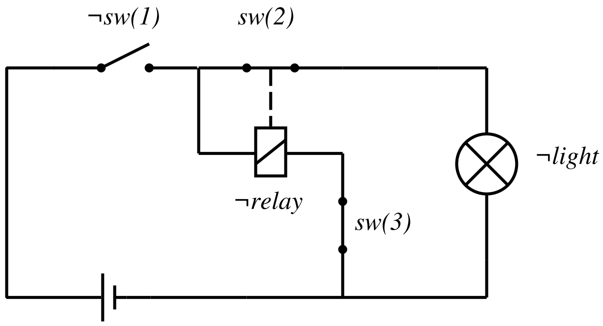


Figure 8.1: Electric circuit with a lamp and a relay.

Without detailing the set of causal rules, the intuitive behavior is the following one. It seems clear that after closing $sw(1)$, the light must be finally off, since the relay is activated, and so switch $sw(2)$ results open. So, the only possible final state is $\{sw(1), \overline{sw(2)}, sw(3), relay, \overline{light}\}$. However, this state is actually obtained following two different sequences of rule applications: (1) assuming that the relay is faster than the lamp, $sw(2)$ becomes immediately false and the light is never turned on during the intermediate states; and (2) assuming than the lamp is faster than the relay, and so the former is momentarily set to true, although the relay is activated later and then the light is turned off again. As a result, there exists a rule application path in which the light experiments a momentary flash. In [107], this example is further modified afterwards by incorporating a light detector (fluent *detect*), which is set to true, whenever the literal $light$ is triggered. With this variation, we obtain now two *different* successor states (the outcome is nondeterministic), where *detect* may become true or false depending on the relative delays between the lamp and the relay.

As we had explained, causal relationships are always associated to some state constraint. In fact, the causal relationships associated to the same constraint can be seen as a description of all its possible causal decompositions. For example, the constraint:

$$sw(1) \wedge sw(2) \equiv light \tag{8.61}$$

is decomposed into the causal directions:

$$\begin{aligned} sw(1) \textbf{ causes } light \textbf{ if } sw(2) \\ sw(2) \textbf{ causes } light \textbf{ if } sw(1) \\ \overline{sw(1)} \textbf{ causes } \overline{light} \textbf{ if } \top \\ \overline{sw(2)} \textbf{ causes } \overline{light} \textbf{ if } \top \end{aligned}$$

so that the above constraint is always understood as an influence from the switches to the light. In [107] an automatic process was provided in order to obtain the set of causal relationships from a given constraint, assuming that additional *influence information* is available. This influence information is simply provided as pairs of fluent names, like $(sw(1), light)$, pointing out that $sw(1)$ may causally affect $light$. Although we will not detail here the method, the above relationships (8.62)-(8.62), for instance, are actually obtained from influence pairs $\{(sw(1), light), (sw(2), light)\}$ applied to the constraint (8.61).

Finally, we will also consider one more example presented in [107] whose purpose was to show the real need for the distinction, inside each rule (8.58), between the triggering effect A and the condition ϕ.

Example 18 **(The trapdoor)** In order to hunt a turkey, we use this time a manually activated trapdoor. When the turkey is at_trap and the trapdoor is open ($trapdoor\_open$, the ground underneath the trapdoor is designed so that the turkey cannot be alive: $\overline{alive}$. The possible actions we can perform correspond to $open$ the trapdoor and to $entice$ the turkey. However, the turkey would never kill himself by moving towards the trapdoor. In other words, it is the change in $trapdoor\_open$ that influences $alive$, but not the change in at_trap. □

This example is simply formalized with the constraint:

$$at\_trap \wedge trapdoor\_open \supset \overline{alive}$$

which, together with the influence information $(trapdoor\_open, alive)$, leads to the single causal rule:

$$trapdoor\_open \textbf{ causes } \overline{alive} \textbf{ if } at\_trap$$

Assuming that the actions are described as the triples:

$$\begin{aligned} \langle\{\overline{trapdooropen}\}, entice, \{trapdooropen\}\rangle \\ \langle\{\overline{at\_trap}, alive\}, entice, \{at\_trap, alive\}\rangle \end{aligned}$$

it is easy to see that the application of the algorithm for causal relationships yields the expected results: when we perform $open$ in state $\{alive, at\_trap, \overline{trapdoor\_open}\}$ the turkey is killed, whereas when we perform $entice$ in $\{alive, \overline{at\_trap}, trapdoor\_open\}$ we obtain no successor state. In this way, $\overline{trapdoor\_open}$ acts as an implicit qualification for $entice$. Of course, the keypoint for obtaining this behavior is that we have not included the analogous causal relationship:

$$at\_trap \textbf{ causes } \overline{alive} \textbf{ if } trapdoor\_open$$

8.7.2 Comparison

In order to compare Thielscher's approach with pertinence, we will begin first focusing on the representation of caused/non-caused facts. Clearly, the idea of *effect* (each element of the set E used for application of causal relationships) is strongly related to the concept of being pertinent. In fact, as the effect also represents a truth value, it actually corresponds to the representation $caused(F,V,I)$. In this way, the causal relationships (8.59)-(8.60) can be seen as the rules:

$$holds(up(2),\mathtt{t},I) \wedge caused(up(1),\mathtt{t},I) \supset caused(open,\mathtt{t},I)$$
$$holds(up(1),\mathtt{t},I) \wedge caused(up(2),\mathtt{t},I) \supset caused(open,\mathtt{t},I)$$

which correspond to an alternative representation we saw in pertinence calculus (formulas (6.20) and(6.21)). Notice that we also saw something similar in Denecker et al's approach, but in that case *holds* predicate actually referred to the previous situation, and not to the resulting one. To emphasize how the idea of effect coincides with the concept of pertinence, let us represent the trapdoor example using $\mathcal{P}$-language:

$$trapdoor\_open \quad \textbf{if} \quad open \ \textbf{after} \ \overline{trapdoor\_open} \tag{8.62}$$
$$at\_trap \quad \textbf{if} \quad entice \ \textbf{after} \ \overline{at\_trap} \wedge alive \tag{8.63}$$
$$alive \quad \textbf{if} \quad entice \ \textbf{after} \ \overline{at\_trap} \wedge alive \tag{8.64}$$
$$\overline{alive} \quad \textbf{if} \quad trapdoor\_open \wedge at\_trap \wedge \overline{!at\_trap} \tag{8.65}$$
$$\bot \quad \textbf{if} \quad trapdoor\_open \wedge at\_trap \wedge alive \tag{8.66}$$

The three first rules (8.62)-(8.64) correspond to effect axioms and follow exactly the definition of *entice* and *open* done in Thielscher's example. The shape of (8.64) may seem unnatural[10] at a first sight, but its purpose is to act as a qualification constraint on action *entice*. The constraint is simulated by rule (8.66). As the rules are acyclic, we may apply the operational interpretation, which coincides with the four semantics presented in this work. It is easy to see that the two transitions commented before (opening the trap with the turkey above, and enticing the turkey with an open trap) yield the same results. Rule (8.65) shows some similarity with (6.36) from the alarm example: we are requiring that *at_trap persists* true.

As a representational limitation of causal relationships, it is evident that they do not allow referring to the previous state. In other words, the ' **after** ' precondition we used for $\mathcal{P}$-language rules cannot be represented when using causal relationships. However, the extension of Thielscher's formalism for allowing this feature is straightforward, just allowing an **after** clause as in $\mathcal{P}$-language.

Thielscher's causal relationships do not seem to be intended for using the distinction effect/non-effect as a representational feature. This distinction is implicitly used in the rule notation, but there is no way to assert, for instance, that two facts have simultaneously become effects (they both have been caused) or that a given fact *is not* an effect. Note that all these cases can be simply represented in pertinence rules by an explicit inclusion of pertinence literals. Nevertheless, apart from this minor representational limitation, it is easy to see that causal relationships satisfies *all* the pertinence postulates. For instance, notice how P5 (the most problematic postulate in many cases) is directly implemented in the definition of rule applicability: rules are never applied without a triggering effect, which plays the role of pertinence of the condition. Besides, rule applicability does not rely on any additional transformation on the rule condition,

[10] For a perhaps more natural representation of this domain, see appendix B.

as happened with TAL (which required a change in truth value for the condition) or with Event Calculus (which relied on a change in the derived fluent value). As a result, it is possible to express the account balance scenario in a direct way:

$$transac(Y) \textbf{ causes } balance(X+Y) \textbf{ if } \top \textbf{ after } balance(X)$$

assuming the extension of causal relationships to cope with the **after** clause. The possibility of referring to the previous situation is, indeed, essential for this example. Using instead the rule:

$$transac(Y) \textbf{ causes } balance(X+Y) \textbf{ if } balance(X)$$

for a similar purpose, would clearly fail: whenever $transac(Y)$ becomes an effect, we would get an unlimited number[11] of intra-state transitions increasing the value for *balance*.

A very interesting feature is the use of influence information, which can be related to the definition of pertinence of a formula or to the initiation of complex formulas we saw in Denecker's approach. Notice that, without influence information, the representation of causal relationships is quite cumbersome, since we must express all the possible combinations of effect literals that may trigger the same rule condition. For instance, the relay example in 8.1 needs 13 causal relationships, although they are actually generated from 3 state constraints. So, the use of influence information together with state constraints is essential for a comfortable description of the causal dependences in the domain. However, this representation has different properties with respect to complex initiations or pertinence. One of the disadvantages is that one must always use a state constraint together with the causal rules, while both kind of expressions seem to be of a different nature and for different purposes[12]. Besides, we force to include a description of influence pairs which may not always be so clear a priori and so, the process may be not very elaboration tolerant. On the contrary, when using $\mathcal{P}$-language rules, these influence pairs are somehow automatically extracted from the representation, and so, the addition of new rules does not force us to explicitly reconsider the pairs for expressing dependences among fluents.

Although the "effect"-triggering mechanism behaves similarly to pertinence, there are, however, some major differences which deserve to be commented. The main objection is perhaps that causal relationships allow different values for the same fluent at the same situation. For instance, as we saw in the relay example, one of the possible outcomes of closing switch 1 is that fluent *light* momentarily changes to true, although it finally comes back to false. This behavior seems to break one of the most important abstractions handled for reasoning about actions: the concept of *state simultaneity*. As we explained in the introduction, causality has more to do with an abstraction than with a thorough representation of the physical behavior of the system. The aim of abstraction has a clear practical orientation: the higher is the abstraction level used in a problem representation, the easier will be the process for finding a solution. If we are really interested in representing momentary changes of a fluent (for instance, changes in the light when the detector is added), then we should perhaps think about augmenting the grain detail, and considering each intermediate step as a state itself. Thielscher's approach establishes instead something like a two-level hierarchy of states: "first class" or steady states; and intermediate or unstable states. In this way, ramifications are understood as delayed effects obtained in an

[11] A similar example of this infinite firing of causal relationships was also presented in [28] (page 33) where they used instead an integer counter.

[12] In fact, this was extensively discussed in [28], where they claimed the need for separating causal rules from state constraints as two completely independent representation tools.

intra-state transition sequence, considering all the possible orderings for the relative delays[13]. In fact, as later explained in [108], this distinction between stable and intermediate states forces us to classify the causal constraints into *steady* and *stabilizing*, respectively depending on whether they are totally simultaneous or they may experiment a causal lag.

The work presented in this thesis relies on assuming the state simultaneity hypothesis, abstracting any intermediate unstable variations. However, our pertinence formulation provides more information than other approaches which consider the state as exclusively delimited by fluent values. Think, for instance, in the representation of the relay domain using $\mathcal{P}$-language using rules:

$$
\begin{array}{rll}
sw(N) & \textbf{if} & toggle(N)\ \textbf{after}\ \overline{sw(N)} \\
\overline{sw(N)} & \textbf{if} & toggle(N)\ \textbf{after}\ sw(N) \\
light & \textbf{if} & sw(1) \wedge sw(2) \\
\overline{light} & \textbf{if} & \overline{sw(1)} \\
\overline{light} & \textbf{if} & \overline{sw(2)} \\
relay & \textbf{if} & sw(1) \wedge sw(3) \\
\overline{relay} & \textbf{if} & \overline{sw(1)} \\
\overline{relay} & \textbf{if} & \overline{sw(3)} \\
\overline{sw(2)} & \textbf{if} & relay
\end{array}
$$

These rules are acyclic, and so, the four semantics we studied in this work actually coincide. Furthermore, we can use the operational interpretation (algorithm in figure 5.2 to compute any successor state. In this way, performing action $toggle(1)$ in the initial state depicted in figure 8.1, we obtain that the light is caused to be off ($light$ is false and pertinent). Pertinence postulates allow us to interpret this fact by concluding that the light has experimented a causal intervention, i.e., that we *cannot guarantee* that the light has persisted off. Notice that this is, somehow, an abstraction for the different paths that we followed when using causal relationships: nondeterministically assuming that either the relay was faster than the lamp (and so, the lamp persists off) or vice versa (and so, persistence of $\overline{light}$ cannot be guaranteed). As we must reduce here everything to a single successor state, pertinence provides at least a hint about a possible causal intervention. In fact, this hint is sometimes more interesting than a detailed description, since in a real system, the relative delay differences between the lamp and the relay could be variable, or, we could pass through intermediate states in which the light is neither fully on nor fully off. Thus, we claim that the information represented by $pert(light, \mathtt{p}, 1)$ is enough to allow a causal understanding while simultaneously preserving an appropriated abstraction level.

Another interesting lesson we can draw from this example is that, contrarily to the four alternative semantics we have studied, which lead to determinism under an acyclic set of rules, Thielscher's approach is generally nondeterministic, even when rules are acyclic. Although it is true that, in most cases, we usually do not want to represent all the possible relative delays among

[13] The idea of causality as a propagation of delayed effects has also been studied in other approaches. For instance, as commented in the introduction, the historical paper [81] already included the idea of *causal assertions* which were based on this principle of delayed effects. As another example, the more recent proposal presented in [98] is very close to causal relationships. It defines a semantics in which, in order to compute the successor state, a *cascade* of sub-transitions is activated until a stable point is reached. In [90], a similar idea is introduced, but the "intermediate" transitions become usual ones. This is possible because ramifications are understood as delayed effects whose justification is obtained via a so-called *natural action*, i.e., a cause which does not need any external intervention, but only the time passing.

causal propagations, it is also true that for obtaining the same behavior, the use of an explicit representation of the intermediate states together with the incorporation of nondeterministic rules would probably be much more complicated. So, if we are really interested in the study of momentary delays, Thielscher's algorithm could be an alternative option to go further than pertinence, which, as we have seen, just provides a hint of what may have happened. Thus, as future work, it could be interesting the analysis and implementation of a fifth semantics for $\mathcal{P}$-rules using a similar algorithm to causal relationships.

One objection about we could perhaps do with respect to the treatment of causal delays when using causal relationships is the lack of homogeneity between direct and indirect effects. Consider, for instance, the simpler lamp circuit represented in figure 1.1, and its representation into causal relationships:

$$sw(1) \textbf{ causes } light \textbf{ if } sw(2) \tag{8.67}$$

$$sw(2) \textbf{ causes } light \textbf{ if } sw(1) \tag{8.68}$$

$$\overline{sw(1)} \textbf{ causes } \overline{light} \textbf{ if } \top \tag{8.69}$$

$$\overline{sw(2)} \textbf{ causes } \overline{light} \textbf{ if } \top \tag{8.70}$$

It is clear that by simultaneously toggling both switches, the lamp remains off. The facts $sw(1)$ and $\overline{sw(2)}$ are considered right from the very beginning, both in the intermediate state S and in the set of effects E. As a result, only rule (8.69) is applicable, switching off the light. However, we do not obtain an alternative application in which the light is momentarily on. The question here is, why not to consider also a possible delay between both switches, as it happened in the relay example? To emphasize this difference, notice that if we make a further elaboration, by adding the rules:

$$moveup(N) \textbf{ causes } sw(N) \textbf{ if } \top$$
$$movedown(N) \textbf{ causes } \overline{sw(N)} \textbf{ if } \top$$

then, surprisingly, the direct effects $\{movedown(1), moveup(2)\}$ actually lead to nondeterminism (there is a path in which the light is momentarily on) just because $sw(1)$ and $\overline{sw(2)}$ are now indirect effects instead of direct ones.

8.8 Causal explanation

Together with Thielscher's approach and Lin's work, the other main proposal for solving the ramification problem that appeared around 1995 was McCain and Turner's causal rules [69], partly inspired by Geffner's work [37]. This approach relied on a modal conditional operator, $\phi \Rightarrow \psi$, used for a transition-based description of the system behavior. The nonmonotonic behavior was achieved by a fixpoint condition (for the successor state), which served both for implementing inertia and for providing the '$\Rightarrow$' operator with the behavior of an inference rule.

Later, in [70], this initial approach was considerably simplified giving rise to the so-called *Causal Explanation* logic which was thereafter succesfully used for satisfiability planning [71] and as a basis of the $\mathcal{C}$ action language [46] (a successor of $\mathcal{A}$-language that allows dealing with ramifications) and its implementation, the *causal calculator* (`CCALC`) [23]. Although it is frequent to identify both approaches,[69] and [70], as a same formalism, they actually present important differences. For instance, Causal Explanation has a simpler semantics and its fixpoint condition does not depend on any actions or transition-oriented framework (it can be used for other kind of defaults different from inertia). In fact, one more further generalization of both approaches was introduced in [111], receiving the name of *Universal Causation Logic* (UCL).

Among this variety of related approaches, we will pay in this section more attention to Causal Explanation [70], although we will simultaneously provide some informal insight about its formulation into UCL. The interest of the latter is that it provides constructions like $\mathbf{C}\,\phi$ (where $\mathbf{C}$ is a modal S5-necessity operator) to stand for "ϕ is caused."

As commented in [70], the intuitive idea behind Causal Explanation arises from the distinction between two kind of causal conditionals:

i) *the fact ϕ causes the fact ψ*

ii) *Necessarily, if ϕ happens to be true then ψ is caused*

Using UCL notation, these two types of conditional would respectively correspond to the formulas:

$$\begin{array}{rcl} \mathbf{C}\,\phi & \supset & \mathbf{C}\,\psi \\ \phi & \supset & \mathbf{C}\,\psi \end{array}$$

The first type of conditional is similar to an inference rule and, in fact, is equivalent to the causal rules used in the first approach [69] introduced by McCain and Turner. Note that the antecedent condition is required to be caused. Sometimes, however, we do not need to identify the causes for all the facts and it is enough with knowing sufficient conditions in order to establish the caused consequents. Thus, in this second type of description, the condition ϕ does not need to be caused: it is just an *explanation* for $\mathbf{C}\,\psi$. This second orientation corresponds to Causal Explanation, where $\phi \supset \mathbf{C}\,\psi$ is simply denoted as $\psi \leftarrow \phi$.

The formal description for Causal Explanation is very simple. Syntactically, we handle an extension of propositional logic with a new conditional operator $\leftarrow$. We assume that this operator is not nested[14], that is, for any formula $\psi \leftarrow \phi$ (called *causal rule*), both the antecedent and the consequent cannot contain $\leftarrow$ in their turn.

The semantics is defined as follows. Given a set D of these causal rules and any propositional interpretation M, the theory D^M is defined as:

$$D^M = \{\psi \mid (\psi \leftarrow \phi) \in D, M \models \phi\}$$

where $\models$ stands for classical satisfaction of formulas. Clearly, D^M is a classical propositional theory. Using the UCL point of view, D^M would stand for the set of formulas $\mathbf{C}\,\psi$ "known to be caused," after using M to interpret all the explanations.

Definition 40 (Causally explained model) An interpretation M is a *causally explained model* of a causal theory D iff M is the unique model of D^M. □

Intuitively, we require now that, once M has fixed the explanations, the resulting caused formulas $\mathbf{C}\,\phi$ must be exactly *all* the consequences derived from the single model M. In other words, causally explained models satisfy $\phi \equiv \mathbf{C}\,\phi$. This is where the idea of "universal causation" actually comes from: we want *every* consequence to be finally caused.

[14] One of the ways in which UCL generalizes Causal Explanation is, for instance, by allowing nested conditionals.

Another important observation is the strong similarity of this definition with the already seen fixpoint conditions used in logic programming. As a matter of fact, we will see later that Causal Explanation can be seen as a generalization of the idea of *supported model* or its syntactic counterpart, Clark's Completion.

The following would be some typical uses of causal rules.

$$holds(up(N), I+1) \leftarrow occurs(toggle(N), I) \wedge \neg holds(up(N), I) \tag{8.71}$$
$$\neg holds(up(N), I+1) \leftarrow occurs(toggle(N), I) \wedge holds(up(N), I) \tag{8.72}$$
$$holds(open, I) \leftarrow holds(up(1), I) \wedge holds(up(2), I) \tag{8.73}$$
$$holds(F, I+1) \leftarrow holds(F, I+1) \wedge holds(F, I) \tag{8.74}$$
$$\neg holds(F, I+1) \leftarrow \neg holds(F, I+1) \wedge \neg holds(F, I) \tag{8.75}$$

The first pair of rules would be an example of effect axioms. Rule (8.73) is a ramification rule, whereas rules (8.74)-(8.75) represent the encoding of inertia. Notice the curious shape of this representation: whenever we want to derive a default consequence, this one must be included as part of the condition in the antecedent. In fact, similar constructions are usually included for completing the initial situation:

$$holds(F, 0) \leftarrow holds(F, 0) \tag{8.76}$$
$$\neg holds(F, 0) \leftarrow \neg holds(F, 0) \tag{8.77}$$

or for generating all the possible action executions:

$$occurs(A, I) \leftarrow occurs(A, I) \tag{8.78}$$
$$\neg occurs(A, I) \leftarrow \neg occurs(A, I) \tag{8.79}$$

For instance, the formula(8.76) should be read as "by default, fluent F can be assumed to be true at 0."

8.8.1 Literal completion. Relation to Logic Programming

Great part of the success of Causal Explanation is that, when rule consequents are limited to literals, there exists a transformation, called *literal completion*, that allows the conversion of any causal domain into a classical propositional theory.[15]. The idea of literal completion can be seen as a straightforward generalization of Clark's completion for logic programming, but applied to literals instead of atoms.

Definition 41 (Literal Completion) Let D be a set of causal rules where all the consequents are literals. Then, the *literal completion* of D, LCOMP$[D]$, corresponds to the propositional theory that contains a formula:

$$L \equiv \phi_1 \vee \cdots \vee \phi_n$$

for any possible literal L formed with atoms in the signature, being the ϕ_i are all the antecedents such that $(L \leftarrow \phi_i) \in D$ (as always, the empty disjunction is equivalent to $\perp$). □

[15] In fact, this is the working principle of the tool `CCALC`, which is a Prolog program consisting of an interpreter for the C action language plus a module for grounding rules and processing their literal completion. The main inference work is delegated afterwards to an external call to some propositional prover like Crawford's NTAB [27, 26], Zhang's SATO [115, 114] or more recently Malik et al's CHAFF [83, 24]

Property 16 *Let D be a set of causal rules where all the consequents are literals. Then, the causally explained models of D are the classical models of its literal completion,* LCOMP$[D]$. $\Box$

We can define Clark's completion as a particular case of literal completion, where negative literals are always assumed by default, that is, we include the axiom schemata:

$$\neg p \leftarrow \neg p \tag{8.80}$$

for any atom in the signature.

As we explained in the background, when considering Clark's completion, the logic program operators '$\leftarrow$', ',' and '*not* ' are respectively interpreted as the propositional connectives '$\supset$', '$\wedge$' and '$\neg$'. We will implicitly assume this equivalence inside this section.

Theorem 12 *Let P be a normal logic program. Then, for any propositional interpretation I, $I \models$* COMP$[P]$ *iff* $I \models$ LCOMP$[P \cup (8.80)]$.

Proof

It is straightforward. Note that the literal completion LCOMP$[P]$ consists of the completion for positive literals, which in this case is COMP$[P]$, plus the completion for negative literals, which in this case is a set of formulas like $\neg p \equiv \neg p$, which are propositional tautologies. $\Box$

Note that this result is similar to the well known relation between answer sets and stable models ([39], section 6): the stable models of a normal logic program P are the answer sets of:

$$P \cup \{\neg p \leftarrow not\ p \mid p \in \Sigma\}$$

being Σ the propositional signature. In a similar way, answer sets can be defined in terms of stable models, by renaming the negated literals $\neg p$ into a new type of atom $\overline{p}$. This same operation can be done for defining the literal completion in terms of Clark's completion:

Theorem 13 *Let D be a set of causal rules where all the antecedents are conjunctions of literals and all the consequents are literals. Let Σ be the propositional signature for D and let D' be the result of replacing in D each literal $\neg p$ by the atom $\overline{p}$ plus the addition of the following rule schemata:*

$$\mathbf{false} \leftarrow p,\ \overline{p} \tag{8.81}$$

$$\mathbf{false} \leftarrow \text{not } p,\ \text{not } \overline{p} \tag{8.82}$$

where **false** *is an additional atom* **false** $\notin \Sigma$. *Then, the models of* LCOMP$[D]$ *are the models of* COMP$[D'] \cup \{\neg\mathbf{false}\}$ *(modulo the reified literals representation).*

Proof

The formula $\neg\mathbf{false}$ means that $\mathbf{false} \equiv \bot$ and so (8.81) and 8.82 are equivalent to $\neg(p \wedge \overline{p}$ and $\neg(\neg p \wedge \neg\overline{p})$ respectively. But the conjunction of this pair of formulas is equivalent to $\overline{p} \equiv \neg p$. Now, it is easy to see that if we apply propositional universal substitution, COMP$[D'] \cup \{\neg\mathbf{false}\}$ is equivalent to LCOMP$[D] \cup \{\overline{p} \equiv \neg p | p \in \Sigma\} \cup \{\mathbf{false} \equiv \bot\}$. Therefore, for any model I of LCOMP$[P]$, the interpretation $I' = I \cup \{\overline{p} | p \in \Sigma - I\}$ is model of COMP$[D'] \cup \{\neg\mathbf{false}\}$. And vice versa, for any I' model of COMP$[D'] \cup \{\neg\mathbf{false}\}$, the interpretation $I = I' \cap \Sigma$ (which contains less atoms) is a model of LCOMP$[D]$. $\Box$

Notice that we must require (8.81) to avoid inconsistence and (8.82) to obtain a unique model. Otherwise, atoms p and $\overline{p}$ would not have any logical connection. Another interesting observation is that, in the encoding of answer sets into stable models, only rule schemata (8.81) is actually added. The explanation for this is that answer sets are not complete (i.e., an answer set represents several models) and so rule (8.82) is not included.

The actual relevance of this pair of theorems is that, when we reduce McCain and Turner's causal rules to deal with literals (as done, for instance, in `CCALC`) the representativity is exactly the same as when using Logic Programming supported models (i.e., Clark's completion). Therefore, an hypothetical alternative semantics of $\mathcal{P}$-language relying on Causal Explanation would not be of any interest, since it would simply correspond to the already seen semantics based on Clark's completion. Besides, these results also help to clarify the real orientation of McCain and Turner's causality: roughly speaking, since we obtain the same expressivity as in Logic Programming, Causal Explanation seems to be thought as an alternative for implementing *inference rules* rather than as a way of representing change in a dynamic domain.

8.8.2 Pertinence postulates in Causal Explanation

In order to study Causal Explanation under pertinence postulates, the first difficulty is to identify how caused facts are represented. In the previous comparisons, this was practically straightforward, since all of them handled a particular change predicate which could be related to pertinence $pert(F, V, I)$. Unfortunately, such a change predicate cannot be found in Causal Explanation. Each causally explained model contains fluent values but not any information about how these values have been obtained.

Nevertheless, under UCL notation, when we use the modal operator $\mathbf{C}$ to interpret causal rules, we actually handle the concept of "caused formula." At a first glimpse, this seems to point out that formulas like $\mathbf{C}\, open$ or $\mathbf{C}\, \neg open$ to atoms like $caused(open, \mathtt{t})$ or $caused(open, \mathtt{f})$, respectively. This relation was studied by Turner (see theorem 5.23, page 177 in [110]) which established an interesting correspondence between causal explanation rules (under their UCL shape) and Lin's formulation (without allowing *caused* in the conditions). Turner's result, however, is not a complete correspondence: although the truth values for fluents we obtain in UCL correspond exactly to the *holds* atoms we get under Lin's formulation, no relation is established for *caused* atoms. Of course, the latter is not surprising at all: UCL requires that all the consequences must be finally caused, whereas Lin's approach not. Notice how, despite the similarity in the shape of the rules (when the condition is true, the consequent must be caused), the concept of "caused" has a *completely different* meaning in both approaches. On the one hand, any causally explained model satisfies $\mathbf{C}\, \phi \equiv \phi$ which means that "caused" does not provide any additional information with respect to the fluent truth values, if we exclusively look at the set of selected models. On the other hand, the models in Lin's approach may contain fluent values that are not caused. Furthermore, the usually *must contain* them, since the application of inertia for some fluent f is only enabled when $\neg caused(f, \mathtt{t}, S) \wedge \neg caused(f, \mathtt{f}, S)$. So, in Lin's approach there is an important qualitative difference between the formulas $holds(f, S)$ and $caused(f, \mathtt{t}, S)$ whereas in causally explained models this difference simply *does not exist*.

From this discussion, it is easy to see that the purpose of the $\mathbf{C}$ operator handled in UCL does not correspond with the idea we had in mind when introducing predicate *pert*. The $\mathbf{C}$ operator is exclusively intended for fixing the final selected models, but it cannot be used as a relevant information with respect to nonmodal formulas. This difference becomes even clearer if we observe that, under UCL terminology, as everything must be finally caused, this also includes those facts resulting from the universal frame rules (8.74)-(8.75). In other words, inertia is one more causal rule. As we could see when comparing Causal Explanation to logic programming, it seems that the idea of causality here is more related to defining some conditional operator that provides the behavior of inference rules plus some general purpose nonmonotonic behavior.

As a result, pertinence postulates are simply not applicable neither to Causal Explanation nor UCL, unless we additionally include some predicate to differentiate between change and inertia. To see how these two different cases are completely mixed in Causal Explanation, we can study, for instance, what would happen if we tried to apply postulate P2. As an example, consider the formulas (8.71)-(8.75). It is easy to see that, in a given transition, it is possible to apply both the ramification rule (8.73) and the inertia rules (8.74)-(8.75) *simultaneously*. For instance, if we have just opened the suitcase and we perform a *wait* action, the resulting fact $holds(open, 1)$ is justified both by the ramification rule and by the inertia law (8.74).

8.9 Schwind's comparative

To end up with the comparative study, we include some remarks about Schwind's paper [100] which is perhaps the most complete up-to-date comparison among different causal approaches existing in the literature. Schwind's work covers the three main causal approaches arisen around 1995, Lin's *caused* predicate, Thielscher's causal relationships and McCain and Turner's causal rules[16], all of them also covered in this work, plus an additional approach [44] developed by Schwind herself together with Giordano and Martelli. We will particularly focus in the criteria used in that overall comparison, rather than on the study of this last work. The reason for this is that, apart from its temporal representation relying on Dynamic Logic [49], it uses the same modal notation as the UCL encoding of Causal Explanation, and so, it does not introduce any relevant difference with respect to pertinence postulates.

The main difference between Schwind's comparative and the study included in this section is the change in the orientation. While our comparison is more oriented to the change-based understanding of causality, Schwind's work is *completely inferential* because it is exclusively focused on the properties of causal conditionals from the point of view of their inferred consequences. This means that there is no special interest in distinguishing whether a fluent value has been caused or it has persisted. Furthermore, in order to study these inference properties, there is not even any actual need for considering a dynamic framework at all. In this way, Schwind presents nine criteria (section 2.1 in [100]) for characterizing causal conditionals without including in their definitions any single mention to actions, fluents, situations nor inertia. Notice how, on the opposite, these concepts are continuously handled in the pertinence postulates.

Forgetting this difference in orientation, we can at least analyze the pertinence rules from the point of view of the nine inferential criteria proposed by Schwind. If we write '$\Rightarrow$' to stand for any generic conditional operator, these criteria can be enunciated as follows:

1. *Monotonicity.*

$$\frac{\phi \Rightarrow \psi}{\phi \wedge \gamma \Rightarrow \psi}$$

[16] In fact, McCain and Turner's approach is studied both under the point of view of the causal rule as a conditional operator which, for instance, happens to be monotonic, and under the point of view of the induced inference relation which, of course is nonmonotonic. We claim that studying the latter does not make much sense since it seems a quite different thing to analyze the properties of conditional operators from considering the properties of their induced inference relations. For instance, we can equally consider the induced inference relations for Lin's and Thielscher's approaches and, of course, they will result being nonmonotonic.

2. *Transitivity.*

$$\frac{\phi \Rightarrow \psi, \ \psi \Rightarrow \gamma}{\phi \Rightarrow \gamma}$$

3. *Contraposition.*

$$\frac{\phi \Rightarrow \psi}{\neg\psi \Rightarrow \neg\phi}$$

4. *Reflexivity.*

$$\frac{\top}{\phi \Rightarrow \phi}$$

5. *Conjunction of preconditions.*

$$\frac{\phi \wedge \gamma \Rightarrow \psi}{(\phi \Rightarrow \psi) \text{ or } (\gamma \Rightarrow \psi)}$$

6. *Conjunction.*

$$\frac{\phi \Rightarrow \psi, \ \phi \Rightarrow \gamma}{\phi \Rightarrow \psi \wedge \gamma}$$

7. *Reasoning by case.*

$$\frac{\phi \vee \gamma \Rightarrow \psi}{\phi \Rightarrow \psi, \ \gamma \Rightarrow \psi}$$

8. *Right weakening (RW).*

$$\frac{\phi \Rightarrow \psi, \ \models (\psi \supset \gamma)}{\phi \Rightarrow \gamma}$$

9. *Left Logical Equivalence (LLE).*

$$\frac{\phi \Rightarrow \psi, \ \models (\phi \equiv \gamma)}{\gamma \Rightarrow \psi}$$

Of course, a first problem we must face is to clarify the formal meaning of each derivation rule:

$$\frac{\alpha_1, \ldots, \alpha_n}{\beta_1, \ldots \beta_m} \tag{8.83}$$

s. Although this is not done in Schwind's, it can be easily deduced from the context. For instance, for those causal approaches that rely on a classical logic representation (like Lin's *caused* predicate) the above rules are understood as classical entailment, that is $\{\alpha_1, \ldots, \alpha_n\} \models \{\beta_1, \ldots, \beta_m\}$. Unfortunately, this is not possible in many of the approaches, like for instance Causal Explanation (not UCL), Inductive Causation or Thielscher's approach, since they do not define satisfaction for a causal conditional, but only for standard propositional formulas. In these cases, the causal rules can be considered as part of the semantics, and we can only study whether their addition will vary the resulting (propositional) consequences or not. Furthermore, even when classical logic is used as the underlying monotonic framework, we can use, instead of classical entailment, the (nonmonotonic) entailment induced by the selected models, let us denote it as $\alpha \approx\!\!\!\mid \beta$. Of course, since selected models are a subset of the classical models, classical entailment is stronger than nonmonotonic entailment, i.e., $\alpha \models \beta$ implies $\alpha \approx\!\!\!\mid \beta$.

In this section, we will use the following criterion:

> A derivation rule like (8.83) represents that, for any theory T (consisting of causal rules and observations) where all the $\alpha_i \in T$, then the consequences of T and $T \cup \{\beta_1, \ldots, \beta_m\}$ coincide.

In other words, rather than "concluding" the rules in β, we assert that their addition is "harmless," that is, it does not affect to our previous representation (which includes the α_i).

We study now each property separatedly.

1. Monotonicity.

   For simplicity sake, we begin considering $\mathcal{P}$-rules without precondition, that is, of shape E **if** C. Although the encoding of this expression relies on classical implication (for the circumscriptive semantics) or a logic programming rule (in the other cases), which are both monotonic conditional operators, the $\mathcal{P}$-rule itself is *surprisingly nonmonotonic.* The reason for this is the implicit pertinence of the rule condition. As a counterexample of monotonicity, consider the following variation of the gong example, where we simply add a new instrument: a whistle. We have a domain represented by the set of rules R containing:

$$gong \ \textbf{if} \ strike \tag{8.84}$$
$$dance \ \textbf{if} \ gong \tag{8.85}$$
$$\overline{dance} \ \textbf{if} \ finish \tag{8.86}$$
$$whistle \ \textbf{if} \ blow \tag{8.87}$$

   where $strike$, $blow$ and $finish$ are actions and $gong$, $whistle$ and $dance$ are fluents. Consider the expression (8.85), which is the only ramification rule. If we add a fourth rule by strengthening its condition:

$$dance \ \textbf{if} \ gong \wedge whistle \tag{8.88}$$

   the resulting domain $R' = R \cup (8.88)$ *does not yield the same consequences* (the resulting automaton is different). To see this, consider the initial state $\sigma_0 = \{gong, \overline{dance}, \overline{whistle}\}$ and that we perform action $blow$. Using the rules in R, the result is that everything persists, excepting $whistle$ which becomes true. However, the same scenario using $R \cup (8.88)$ leads to $\sigma_1 = \{gong, dance, whistle\}$ and $\pi_1 = \{\overline{!gong}, !dance, !whistle\}$, that is, the ballerina starts dancing.

   The explanation for this is simple: once we include $whistle$ in the condition, we are providing a new possible cause for $dance$ which can make the rule applicable while $gong$ persists. In this way, although $gong$ is true, rule (8.85) is not applicable because $gong$ is not pertinent (not caused). However, in R', rule (8.88) has as condition $gong \wedge whistle$ which is true ($gong$ persists true and $whistle$ is caused true by $blow$), but additionally *becomes pertinent.* If we wanted to express that *whistle* is a simple additional condition which *must not act as a cause*, then we should rather formulate the rule as:

$$dance \textbf{ if } gong \wedge !gong \wedge whistle \tag{8.89}$$

asserting, now in an explicit way, that the pertinence is due to *gong* (at least), or alternatively as:

$$dance \textbf{ if } gong \wedge whistle \wedge \overline{!whistle} \tag{8.90}$$

which would further assert that whistle must persist true.

Despite of this nonmonotonicity, we do not obtain the usual counterfactual behavior of most nonmonotonic conditional operators. Typically, nonmonotonicity is used for expressing things like:

$$\begin{aligned} rain &\Rightarrow wet \\ rain \wedge umbrella &\Rightarrow \neg wet \end{aligned}$$

that is, rules with stronger conditions may *override* those with weaker requisites which, in this way, may become defeated. When using $\mathcal{P}$-rules, this is not exactly the case. For instance, consider that R'' contains R plus the rule:

$$\neg dance \textbf{ if } gong \wedge whistle \tag{8.91}$$

that is, the combination of a *gong* and a *whistle* makes the ballerina stop dancing. Assume we simultaneously blow the whistle and strike the gong. Then both (8.85) and (8.91) are applicable leading to inconsistence. Something similar happens if we just strike the gong while the whistle persisted sounding. However, if we blow the whistle *while the gong persisted true*, then rule (8.85) is not applicable and we get that the ballerina stops dancing. In other words, this pseudo-counterfactual behavior depends on which conjuncts of the condition are pertinent or not.

Let us consider now $\mathcal{P}$-rules with precondition. As the precondition of a rule is not modified in the rule encoding (pertinence does not affect to the precondition), and as this encoding uses monotonic conditional operators (either classical implication or logic program rules) it is easy to see that the following applies:

$$\frac{E \textbf{ if } C \textbf{ after } D}{E \textbf{ if } C \textbf{ after } D \wedge \alpha}$$

for any conjunction[17] α of fluent facts. In other words, $\mathcal{P}$-rules are monotonic with respect to their preconditions.

2. Transitivity.

Again, we study first precondition-free rules. As the rule consequent is always a fluent fact, we must restrict the transitivity study considering instead:

$$\frac{E \textbf{ if } C,\;\; F \textbf{ if } E}{F \textbf{ if } C}$$

[17] In fact, any propositional combination of fluent facts can be properly handled by obtaining its disjunctive normal form and transforming each negation $\neg holds(f, v)$ into the disjunction $holds(f, v_1) \vee \cdots \vee holds(f, v_n)$ for the rest of values of fluent f which are not the value v.

where E and F are fluent facts. It is easy to see that this is always true. The encoding of E **if** C and $,F$ **if** E would be:

$$\begin{array}{rcl} C\wedge !C & \rightarrow & E \\ C\wedge !C & \rightarrow & !E \\ E\wedge !E & \rightarrow & F \\ E\wedge !E & \rightarrow & !F \end{array}$$

where $\rightarrow$ is either a classical implication or a logic program rule. For both conditionals, transitivity is satisfied, and so we get that we can add:

$$\begin{array}{rcl} C\wedge !C & \rightarrow & F \\ C\wedge !C & \rightarrow & !F \end{array}$$

without introducing any variation.

As for rules with precondition, of course transitivity can be proved for the case

$$\frac{E \textbf{ if } C \textbf{ after } D,\ \ F \textbf{ if } E \textbf{ after } D}{F \textbf{ if } C \textbf{ after } D}$$

being careful of not interchanging the roles of condition and precondition (note that they actually refer to different situations).

3. Contraposition.

As we explained in the introduction, one of the main goals of introducing causality is to avoid contrapositive reasoning. In fact, we saw that many of the typical representational problems handled in the literature (not exclusively ramification problems) are related to an undesired application of contrapositive reasoning. So, it is not any surprise that this property is not satisfied in any of the four proposed semantics. Consider again the original ballerina domain: rules (5.20)-(5.22). Clearly, if we add the rule:

$$\overline{gong} \quad \textbf{if} \quad \overline{dance}$$

introduces an important variation: in this case, when we order the ballerina to $finish$ the choreography we get as an indirect effect that the gong stops sounding!

Although the contraposition of a rule generally yields a different transition relation, it must be noticed that, for instance, the pair of rules:

$$\begin{array}{ccc} b & \textbf{if} & c \\ c & \textbf{if} & b \end{array}$$

do not behave as a classical equivalence $b \equiv c$. It is perfectly possible to have a state $\sigma = \{b, \overline{c}\}$ without entering intro contradiction, provided that, in this case, b persists.

Unfortunately, requiring that a given causal conditional does not allow contraposition is not strong enough to rule out possible contrapositive consequences. For example, let us consider the circumscriptive encoding of pertinence. In that case, contraposition is not allowed: the gong variation we saw above would still mean a modification in the transition relation w.r.t. the original gong example. However, we saw that for some domains (like the alarm problem, example 6) the circumscriptive encoding allowed "applying" a rule to obtain results from its consequent to its antecedent. In this sense, we consider our operational interpretation as a more reliable criterion than the contraposition property, as stated here.

4. Reflexivity.

   The meaning of this property can be easily misunderstood. For instance, under Schwind's interpretation, reflexivity is not desirable because it would mean that a given formula ϕ may cause ϕ itself. We claim that this last sentence does not correspond to the property of reflexivity but, instead, corresponds to the idea of *self-supportedness* we commented in detail in section 7.1.2. Thus, we understand that the mere presence of $\phi \Rightarrow \phi$ should not be enough to obtain as a consequence that ϕ is finally caused, unless there exists another causal chain to obtain ϕ.

   This idea is very different to trying to avoid that $\phi \Rightarrow \phi$ is a tautology (in Schwind's terminology) or, in our case, that its addition is harmless. Furthermore, it seems that we should exactly look for the opposite, that is, the addition of the formula $\phi \Rightarrow \phi$ should not modify the consequences of our previous representation, whichever it was. This would be a good hint for absence of self-supportedness. In this way, reflexivity *is satisfied* by the stable models and the well-founded encodings of pertinence, which are precisely those ones that do not allow self-supportedness.

   To show that the interesting property is to avoid self-supportedness and not reflexivity, consider the other two encodings of pertinence: circumscription and completion. As we explained in chapter (7), both may obtain self-supported conclusions, that is, a fact may become surprisingly "caused by itself." However, completion does not satisfy reflexivity, i.e., introducing a positive cycle like example 7:

   $$b \textbf{ if } b$$

   usually modifies the consequences. In fact, even circumscription does not satisfy reflexivity: we saw that while b **if** b never implies any change, an apparently equivalent variation, like example 8, b **if** $b \wedge a$, (which under a classical logic reading is still a tautology) may completely change the result for circumscription (leading to self-supportedness as we explained in section 7.1.2).

5. Conjunction of preconditions.

   Again, there exists a slight problem when formulating this property. We must clarify the meaning of the informal disjunction we have in the consequent. For instance, in Schwind's paper, the proof of this property for Lin's approach is done by relying on classical logic, simply observing that:

   $$holds(f,S) \wedge holds(g,S) \supset caused(h,v,S) \tag{8.92}$$

   classically entails (in fact, it is equivalent to):

   $$\big((holds(f,S) \supset caused(h,v,S)) \quad \vee \quad (holds(g,S) \supset caused(h,v,S)\big)$$

However, we must bear in mind that in Lin's rules, situations are universally quantified, and so we are actually handling:

$$\forall S.\ \big((holds(f,S) \supset caused(h,v,S)) \quad \vee \quad (holds(g,S) \supset caused(h,v,S)\big)$$

which *does not entail*:

$$\forall S.\ \big((holds(f,S) \supset caused(h,v,S)\big) \quad \vee \quad \forall S.\ \big(holds(g,S) \supset caused(h,v,S)\big)$$

which is the actual shape of a disjunction of two causal rules.

To avoid these problems, we will pose this property as follows. Having some set of rules R which contain, for instance:

$$E \textbf{ if } C_1 \wedge C_2$$

we want to prove that either the set of rules $R' = R \cup \{E \textbf{ if } C_1\}$ or the set of rules $R'' = R \cup \{E \textbf{ if } C_2\}$ have the same consequences than R. It is easy to see that this does not hold. Consider the case in which E, C_1 and C_2 are simply fluent atoms. When we handle R' (resp. R'') it suffices with causing C_1 (resp. C_2) to obtain E. The other condition is not needed at all (it may be even false). However, when we consider R alone, we need that one of the conditions is caused while the other is at least required to be true.

6. Conjunction.

   This property is not analyzable for $\mathcal{P}$-rules, since their consequent is only defined for fluent facts. However, we could study it for the more general case of the L^2 conditional, '$\Leftarrow$,' considering the monotonic entailment induced by L^2 models, let us write it as $\models_{\mathtt{t}}$ (remember that a model is an interpretation that assigns $\mathtt{t}$ to all the formulas, regardless their pertinence). Then, it is easy to see that:

   Theorem 14

$$\{\psi \Leftarrow \phi,\ \gamma \Leftarrow \phi\} \quad \models_{\mathtt{t}} \quad \psi \wedge \gamma \Leftarrow \phi$$

   Proof
   (See appendix A). □

   It must be noticed that the contraposition of this property does not hold, that is:

$$\{\psi \wedge \gamma \Leftarrow \phi\} \not\models_{\mathtt{t}} (\psi \Leftarrow \phi) \wedge (\gamma \Leftarrow \phi)$$

7. Reasoning by case.

   Again, $\mathcal{P}$-rules cannot be used to study this property, but we can instead analyze it for general L^2 expressions.

   Theorem 15

$$\{A \Leftarrow B \vee C\} \quad \models_{\mathtt{t}} \quad (A \Leftarrow B) \wedge (A \Leftarrow C)$$

   Proof
   (See appendix A). □

As happened before, it must be noticed that the opposite of this property is not necessarily true, i.e., we may have a model of $\{A \Leftarrow B,\ A \Leftarrow C\}$ which is not model of $A \Leftarrow B \vee C$. As a counterexample, simply consider $M(B) = \mathtt{tn}$, $M(C) = \mathtt{fp}$ and $M(A) = \mathtt{fn}$.

8. Right weakening (RW).

   One more time, RW cannot be studied for $\mathcal{P}$-rules because the consequent must be an atom (a fluent fact). Since we want to replace this atom p by some formula γ which is a logical consequence of p (in the sense of classical logic), we will easily find out that γ cannot be an atom in its turn, and so, it cannot be used as a rule effect. Thus, γ will be either some nonatomic formula, like for instance $p \vee q$, or the constant $\top$ (which cannot be used as a rule effect either). Notice that we look for *logical* consequences, i.e., we cannot use consequences derived from *nonlogical* constraints like $p \supset q$ or $p \equiv \neg r$ we may have included in the representation. To overcome this non-applicability, we move the study again to L^2 conditionals.

   It is easy to see that, when dealing with pertinence, one cannot rely on transformations due to logical equivalences. The intuitive reason for this is that pertinence of a formula is *sensitive* to the atoms occurring in it. As a result, we get that, for instance, having the rule:

$$p \wedge q \quad \Leftarrow \quad r \tag{8.93}$$

   it does not entail:

$$p \Leftarrow r \tag{8.94}$$

   although, clearly, p logically follows from $p \wedge q$. To see a counterexample, consider the L^2 interpretation $M(p) = \mathtt{tn}$, $M(q) = \mathtt{tp}$ and $M(r) = \mathtt{tp}$. The first conditional (8.93) is satisfied by M, since being the antecedent r true and pertinent, the consequent $p \wedge q$ is also true and pertinent (since q is pertinent). However, the second conditional (8.94) is valuated as false, because being r true and pertinent, in order to satisfy this rule, the consequent p should be valuated $M(p) = \mathtt{tp}$ and this is not the case (p is non-pertinent).

9. Left Logical Equivalence (LLE).

   The same reasoning as for RW is applicable to LLE, that is, logical equivalences cannot be freely added to a L^2 conditional. In fact, we have already commented this in example 3.2 (chapter 3). Assume we have the already seen rule (3.1) stand-alone:

$$d \quad \Leftarrow \quad a$$

   Although $c \vee \neg c$ is a tautology, it is easy to see that (3.1) does not entail rule (3.2), that is:

$$d \quad \Leftarrow \quad a \wedge (c \vee \neg c)$$

   For instance, the interpretation $M(a) = \mathtt{tn}$, $M(c) = \mathtt{fp}$ and $M(d) = \mathtt{fn}$ is a model of (3.1) but not of (3.2).

   This example shows the that pertinence can be classified as a relevance logic (as studied for instance in [4]) so that, the satisfaction of formulas is affected by the atoms occurring in them. Of course, from a classical logic point of view, it may seem counterintuitive the fact that the introduction of logical tautologies affects the satisfaction of formulas. However, we must always bear in mind the underlying "dynamic" understanding of causal formulas and that, as we explained in example 3.2, pertinence may mean momentary instabilities for which usual truth values may be undefined.

Chapter 9

Pertinence Action Language

In the previous chapters we have studied the relation between pertinence and causality trying to simplify the representation as much as possible. In this way, we defined a special rule syntax, we called $\mathcal{P}$-language, that was simple but expressive enough to cover most of the typical representational examples used in the literature. However, for practical purposes, $\mathcal{P}$-language must be significantly improved. For instance, we can try to handle expressions involving the fluents (and actions) values, rather than considering separatedly each conjunctive case and using auxiliary variables. As an example, consider the following typical planning problem (as extracted from [77]):

Example 19 (The missionaries and cannibals problem)
"Three missionaries and three cannibals come to a river and find a boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten. How shall they cross?" □

Assume we represent this domain with a set of actions $carry(M,C)$ pointing out that M missionaries and C cannibals are carried in the boat from one bank to the opposite one. Besides, we use a fluent $num(P,B) \in [0,3]$ to point out how many persons of type $P \in \{mis, can\}$ are in a given bank $B \in \{left, right\}$. The fluent *boatbank* is used to point out the location of the boat. Finally, for commodity sake, we include an auxiliary fluent $moved(P)$ that points out how many persons of type P are carried, i.e., if $carry(M,C)$ is true, then $moved(mis) = M$ and $moved(can) = C$.

Now consider the formulation of this domain in $\mathcal{P}$-language. We would need to represent that the boat cannot be moved when it is empty:

$$\bot \textbf{ if } carry(0,0) \tag{9.1}$$

and that the capacity cannot be exceeded:

$$\bot \textbf{ if } carry(M,C) \tag{9.2}$$

for any pair of values of M and C so that $M + C > 2$.

If we assume that the rules:

$$holds(moved(mis), M) \textbf{ if } carry(M,C) \tag{9.3}$$
$$holds(moved(can), C) \textbf{ if } carry(M,C) \tag{9.4}$$

are also included, the variations of the number of persons in the departure and destination banks are respectively represented by the rules:

$$holds(num(P,B),N-M) \quad \textbf{if } holds(moved(P),M) \quad \textbf{after } holds(num(P,B),N) \wedge holds(boatbank,B) \tag{9.5}$$

$$holds(num(P,B),N+M) \quad \textbf{if } holds(moved(P),M) \wedge holds(boatbank,B) \quad \textbf{after } holds(num(P,B),N) \tag{9.6}$$

We represent the movement of the boat using:

$$holds(boatbank,B) \textbf{ if } carry(M,C) \textbf{ after } holds(boatbank,B') \tag{9.7}$$

where B is the opposite bank to B'. Finally, we must avoid that the cannibals eat the missionaries:

$$\perp \textbf{ if } holds(num(mis,B),M) \wedge holds(num(can,B),C) \tag{9.8}$$

for any $C > M$ and $M > 0$.

Notice the problems of this representation. First, the constraints involving variables values cannot be directly represented in $\mathcal{P}$-language – in fact, variables are just an abbreviation we use to replace all the grounded instances. Second, although fluents like *num*, or *moved* have a numerical range, we actually do not take much benefit from this, since we must finally use expressions involving auxiliary variables, rather than directly the fluent name. This is more similar to the way in which arithmetic expressions are handled, for instance, in Prolog. As a matter of fact, notice that expressions like $N - M$ are also abbreviations of some grounded instance, and are not actually allowed in $\mathcal{P}$-language.

In this chapter we briefly comment about an extension of $\mathcal{P}$-language, called *Pertinence Action Language* (PAL[1]) [87] (first presented in [21]) which is thought to overcome these limitations. The goal of defining this language is to simplify the representation as much as possible by making use of the functional nature of symbols (either actions or fluents). In this way, we may use any action or fluent name inside logical or arithmetic expressions, or even as parameters of other actions or fluents. The syntax style has a strong resemblance to traditional programming languages like C or Pascal.

9.1 Missionaries and cannibals in PAL

Perhaps the best way to present PAL is directly using an example. Figure 9.1 shows the representation of the missionaries and cannibals problem in PAL.

As we can see, the first group of lines are used to define the set of blocks as the integer interval from 1 to 4, and the set of locations as any block or the table. A set is usually defined as a group of elements embraced by {...} , like for instance the singleton set {`table`}. Elements in a set can be both atom names (an identifier with a lower-case initial, like `table`) or integer numbers. As usual, the interval notation `[1,4]` is an abbreviation of {`1,2,3,4`}. Set expressions can be constructed using binary operators `+,-,*` standing for union, difference and intersection respectively. Two set names are predefined:

[1]Unfortunately, the same acronym has also been recently used in [13] to stand for *Probabilistic Action Language* which has no relation to the current work.

```
options
  not concurrent;

constants
  capacity = 2;
  nummis = 3;
  numcan = 3;

sets
  persontype = {mis,can};
  number = [0,nummis] + [0,numcan];
  bank = {left,right};

fluents
  num: persontype x bank -> number;
  moved: persontype -> [0,capacity];
  boatbank: bank;
  opposite: bank -> bank;

actions
  carry: [0,capacity] x [0,capacity] -> event;

vars
  M,C : [0,capacity];
  B : bank;
  P : persontype;

rules
  false if carry(M,C) and M+C>capacity;
  false if carry(0,0);

  moved(can):=C if carry(M,C);
  moved(mis):=M if carry(M,C);

  num(P,prev(boatbank)):=prev(num(P,prev(boatbank)))-moved(P);
  num(P,boatbank):=prev(num(P,boatbank))+moved(P);

  boatbank:=opposite(prev(boatbank)) if carry(M,C);

  false if num(mis,B)>0 and num(can,B)>num(mis,B);

initially
  opposite(left):=right,
  opposite(right):=left,
  num(P,left):=3,
  num(P,right):=0,
  boatbank:=left;
```

Figure 9.1: Missionaries and cannibals problem in PAL.

```
boolean = {true,false};
event = {true};
```

After these auxiliary declarations, we describe the `fluents` and `actions` involved in the domain. Syntactically, their declarations have the same shape, following the usual description of function types:

$$<fname> \ \texttt{:} \quad <set_1> \ \texttt{x} \ <set_2> \ \texttt{x} \ \ldots \ \texttt{x} \ <set_n> \ \texttt{->} \ <range>$$

which means that $fname$ is a n-ary function so that given $x_1, \ldots, x_n$ with $x_i \in set_i$, we have that $fname(x_1, \ldots, x_n) \in range$. Using this convention, the fluents in figure 9.1 are quite self-explanatory. When the fluent is 0-ary, we may simply omit the arrow `->`, as in the declaration of `boatbank`.

As a remarkable observation, notice how the range for action `carry` is `event`, that is, its only defined value is '`true`'. This is because, as we explained in section 4.1, unperformed actions have no defined value. In this way, we are free to define an action like:

```
set_bit : boolean;
```

which can be performed with value `true`, performed with value `false` or not performed at all. The *options* line expresses that we will not consider concurrent actions.

In order to define the rules, we declare auxiliary variables (always varying in a finite range). The rule syntax allows arithmetic, relational and logical operations, as in any programming language, plus a conditional '`if`'. As we can see, the expressions may freely mix variable names with fluent or action names. Besides, instead of handling a precondition, like the '**after**' in $\mathcal{P}$-rules, we include a fluent name "modifier" so that, for any fluent name `f`, the expression `prev(f)` denotes its value at the previous situation. This allows mixing previous values with current ones, as happens in the rules for computing the fluent `num`:

```
num(P,prev(boatbank))  :=  prev(num(P,prev(boatbank)))-moved(P);     (9.9)
      num(P,boatbank)  :=  prev(num(P,boatbank))+moved(P);           (9.10)
```

Notice that this integration even allows fluent expressions as parameters for non-0-ary fluents. In this way, using a compact representation like `num(P,prev(boatbank))`, we may easily express an apparently complex concept like:

> "the (resulting) number of persons of type `P` in the bank where the boat was previously."

The advantage of this representation becomes evident if we compare (9.9) and (9.10) to the previous seen $\mathcal{P}$-rules, (9.5) and (9.6), respectively. Note how we have reduced the amount of auxiliary variables, leaving just the one actually needed: the type of person `P` for which we wish to update the number. Of course, it is still possible to maintain these auxiliary variables, handling a closer notation to $\mathcal{P}$-language, like for instance, with the equivalent rule:

```
num(P,B):=N-M if moved(P)=M and prev(num(P,B))=N and prev(boatbank)=B;
```

Let us see next other interesting features with a second typical example.

9.2 The blocks world

Another typical scenario for planning problems is the following one:

Example 20 (The blocks world)
A robot may move a set of cubic blocks which can be placed one on top of another or directly on the table (we assume we always have enough room on the table). A block cannot be moved if it has something on top. Besides, only one block can be moved at a time. □

Figure 9.2 shows the representation of the blocks world scenario in PAL. This example introduces a new modifier `pert` which refers to the pertinence of a given symbol. In this way, `pert(move(B))`, is used to test whether the action `move(B)` is pertinent (i.e., has been performed) or not, regardless the action value. Of course, we could also represent this same concept by using an alternative block variable `C`, and requiring `move(B)=C`, but this additional variable would not have any particular use (it would actually correspond to the use of the underscore '_ ' in Prolog) and would unnecessarily interfere in the grounding process. Besides, the reference to pertinence of an action is *essential* for expressing that the action *has not* occurred. For instance, in order to express that we have not moved block `B`, the expression `not (moved(B)=C)` cannot be used as a condition, since both `B` and `C` are variables universally quantified in the whole rule, and so, existentially quantified inside the condition. Furthermore, the mere reference to an action name (without using `pert`) implicitly leads to assume that this action is pertinent. So, the correct way to represent the non-occurrence of `move(B)` would be `not pert(move(B))`. AS for the case of fluents, we can also use `pert` to refer to their pertinence/non-pertinence, but we do not have this time the problem of undefined values.

Another new feature is the use of fluent literals to the left of '`if`'. For instance, the rules for fluent `clear` (marked as `r2` and `r3`) would respectively be simple abbreviations of:

```
clear(move(B)):=false;
clear(prev(loc(B))):=true if pert(move(B));
```

Notice again how we exploit the functional nature of actions and fluents as much as possible. For instance, rule `r3` expresses in a compact way the sentence:

> "if we move some block `B`, then the previous location of `B` becomes clear."

As another piece of example, rule `r5` qualifies action `move(B)`:

> "it is not possible that the block where we move `B` was not previously clear."

Some implicit assumptions have also been taken into account. For instance, consider the expression `clear(move(B))`. As `move(B)` is a location, it may include the case of `move(B)=table`. However, `clear(table)` is not defined, because `clear` is only applicable to a block. When the grounding leads to a case like this, no grounded $\mathcal{P}$-rule is generated at all.

As happened in the previous example, PAL rules are simpler than the usual representation of this domain in logic programming or in other action approaches. For instance, we typically would have to include a set of constraints to express that we cannot place the same block at two different locations or that we cannot move the same block to different targets. These constraints become implicit here because `move` and `loc` are defined as functions (whose result value is always unique by definition). Of course, this feature is not only provided by PAL, being also present in other action formalisms that allow the definition of multi-valued fluents.

```
options
  not concurrent;

sets
  block = [1,4];
  location = block + {table};

actions
  move: block -> location;

fluents
  loc: block -> location;
  clear: block -> boolean;

vars
  B: block;

rules
  loc(B):=move(B);                                  /* r1 */
  not clear(move(B));                               /* r2 */
  clear(prev(loc(B))) if pert(move(B));             /* r3 */
  false if pert(move(B)) and not prev(clear(B)); /* r4 */
  false if not prev(clear(move(B)));                /* r5 */
  false if move(B)=B;                               /* r6 */
initially
  loc(B):=table,clear(B);
```

Figure 9.2: Blocks world scenario in PAL.

For instance, in the very recent work [45], multi-valued fluents are incorporated into the $\mathcal{C}$ action language. The resulting extension has received the name of $\mathcal{C}+$ and has also been embodied into CCALC tool. In $\mathcal{C}+$, for each fluent f, we have a set of values $Dom(f)$ (which corresponds to our $range(f)$) so that we may use a new type of atom, $f = v$, inside the causal expressions. Using this feature, they present a very similar representation to our blocks world scenario where, for instance, the location l of a block b is also represented as $loc(b) = l$. It is worth to note, however, that despite of this similarity, $\mathcal{C}+$ still forces to use auxiliary variables, rather than directly constructing expressions with the fluent name. To put an example, the rule for increasing the number of cannibals we saw in the previous example, would be represented in $\mathcal{C}+$ as:

$$\textbf{caused}\ num(can, B) = N - M \quad \textbf{if} \quad moved(can) = M\ \textbf{after}\ num(can, B) = N \wedge boatbank = B;$$

and, using CCALC syntax, finally written as:

```
caused num(can,B) eq X if moved(can) eq M
                after num(can,B) eq N && boatbank eq B && X=N-M
```

Anyway, it is easy to see that moving $\mathcal{C}+$ towards PAL representation could be directly proposed without any additional difficulty.

9.3 Rule grounding

Despite of the different looking between a PAL rule and a $\mathcal{P}$-language rule, the way in which the former is interpreted is, in fact, by translation into a set of grounded instances of the latter. This translation follows the next steps:

1. First, we identify all the components to be instantiated, that is, variables and symbol names, understanding as different components `p`, `prev(p)` and `pert(p)`, for each symbol `p`. Then, we generate a grounded instance per each possible combination of these components. As an exception, we do not instantiate the leftmost fluent occurring at the left of `:=`, let us call it the *effect fluent*. We denote p_i/v_i to stand for the instantiation of symbol p_i into one of its values $v_i \in range(p_i)$.

2. Once the combination of values is established, we simply make a bottom-up evaluation of all the expressions. Whenever an `if` condition is valuated as `false` or a parameter is out of its symbol domain, we jump to the next instance without generating any grounded rule.

3. Finally, we generate a (grounded) $\mathcal{P}$-rule:

$$E\ \textbf{if}\ holds(p_1, v_1) \wedge \ldots holds(p_n, v_n)\ \textbf{after}\ holds(f_1, u_1) \wedge \cdots \wedge holds(f_m, u_m)$$

   where:

   i) E is the atom $holds(f, v)$ where f is the effect fluent, and v is the final value of the expression to the right of `:=`. If the PAL rule had not effect fluent (i.e., its head was `false`) or $v \notin range(f)$ then E becomes $\bot$ instead.

   ii) each $holds(p_i, v_i)$ corresponds to the instantiation p_i/v_i for each symbol p_i which was not referred via `prev` modifier.

iii) each $holds(f_i, u_i)$ corresponds to the instantiation $prev(f_i)/u_i$ for each fluent f_i which was referred using the `prev` modifier.

Let us see some examples. Consider the rule:

```
false if carry(M,C) and M+C>2;
```

We must instantiate the values for `M` $\in [0,3]$, `C` $\in [0,3]$ and `carry(M,C)` $\in \{true\}$. When an instantiation like (`M/2`, `C/1`, `carry(M,C)/true`) makes the condition become `false`:

```
true and 2+1>2
```

no rule is generated. Thus, the final set of rules would be:

$$
\begin{array}{lll}
\perp & \textbf{if} & holds(carry(3,3),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(3,2),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(3,1),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(3,0),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(2,3),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(2,2),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(2,1),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(1,3),\mathtt{t}) \\
\perp & \textbf{if} & holds(carry(1,2),\mathtt{t})
\end{array}
$$

Notice that, unless the symbol parameters are constant (or the symbol is 0-ary), the final complete name of the fluent is not established until we make the bottom-up valuation (step 2). For instance, we first detect in step 1 some `carry(M,C)` or some `num(P,boatbank)`, using their respective ranges to generate grounded instances, but we do not finally obtain a grounded symbol, e.g. like `carry(2,1)` or `num(mis,left)`, until we do the bottom-up valuation of the grounded expressions. When doing this, it may be the case in which the parameter expression is out of domain, like in:

```
clear(move(B)):=false;
```

when `move(B)=table`. Again, as stated in step 2, no rule is generated in such a case. In this way, we would obtain:

$$
\begin{array}{lll}
holds(clear(1),\mathtt{f}) & \textbf{if} & holds(move(1),1) \\
holds(clear(2),\mathtt{f}) & \textbf{if} & holds(move(1),2) \\
holds(clear(3),\mathtt{f}) & \textbf{if} & holds(move(1),3) \\
holds(clear(4),\mathtt{f}) & \textbf{if} & holds(move(1),4) \\
 & \vdots &
\end{array}
$$

where `table` never occurs.

As an additional interesting example of grounding, consider the domain:

```
sets
  int = [0,4];

actions
  i : int;
fluents
  b: [1,3] -> int;

rules
  b(i):=prev(b(i))+1;
```

Intuitively, when we assign a value `k` for the action `i`, the `k`-th position of `b` is incremented by one. Of course, we implicitly require that $\mathtt{i} \in [1,3]$ since, otherwise `prev(b(i))` is not defined, and so, no rule is generated. Besides, for any instance with `prev(b(i))=4` the expression `prev(b(i))+1` is valuated as `5`, and so, it is out of range for the effect fluent `b(i)`. Therefore, the PAL interpreter automatically generates the constraints:

$$\begin{array}{lll} \bot & \textbf{if} & holds(i,1) \ \textbf{after} \ holds(b(1),4) \\ \bot & \textbf{if} & holds(i,2) \ \textbf{after} \ holds(b(2),4) \\ \bot & \textbf{if} & holds(i,2) \ \textbf{after} \ holds(b(2),4) \end{array}$$

All these implicit transformations allow avoiding range and domain checkings like:

```
b(i):=prev(b(i))+1 if i>=1 and i<=3;
false if prev(b(i))=4;
```

It is also interesting to note that the resulting interpretation for boolean expressions corresponds to the definition of pertinence for propositional connectives. For instance, consider the case of a disjunction like:

```
fluents
  b,c,d:boolean;
rules
  b if c or d;
```

The grounding would generate four possible pairs of values for `c` and `d`, but only three of them satisfy the condition `c or d` (the case `c=false`, `d=false` is ruled out). So, the resulting rules would be:

$$\begin{array}{lll} holds(b,\mathtt{t}) & \textbf{if} & holds(c,\mathtt{t}) \wedge holds(d,\mathtt{f}) \\ holds(b,\mathtt{t}) & \textbf{if} & holds(c,\mathtt{t}) \wedge holds(d,\mathtt{t}) \\ holds(b,\mathtt{t}) & \textbf{if} & holds(c,\mathtt{f}) \wedge holds(d,\mathtt{t}) \end{array}$$

The generation of the three cases is important, since $\mathcal{P}$-rules are then interpreted using pertinence. For instance, it must be observed that we cannot translate the PAL rule by just handling:

$$\begin{array}{lll} holds(b,\mathtt{t}) & \textbf{if} & holds(c,\mathtt{t}) \\ holds(b,\mathtt{t}) & \textbf{if} & holds(d,\mathtt{t}) \end{array}$$

because, as we saw when studying disjunction in static L^2, the formula $\mathtt{b} \vee \mathtt{c}$ can be also caused when one of its disjuncts is true and not pertinent while the other is false but pertinent.

9.4 Temporal projection

Apart from the scenario description, PAL allows solving, in its current version, two kinds of reasoning problems: *temporal projection* and *temporal queries*. These two types of problems are integrated in the same framework, so that, we can simultaneously mix simulation, which goes updating a "real" narrative, and solving hypothetical queries, where the initial state is assumed to be the current one in the real narrative. Let us see first how temporal projection works.

To see a well known example, consider the suitcase scenario:

```
sets
  lock={1,2};
fluents
  open: boolean;
  up: lock -> boolean;
actions
  toggle: lock -> event;
vars
  L:lock;
rules
  up(L):=not prev(up(L)) if toggle(L);
  open if up(1) and up(2);
```

A typical temporal projection description would have the shape:

```
initially
  not up(1),up(2),not open;

do {
  toggle(2);
  toggle(1);
  toggle(1),toggle(2);
  ;
  toggle(1);
}
```

The `initially` clause is followed by a set of fluent value assignments (in this case represented as literals) which resets the real narrative to the corresponding initial state. In fact, we can use variables and conditional expressions for these assignments, provided that no reference to other fluent is done. For example, this expression would also be correct:

```
up(L):=false if L!=2
```

inside `initially`.

After initializing the real narrative, the `do` clause describes the sequence of action assignments, using a semicolon to delimit the end of a transition, and a comma to separate concurrent actions inside the same transition. To put an example, the output of the previous execution would be:

```
1)
toggle(2):=true
```

```
up(2):=false
2)
toggle(1):=true
up(1):=true
3)
toggle(1):=true
toggle(2):=true
up(1):=false
up(2):=true
4)
5)
toggle(1):=true
open:=true
up(1):=true
```

As it could be expected, only the pertinent facts are shown (the rest can be deduced by persistence). Note how from a programmer's point of view, pertinence means an advantage, since it emphasizes the relevant part of the domain. This is very important, specially when dealing with a relatively large amount of fluents. In fact, this necessity of limiting the output to the relevant information has also arisen when using other systems. For example, when using the logic programming tool SMODELS [105] to represent action domains, we typically include some assertions of the style:

```
hide.
show caused(F,V,S).
```

to express that, when showing a stable model, we want to hide all the atoms excepting the ones for predicate `caused`, which are the actually relevant ones. The advantage of pertinence is that this relevance is *context-dependent*, so that a fact for a given fluent may be relevant in a given transition but non-relevant in a different one.

The changes done when using `do` clauses are incremental, so that:

```
do {toggle(1);}
do {toggle(2);}
```

is equivalent to:

```
do {toggle(1);toggle(2);}
```

The current "real" state is, at each moment, the one corresponding to the last `do`-transition. In order to compute the successor state, PAL uses by default the well-founded encoding for $\mathcal{P}$-rules. In this way, there exists at most a unique successor state, although this state may contain some undefined fluents (or just some of their values). For instance, consider the already seen cycle example R_2 (rule (7.6) in section 7.1) which corresponds to:

```
actions
  a: event;
fluents
  b: boolean;
rules
  b if b and a;
```

The output for the following temporal projection:

```
initially  b;
do {a;}
```

is the resulting state:

```
1)
a:=true
b:=?   Unfounded values={ false }
```

which points out that, at situation 1, `a` is (trivially) `true` and pertinent whereas the pertinence of `b` could not be established, although we know for sure that `b` cannot be `false`, (i.e., the value `false` is "unfounded" for `b`).

Similarly, the example R_4 in section 7.1, containing rules (7.9)-(7.10), would correspond to the PAL description:

```
actions
  a: event;
fluents
  b,c: boolean;
rules
  b if not c and a;
  c if not b and a;
```

The execution of:

```
initially
  not b, not c;
do { a; }
```

yields the following output:

```
1)
a:=true
b:=?   Unfounded values={ }
c:=?   Unfounded values={ }
```

that is, both `b` and `c` are completely undefined.

Apart from the well founded encoding, PAL currently allows switching to the stable models interpretation (circumscription and completion are still under development) using the tool SMODELS[2] as a back-end search engine for obtaining the stable models.

For changing the interpretation of rules, we can include the option:

```
options
  inference=smodels;
```

Using this option, the output for R_2 becomes instead:

```
No models found.
```

[2]This connection between PAL and SMODELS is relatively simple thanks to the availability of an API for handling SMODELS from any C++ program.

whereas the output for R_4 is:

```
Several models were found.
1)
a:=true
b:=true
```

Two important remarks must be made here. First, as the stable models interpretation leads to a nondeterministic transition relation, temporal projection *arbitrarily selects one* of the possible successor states as the resulting real state. In the case of R_4, we have obtained the model where `b` becomes true and pertinent while `c` persists false (and so, it is not shown). Second, the case in which "`no models are found`," (there is no stable model for the corresponding program) never happens when using well-founded semantics. At most, we can get that there does not exist a successor state, but this is always due to the application of a rule with effect $\perp$. As a result, it is always possible to *show one rule* (at least) that has caused the inconsistence. For instance, the execution for the blocks scenario:

```
initially
  loc(1):=2,
  loc(B):=table if not B=1,
  not clear(2),
  clear(B) if not B=2;
do {
  move(2):=table;
}
```

using the well founded interpretation, leads to the output:

```
Inconsistence (rule 4, line 23, applied with 'false' head).
```

that is, the rule we have marked as `r4` has been applied (we cannot move a block that was not clear).

9.5 Queries

As we said above, together with the solution of temporal projection or simulation problems, PAL also allows solving temporal queries. These queries can be done after or before any `do` clause and never affect to the real narrative.

The simplest case of query is the one for consulting information about the current (or previous) state. For instance, coming back to the suitcase scenario, the following queries:

```
query
  open ?
  up(1) or prev(up(2)) ?
```

are quite straightforward, respectively asking, in the first case, whether the suitcase is open, and in the second case, if it is the case that lock 1 is up, lock 2 was up or both. In fact, a query accepts practically the same syntax than any rule condition, and it follows similar grounding steps, generating grounded query instances. So, we can ask about pertinence, like in:

```
query
  pert(up(1)) and pert(up(2)) ?
```

to check whether both fluents have been affected by the actions, we can use variables, like in:

```
query
  up(L) ?
```

to check which locks are currently up, or we can construct expressions with the fluent and action names, like in:

```
query
  num(mis,left)=num(can,right)+2 ?
```

The answer to a query containing variables usually leads to multiple solutions, and the output is similar to the one for a Prolog query:

```
vars
  B,C:block;
query
  loc(B)=table and loc(C)=table and B>C?

B=2,C=1
B=3,C=1
B=3,C=2
B=4,C=1
B=4,C=2
B=4,C=3

6 solutions
```

We can also fix the maximum number of solutions using the option:

```
options
  solutions=3;
```

which is usually more interesting for temporal queries.

Apart from the queries about the current state, we can propose hypothetical future observations so that PAL tries to find an explanation, guessing the actions to be performed. These future observations are placed in the query by separating them with semicolons. For instance, assume that the initial state of the suitcase scenario is:

```
initially
  not up(1), not up(2), not open;
```

If we perform the query:

```
query
  true; open?
```

we are asking for a way of making `open` true in a hypothetical next state. The output in this case is:

```
Solution 1:
1)
toggle(1):=true
toggle(2):=true
open:=true
up(1):=true
up(2):=true

1 solution
```

Of course, the semicolon can be used as many times as desired, like:

```
query
  true ; true ; true ; open ?
  true ; true ; true ; up(1) ; up(1) ; up(1); open ?
```

Besides, when the formula in a given situation is simply the expression '`true`', it can be omitted. For instance, the first queries above, can also be represented as:

```
query
  ; ; ; open ?
  ; ; ; up(1) ; up(1) ; up(1); open ?
```

As another abbreviation, when we want to repeat the last formula ϕ along the next `k` situations we can use the notation:

$$\phi ... \{\text{k}\} \psi$$

to replace the expression:

$$\overbrace{\phi;\ \ldots;\phi;}^{\texttt{k}}\ \phi \text{ and } \psi$$

In this way, the example queries can be further simplified into:

```
query
  ...{3} open ?
  ...{2} ; up(1) ...{2} ; open ?
```

As an example of the typical use of this syntax for solving planning problems, the query for solving the missionaries and cannibals problem in 11 steps would have the shape:

```
query
  ...{11} num(can,right)=3 and num(mis,right)=3 ?
```

9.6 Queries for stable models: temporal explanation vs. planning

It must be emphasized that PAL queries are actually solving *temporal explanation* problems, which is not strictly the same than planning problems. As we already discussed in section 4.2, when the transition relation is deterministic, like in the well-founded encoding of $\mathcal{P}$-rules, or simply, when cycles are not present in any of the encodings, there is no difference between both types of problems. However, under a nondeterministic framework, temporal explanation is weaker than planning. Thus, when we deal with cycles and we use the stable models option, the result of a query may not be a valid plan.

For instance, consider the already seen cycle R_4 using the `smodels` inference option, and assume we pose the query:

```
initially
  not b, not c;
query
  ; c ?
```

The answer has one solution:

```
Solution 1:
1)
a:=true
c:=true

1 solution
```

which provides the only possible *explanation* (performing the action `a`) that justifies the observation `c` true at the next situation. However, as the execution of `a` is nondeterministic, this solution *does not guarantee* achieving `c` true in the next state. In fact, as we saw before, if we perform:

```
do { a; }
```

the temporal projection selects (arbitrarily) the other possible stable model in which `b` becomes true and pertinent and `c` persists false. So, the goal `c` is not satisfied.

In order to establish whether a given explanation is a valid plan, we must perform an additional query where, together with the obtained actions sequence[3], we negate the goal in the last situation. If no solution is found, then the explanation is a valid plan. In our example, this simply corresponds to checking whether the query:

```
query
  ; a and not c?
```

has solutions or not. As we obtain the possible model:

```
Solution 1:
1)
a:=true
b:=true

1 solution
```

[3] We are assuming nonconcurrent actions. Otherwise, we should also avoid that the answer includes additional actions as new explanations.

we can assert that performing `a` is not a valid plan for obtaining `c`. In fact, as this was the only explanation, this means that such a valid plan does not exist.

This technique for checking the validity of a given explanation as a plan is also used in CCALC, where the transition relation is (in the general case) nondeterministic. As we saw, when causal rules contain positive cycles, their encoding into Clark's completion may lead to multiple successor states for a given transition. In the case of CCALC, as the completion is a classical theory, searching a plan corresponds to two queries:

i) Find a model M such that $M \models T \wedge G$, where T is the (completed) background theory and G is the goal

ii) Check whether $T \wedge A \wedge \neg G \models \bot$, being A the sequence of actions occurring in M.

that are usually solved by an external propositional solver[4].

In classical logic, step ii) is equivalent to show $T \wedge A \models G$. However, when we use stable models, this equivalence is not valid any more, because the entailment relation is now nonmonotonic (and even noncumulative). Thus, the soundness of this technique depends on the shape of the program. This is guaranteed by the following two features:

- Each grounded query like G or $A \wedge \neg G$ is actually represented as a logic program constraint instead of being included as a set of facts in the program. In other words, we do not add the query as evidence for the program but, instead, we *select* the resulting stable models that contain the query facts.

- It is easy to see that any stable model for the program $P(R)$, translation of a given set of rules R, always provides at least one value for each action or fluent at each situation. Thus, selecting the stable models that do not contain $holds(p, \mathtt{t})$ is the same than selecting the stable models that contain $holds(p, \mathtt{f})$.

[4]One of the problems of this technique is, in fact, that most of the available efficient propositional solvers provide at most one model M for step i) but it may be the case in which M does not pass the test in step ii).

Chapter 10

Conclusions

In this dissertation we have studied the use of causality in action domains, but applying a different focusing with respect to the recent line of causal approaches that have emerged in the area. These approaches understand causality as some mathematical mechanism to rule out undesired models. They are mostly concerned with solving representational problems (like the frame and ramification problems) and so, in many cases, once the desired models are obtained, causal information is disregarded. The work presented here, has studied instead the possibility of dealing with causal knowledge as a significant part of the state, at the same level than fluent values. In this way, we can establish an excluding distinction between facts that have been caused, that is, that have been obtained via some causal intervention, and facts that have persisted, i.e., that have been obtained by inertia. Furthermore, this information can be used inside the conditions of causal rules, exactly as we would do with fluent values.

The separation between inertia and causality has been achieved by defining the concept of *pertinence* with respect to the actions execution and proposing some postulates about its basic features. Essentially, we have defined that a fluent is pertinent whenever its value is result of a causal intervention, regardless its final value. Analogously, the fluent is nonpertinent when it persists by inertia. In fact, this idea is generalized so that we can talk about pertinence/persistence of any formula ϕ (it is pertinent if and only if one of the occurring fluents is pertinent).

Starting from this idea, the main contributions of this thesis can be summarized as follows:

- We have presented a framework for transition systems based on finite state machines, where pertinence is understood as an output function associated to the information included in the state.
- We have introduced a basic causal rule syntax, we called $\mathcal{P}$-language (for analogy with $\mathcal{A}$-language), that allows describing the transition function in a more compact and elaboration tolerant way and provides a representation for the causal dependences underlying in the domain.
- As a first attempt of semantics, we have provided a natural and simple update algorithm that allows computing the succesor state for any possible transition, assuming that causal dependences are acyclic.
- We have presented a whole first order logic formalization for capturing the operational behavior of $\mathcal{P}$-language. This formalization (called *Pertinence Calculus* for analogy with Situation or Event Calculi) incorporates basic axioms for two predicates: *holds* for specifying the fluent values, and *pert* for representing their pertinence.

- As Pertinence Calculus relies on a classical logic representation, we have provided different nonmonotonic mechanism for dealing with inertia and for avoiding the ramification problem. The alternatives we have presented are:

  1. *Circumscription*: we maintain the classical logic representation but minimize the *pert* predicate inside a portion of the theory
  2. Logic Programming (LP) under *stable models semantics*
  3. LP under *well-founded semantics*
  4. LP under Clark's completion

  Assuming acyclicity of causal rules, we have shown the correspondence of these four semantics with respect to the operational behavior.

- We have provided a detailed study on the effects of causal cycles when interpreted under the four nonmonotonic options presented before.

- We have made a thorough comparative with respect to the best known action approaches, particularly focusing on the way in which they represent causal knowledge versus inertia.

- The theoretical study has been put in practice by implementing an interpreter of a high level language, *Pertinence Action Language* (PAL), which simplifies the representation of dynamic domains, thanks to the definition of pertinence, and to the possibility of constructing expressions that involve functional actions and fluents.

Besides these main contributions, some interesting collateral results have been obtained, like for instance, the variation of the transformation based algorithm for WFS presented in [17] to cope with WFSX [2].

The broadness of this research work gives chance to a great deal of improvements and open lines for future work. We can classify these open directions into extensions for the theoretical part and improvements for the practical system. For instance, as an action formalism, the presented work has not dealt with other common features which are typically studied in the area. These features are not essential for the study of causal reasoning, but are very interesting for an appropriated representation of many action scenarios. We may cite:

- (Explicit) *nondeterminism*: it consists in allowing the description of a nondeterministic transition relation using a causal representation. This could consist in incorporating a new type of rule whose effect is *possibly* caused, but its causation is not always granted. In this way, the application of the rule may lead to a state in which the effect is caused, or to a state in which the rule fails.

- *Qualification problem*: it is partially related to the previous topic. The goal is to allow describing in a compact way the exceptions for the effects of an action (or even a ramification rule), avoiding to mention them each time the action is referred along our domain representation.

- *Natural actions*: some caused effects do not have any simultaneous external intervention, but are instead the result of a delayed causal dependence. In order to represent these delayed effects, we have two possibilities: (1) including rules whose conditions do not depend on pertinence; and (2) studying the propagation of pertinence between consecutive situations (notice that, currently, the preconditions of rules do not depend on pertinence).

- *Actions with durations and numerical timeline*: sometimes, it is interesting to represent an action that takes place along a number of situations, or even a time interval. This could be easily done by translating this type of actions into events in a similar way as done in TAL [33] or Event Calculus [53].

- *Sensing and robot knowledge*: another interesting topic is the possibility of representing not only the world state, but also the robot's knowledge about that state. This allows proposing planning problems that may involve sensing actions for completing the robot's knowledge before making any modification of the real state.

Apart from these typical enhancements, we could also extend the three usual types of problems (temporal projection, temporal explanation and planning) for dealing with other interesting features like:

- *Strong equivalence* between causal representations: that is, not only to guarantee that they lead to the same transition relation, but also that this is satisfied even after adding new causal rules. In order to study this property, we can take benefit from recent work [64] about strong equivalence for logic programs under stable models. In the case of well-founded semantics, the strong equivalence has not been fully characterized yet although, for instance, the semantics presented in [20] allows establishing sufficient conditions.

- Study of *temporal properties*: the current version of PAL, and in fact, the existing nonmonotonic satisfiability planners like CCALC or those using SMODELS, are capable of proving temporal queries with a fixed number of situations. In CCALC and SMODELS, this is used for grounding all the rules at each possible situation. An interesting kind of problems are those that consist in proving whether the system will eventually reach a given state or not, regardless the elapsed number of situations. The problem here is that the number of situations can be as large as needed. The most promising type of formalism to be used in order to solve queries like these is Modal Temporal Logic, since there exist decidable tableaux methods[1] for solving queries with formulas like $\Box\phi$ (that is, ϕ will be true forever) or $\Diamond\phi$ (that is, ϕ will eventually become true). However, the combination of these techniques (thought for monotonic propositional modal logic) with some general nonmonotonic formalism still remains to be solved. As an approximation, we can cite the temporal logic programming paradigm TEMPLOG [1, 14] and its extension for dealing with default negation under stable models semantics (Temporal Answer Sets [19]).

- *Machine Learning*: in this area it should be quite straightforward to adapt the results obtained in [68], where a thorough study for applying machine learning to action domains was presented. As that work mainly relied on logic programming implementations, it can be expected that the modification of the proposed techniques for dealing with pertinence will be practically immediate, specially when using any of the logic programming semantics we presented in section 6.6.

There are plenty of improvements that can be done for the PAL interpreter, apart from those derived from the above extensions for the theoretical framework. Among other, we can mention:

- Inclusion of *quantifiers* in queries and rules: some queries like "are all the blocks on the table?" cannot be correctly expressed since query variables are always existentially quantified.

[1] For instance, see section 5.4 in [7]

- Allowing full *temporal explanation* problems, providing a partial description of the initial state and generating an explanation that completes this description.
- Implementing the other two nonmonotonic techniques, i.e., the *circumscriptive encoding* and the one based in *Clark's completion* (or even an adaptation of Thielscher's causal relationships).
- Extending the *set operators* for allowing intensive descriptions. For some problems, it is also interesting to allow *defining sets of tuples* instead of atomic elements.
- Adding *conditional* and *iterative* instructions to the temporal projection syntax. In this way, it would be possible to capture the whole behavior of a small conventional programming language. Something similar has been recently studied in [95].
- Extension of the temporal operators to include *temporal constraints*. This would allow proposing Temporal Constraint Networks inside a typical action domain description. The theoretical study for this option has been presented in [22].

Appendix A

Proofs of theorems

Proof of theorem 1

By definition of $\leq_F$, we must prove $U^+ \subseteq W^+$ and $U^- \subseteq W^-$. Consider first any $L \in U^+ = facts(P)$. The result of applying Γ will always contain the fact L. By definition, $W^+ = \Gamma\Gamma_s(W^+)$, and so $L \in W^+$. Consider now $L \in U^-$. By definition, either $L \notin heads(P)$ or $L \in \overline{facts(P)}$, i.e., $\overline{L} \in facts(P)$. On the one hand, if $L \notin heads(P)$, the result of applying both Γ and Γ_s cannot contain L. Then, $L \notin \Gamma_s(W^+)$, i.e. $L \in \mathcal{H} - \Gamma_s(W^+)$ which, by definition, is W^-. On the other hand, if $\overline{L} \in facts(P)$, as $facts(P) \subseteq W^+$, then $\overline{L} \in W^+$. But then, the modulo $P_s^{W^+}$ will not contain any rule with L as head (in the seminormal program P_s, these rules contains *not* $\overline{L}$ in their bodies). As a result, $L \notin \Gamma_s(W^+)$ which means that $L \in \mathcal{H} - \Gamma_s(W^+)$ i.e. $L \in W^-$. $\square$

Proof of lemma 1

$\xmapsto{\texttt{F}}$

As p is not head of any rule in P, the same applies for P_s, P' and P'_s. Thus, for any of these programs Q and for any interpretation M, when iterating $T_{Q^M} \uparrow (\emptyset)$, p is never obtained and so, any rule with p in the body is never used. Then, it can be deleted without varying the result. This directly implies that $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$, for any M and so, the proofs for (a) and (b) become trivial.

$\xmapsto{\texttt{L}}$

We will similarly show that, for any M and any $Q \in \{P, P_s, P', P'_s\}$: $p \notin T_{Q^M} \uparrow (\emptyset)$. As $p \notin \Gamma(\emptyset)$ we immediately get that p cannot belong to any application of $\Gamma(M)$, because the resulting modulo is a subset: $P^M \subseteq P^\emptyset$. Besides, as $P^\emptyset = P_s^\emptyset$, we have $p \notin \Gamma_s(\emptyset)$, and so p cannot belong to any $\Gamma_s(M)$ since again $P_s^M \subseteq P_s^\emptyset$. Finally, for Γ' and Γ'_s it suffices to see that $P' \subseteq P$ and $P'_s \subseteq P_s$. Then, the rules with p in the body are never used during the iteration $T_{Q^M} \uparrow (\emptyset)$ and so $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$ for any M, being the proofs for (a) and (b) directly trivial.

$\xmapsto{\texttt{P}}$

As proved in $\xmapsto{\texttt{F}}$, since p is not head, it cannot belong to any application of $\Gamma, \Gamma_s, \Gamma'$ or Γ'_s. Besides, for any interpretation M such that $p \notin M$, it is easy to see that $P^M = P'^M$ and

$P_s^M = P_s'^M$. Let us prove first (a). For any fixpoint $M = \Gamma\Gamma_s(M)$ we have that, as M is the result of applying Γ, $p \notin M$. But then, $\Gamma_s(M) = \Gamma'_s(M)$ (which is the first consequent of (a)), and in its turn, $p \notin \Gamma_s(M)$. Finally, this means that $M = \Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$, that is M is fixpoint of $\Gamma'\Gamma'_s$. The proof for (b) is completely analogous.

$\xmapsto{\mathtt{S}}$

Notice first that, for any M, $\Gamma(M) = \Gamma'(M)$ and $p \in \Gamma(M)$, because p is a fact in P, and so, it will always valuated as true when applying T_P (resp. $T_{P'}$). Second, we show now that for any M such that $p \in M$ and $\overline{p} \notin M$, $\Gamma_s(M) = \Gamma'_s(M)$. The fact p occurs as the seminormal rule $p \leftarrow \mathit{not}\ p$ both in P_s and in P'_s, but in P_s^M and $P_s'^M$, this rule will become again the original fact p. As a result, deleting p from the rules will not vary the final outcome, i.e., $\Gamma_s(M) = \Gamma'_s(M)$. Besides, in P_s^M and $P_s'^M$ all the rules for $\overline{p}$ will be deleted (they contain in their bodies *not* p). This means that, additionally, $\overline{p} \notin \Gamma_s(M)$.

Now, we prove (a): let $M = \Gamma\Gamma_s(M)$. Then, as M is the result of applying Γ then $p \in M$. But, at the same time, as Γ_s is defined for M, $\overline{p} \notin M$. Therefore, we can apply the previous results, $\Gamma_s(M) = \Gamma'_s(M)$ (which is the first consequent of (a)). Let us call J to $\Gamma_s(M)$. Then, as we had seen, $\Gamma(J) = \Gamma'(J)$, i.e., $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is again analogous.

$\xmapsto{\mathtt{N}}$

First, note that for any M with $p \in M$, $P^M = P'^M$ and $P_s^M = P_s'^M$. Then, for proving (a), let $M = \Gamma\Gamma_s(M)$. As before, since p is a fact in P and M is the result of applying Γ, we get $p \in M$, and $\overline{p} \notin M$ (otherwise $\Gamma_s(M)$ would not be defined). Therefore $P_s^M = P_s'^M$ and $\Gamma_s(M) = \Gamma'_s(M)$ (the first part of (a)). Now note that the fact p in P becomes the seminormal rule $p \leftarrow \mathit{not}\ \overline{p}$ in P_s. However, in the modulo P_s^M (which is equal to $P_s'^M$) this rule becomes again the fact p, because $\overline{p} \notin M$. This means that $p \in \Gamma_s(M)$, and so, $P^{\Gamma_s(M)} = P'^{\Gamma_s(M)}$. It follows that $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. As always, the proof for (b) is analogous.

$\xmapsto{\mathtt{C}}$

Again, for any M with $p \in M$, in the modulos P_s^M and $P_s'^M$ all the rules with $\overline{p}$ as head are deleted (as they are seminormal, they contain *not* p in the body). As a result, $\overline{p}$ is never added when iterating the direct consequences operator, and so $\Gamma_s(M) = \Gamma'_s(M)$. Now, consider the proof for (a). If $M = \Gamma\Gamma_s(M)$, we have $p \in M$ (because of p being a fact and M the result of Γ) and so, the previous result is applicable: $\Gamma_s(M) = \Gamma'_s(M)$, which is the first part of (a). Now, by definedness of $\Gamma_s(M)$, we get that $\overline{p} \notin M$. But this means that when iterating direct consequences on the program $P^{\Gamma_s(M)}$, the fact $\overline{p}$ is never reached. Therefore, the rules with $\overline{p}$ in the body are never used, and so, the program $P'^{\Gamma_s(M)}$ has the same least model: $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is completely analogous.

$\xmapsto{\mathtt{R}}$

First observe that, for any M with $\overline{p} \notin M$, $P^M = P'^M$ and $P_s^M = P_s'^M$, and so, $\Gamma(M) = \Gamma'(M)$ and $\Gamma_s(M) = \Gamma'_s(M)$. Now, if $M = \Gamma\Gamma_s(M)$ we have (as in the two previous proofs): $p \in M$ and $\overline{p} \in M$. Therefore, we immediately have $\Gamma_s(M) = \Gamma'_s(M)$ (the first part of (a)). Now, note that $\overline{p} \notin \Gamma_s(M)$, because all the rules with $\overline{p}$ as head contain *not* p in their bodies, and we had that $p \in M$. By our first observation, this means that the modulo for P and P' w.r.t. $\Gamma_s(M)$ is the same one: $\Gamma\Gamma_s(M) = \Gamma'\Gamma_s(M) = \Gamma'\Gamma'_s(M)$. The proof for (b) is completely analogous. $\square$

Proof of theorem 2

Simply note that the well founded model in WFSX is defined as the three-valued interpretation $W = (W^+, W^-)$ with $W^+ = lfp(\Gamma\Gamma_s)$ and $W^- = \Gamma_s(M^+)$. As we have proved in lemma reflem:wfsx1, any fixpoint of $\Gamma\Gamma_s$ is fixpoint of $\Gamma'\Gamma'_s$ and vice versa. So $W^+ = lfp(\Gamma'\Gamma'_s)$. Besides, as also proved in lemma 1, for any fixpoint M, $\Gamma_s(M) = \Gamma'_s(M)$. So $W^- = \Gamma'_s(W^+)$. Therefore, if W is WFM of P, it is WFM of P'. Finally, if P has no WFM, as the fixpoints for $\Gamma\Gamma_s$ and $\Gamma'\Gamma'_s$ coincide, then P' has no WFM. □

Proof of theorem 3

It follows from the previous results. Let us consider (i) first. It is easy to see that any program P' containing the facts p and $\overline{p}$ is contradictory (has no fixpoints) in WFSX. As the transformations for WFS are also sound in WFSX, whenever we get the facts p and $\overline{p}$ in some of the transformed programs P', its WFM in WFSX is not defined, and so, the WFM for the original program is not defined as well. To prove (ii), it suffices with additionally applying lemma 1 for the resulting program P' after exhaustively applying all the WFS transformations. The facts of P' (i.e. W^+) are included in X^+ whereas the "non-head" atoms (i.e. W^-) are included in X^-. So $W \leq_F X$ for that program, and also for the original one. □

Proof of theorem 4

We begin proving (ii). Consider $\Gamma_s(U^+)$, and more concretely, the modulo $P_s^{U^+}$. By nonapplicability of $\xmapsto{\mathtt{N}}$, program P cannot contain a rule with *not* p in the body, being p a fact of P. However, in P_s, any rule with $\overline{p}$ in the head contains *not* p in its body. So, $P_s^{U^+}$ is the result of deleting in P_s any rule whose head is in $\overline{facts(P)}$ plus the remaining default literals. As a first consequence $\Gamma_s(U^+) \cap \overline{facts(P)} = \emptyset$. But also, it is easy to see that $P_s^{U^+} \subseteq P^{\emptyset}$. By nonapplicability of $\xmapsto{\mathtt{L}}$, all the positive literals of P are included in $\Gamma(\emptyset)$ whereas by nonapplicability of $\xmapsto{\mathtt{C}}$, there is no positive literal of P in $\overline{facts(P)}$. As a result, all the rule bodies in $P^{\emptyset}$ are true w.r.t. $\Gamma(\emptyset)$ and so $\Gamma(\emptyset) = heads(P^{\emptyset}) = heads(P)$. Finally, since the rules $P^{\emptyset} - P_s^{U^+}$ are those with heads in $\overline{facts(P)}$ and these in their turn never occur in the bodies of P, we get that $\Gamma_s(U^+) = \Gamma(\emptyset) - \overline{facts(P)} = heads(P) - \overline{facts(P)}$. Then, it directly follows that $\mathcal{H} - \Gamma_s(U^+) = \mathcal{H} - (heads(P) - \overline{facts(P)}) = U^-$. Now, we proceed to prove (i). By lemma 1, the trivial interpretation (U^+, U^-) has less information than the WFM. So, $U^+ \subseteq lfp(\Gamma\Gamma_s)$ and it will suffice with showing that U^+ is simply a fixpoint: $\Gamma\Gamma_s(U^+) = U^+$. By (ii), $\Gamma\Gamma_s(U^+) = \Gamma(heads(P) - \overline{facts(P)})$. If we call $J = heads(P) - \overline{facts(P)}$, we want to establish the least model of P^J. By nonapplicability of $\xmapsto{\mathtt{P}}$ and $\xmapsto{\mathtt{R}}$, given any *not* p in P, $p \in heads(P) - \overline{facts(P)}$. So, all the rules with default literals are deleted in P^J. Now, by nonapplicability of $\xmapsto{\mathtt{S}}$ in P, any body atom P^J cannot belong to $facts(P) = facts(P^J)$. This means that, when computing $T_{P^J} \uparrow (\emptyset)$, rules with nonempty body are never used. In other words, the least model of P^J, is $facts(P') = facts(P)$. That is, $\Gamma(J) = \Gamma\Gamma_s(U^+) = facts(P) = U^+$. □

Proof of lemma 2

Showing that (6.9) implies (6.8) is straightforward by definition of '$\equiv$'.

To show that (6.8) implies (6.9), assume that the former is true but the latter false. This means that, for some fluent F, value V, and situation I, $holds(F,V,I) \wedge \neg holds(F,V,J)$ or $\neg holds(F,V,I) \wedge holds(F,V,J)$. For the first case, we have that $holds(F,V,I)$, together with (6.8), implies $holds(F,V,J)$ and so we reach an inconsistence. In the second case, from axiom (6.1) and $holds(F,V,J)$ we get that:

$$\forall V'. \left(V \neq V' \supset \neg holds(F,V',J)\right)$$

Now, applying modus tollens to each $\neg holds(F,V',J)$ and (6.8) we get that $\neg holds(F,V',I)$ for all the values different from V, but due to axiom (6.1), this means $holds(F,V,I)$ and we reach again a contradiction. $\square$

Proof of lemma 3

We will work with the grounded propositional version of $t(R)$. Given any set of models, consider a partition where we collect in each class all the models with the same extent of predicate *holds*. We will show that each class of models coincides both for CIRC$[t(R); pert]$ and PCOMP$[R]$, and so, they have the same total set of models. Let H be one of these extents for predicate *holds*. Then, we can define the modulo of any theory T with respect to H, T^H, by replacing each atom $holds(p,v,i)$ by its truth value with respect to H. For instance, the models of PCOMP$[R]$ for class H would correspond to the models of PCOMP$[R]^H$. On the other hand, the models of CIRC$[t(R); pert]$ for class H are simply the models of $t(R)^H$ with less extent for predicate *pert*. Now, notice that $t(R)^H$ has the shape of a *positive* logic program exclusively containing ground atoms of *pert*. Then, by property 10, its minimal models can be computed by the Clark's completion of the program which is easy to see that it is no other than PCOMP$[R]^H$. $\square$

Proof of theorem 7

As R is definite, the theory T is equivalent to:

$$\text{PCOMP}[t(R)] \cup AX \cup Obs \cup (\text{UFR})$$

Let us consider again the grounded propositional version of T. We prove first that if the operational semantics obtains a narrative ν, this narrative corresponds to the unique model M of T. To prove this, we proceed by induction both in the *layer* function j and in the situation index i, proving that the atoms for (i,j) in M are exactly the ones we obtained in ν.

i=0) Without regarding the layer j, given any symbol p, M must contain an atom $pert(p, \mathtt{n}, 0)$, due to axiom (6.6), and an atom $holds(p,v,0)$ included in the observations $atoms(\sigma_0) \subset T$. As actions which are not pertinent cannot have a value (axiom (6.5)), we get that there is no action a such that $holds(a,v,0) \in M$. Finally, no other atom for situation 0 can be added to M, since axioms (6.1)-(6.4) guarantee that the symbol value and its pertinence value are unique for each situation. So, the set of atoms at 0 in any model M are exactly the ones in the narrative ν: the observed $atoms(\sigma_0)$ and non-pertinence for all the symbols.

i-1) We assume proved up to situation $i-1$ and try to prove for situation i.

i, j=0) The first layer of symbols consists of the actions and those fluents not occurring as effects in any rule. From the definition of **do** , for any fact $holds(a,v) \in \alpha_i$ we get the atom $holds(a, v, i) \in M$ and, by axiom (6.5), also $pert(a, \mathtt{p}, i) \in M$. Besides, any nonperformed action b is forced to become non pertinent $pert(b, \mathtt{n}, i)$, and again due to (6.5), no $holds(a, v, i)$ atom can be included for them. As for the fluents, given any f in layer 0, it is easy to see that in PCOMP[R] we will obtain the formula:

$$pert(f, \mathtt{p}, i) \equiv \bot$$

in other words, $pert(f, \mathtt{n}, i)$ must be true. From this, we get that all the pertinence atoms for M in layer 0 correspond exactly to the π_i^0 established by the algorithmic approach. But, as all the fluents are non-pertinent, the frame axiom (UFR) will shift all their previous values to the current situation, obtaining an atom $holds(f, v, i)$ for each $holds(f, v, i-1) \in M$. Since the induction hypothesis is verified up to $i-1$, this means that the atoms $holds(f, v, i-1)$ correspond to σ_{i-1} in the narrative, and so, the atoms $holds(f, v, i)$ correspond to σ_i^0 as defined by the algorithm.

i,j-1) We assume proved up to layer $j-1$ and proceed to prove for layer j.

i,j) Let us consider any rule E **if** C **after** D with $layer(symb(E)) = j$. Thanks to induction hypothesis, it is easy to show that the rule is applicable in ν^{j-1} iff $M \models \mathbb{C}C_i \wedge D_{i-1}$, since this formula only depends on atoms up to situation i and layer $j-1$, and its interpretation exactly corresponds to the idea of rule applicability. As these rule conditions are satisfied by M, the rule translations $t(R)$ will force $\mathbb{C}E_I$ to be true, and so we obtain $holds(f, v, i)$ and $pert(f, \mathtt{p}, i)$ for any $E = holds(f, v, i)$ effect of one of the applicable rules. As a result, M must mandatorily contain the set of atoms corresponding to Out^j. When all the rule conditions for a fluent f are not applicable, we will equivalently obtain that the disjunction of their translations:

$$\bigvee_k \mathbb{C}C_I^k \wedge D_{I-1}^k$$

is false in M, and by pertinence completion, $pert(f, \mathtt{p}, i)$ must become false, i.e., $pert(f, \mathtt{n}, i)$ true. In this way, we get that the pertinence atoms in M are exactly those in π_i^j. Finally, the frame axiom (UFR) is applied to any nonpertinent fluent in layer j, exactly as we did for layer 0. Therefore, the rest of fluent values in M are taken from the previous situation, which by induction hypothesis corresponds to σ_{i-1}, and so, M also contains $Pers^j$. Since we have shown that M includes π_i^j and $Out^j \cup Pers^j$, it contains at least one $holds$ value and $pert$ value for any fluent in layer j, and so, no more atoms can be consistently included in M.

We have to prove now that whenever M is a model of T then it is also the narrative obtained by the algorithm. Let us assume the opposite, that is, either the algorithm obtains a different M' or no narrative at all. The first case is not possible, since we have just proved that any obtained narrative is the unique model of T. So, the algorithm does not obtain any narrative, i.e., at some point we apply two rules with different effects for the same fluent (or we have obtained some $\bot$ as effect of an applied rule). However, assume we proceed with the inductive proof we have just used up to the point in which the first two rules of this kind are applied. The proof fixes the part of M we have obtained so far and, as we saw, the applicability of rules means satisfaction of their conditions in T. Thus, both effects must also be included in M, but this is not possible due to axiom (6.1). Something similar happens for rules with $\bot$ head: no model M can satisfy the propositional formula $\bot$. $\square$

Proof of theorem 8

We will construct a level mapping, *level*, for all the atoms of $P(R)$. Since R is acyclic, we may use the *layer* function, and its maximum value $maxlayer = max\{layer(p) \mid p \in \mathcal{A} \cup \mathcal{F}\}$. To this aim, we begin defining a $level_i$ "local" to each situation $i \in [0, n]$ so that, for each action A:

$$\begin{aligned} level_i(holds(A, V, i) &\stackrel{\text{def}}{=} &= 0 \\ level_i(pert(A, \mathtt{p}, i) &\stackrel{\text{def}}{=} &= 1 \\ level_i(pert(A, \mathtt{n}, i) &\stackrel{\text{def}}{=} &= 2 \end{aligned}$$

and for each fluent F:

$$\begin{aligned} level_i(pert(F, \mathtt{p}, i) &\stackrel{\text{def}}{=} &= layer(F) * 3 \\ level_i(pert(F, \mathtt{n}, i) &\stackrel{\text{def}}{=} &= layer(F) * 3 + 1 \\ level_i(holds(F, V, i) &\stackrel{\text{def}}{=} &= layer(F) * 3 + 2 \end{aligned}$$

Then, the global *level* is simply defined as:

$$\begin{aligned} level(holds(P, V, i)) &\stackrel{\text{def}}{=} level_i(holds(P, V, i)) + i * (maxlayer + 1) \\ level(pert(P, V, i)) &\stackrel{\text{def}}{=} level_i(pert(P, V, i)) + i * (maxlayer + 1) \\ level(\bot) &\stackrel{\text{def}}{=} (n + 1) * (maxlayer + 1) \end{aligned}$$

It can be easily checked that no rule in $P(R)$ violates this level ordering. In the case of actions, we first establish their value $holds(A, V, i)$, then their positive pertinence $pert(A, \mathtt{p}, i)$, which depends on the value by (6.52)), and finally their negative pertinence $pert(A, \mathtt{n}, i)$ which depends on the positive one by (6.49). As for the fluents, from layer to layer, we also define three levels: first we decide the positive pertinence $pert(F, \mathtt{p}, i)$ since, looking at the $\mathcal{P}$-rule translations, it exclusively depends on symbols of lower layer; second, we include the negative pertinence $pert(F, \mathtt{n}, i)$ which depends on the positive one by (6.49); and finally, we include the fluent value which depends on the negative pertinence because of rule (6.48). □

Proof of theorem 9

We will use the transformation method from [17] presented in the background. Notice that, as the program is acyclic, it will suffice with applying rules $\stackrel{\mathtt{x}}{\longmapsto}$ with $\mathtt{x} \in$ {P,N,S,F} (positive reduction, negative reduction, success and failure), since rule $\stackrel{\mathtt{L}}{\longmapsto}$ is exclusively for positive loop detection. As in the proof for theorem 7, we will proceed by induction in the situation and the layer number. We will show that we can go obtaining an equivalent program so that the rules with head up to situation i and layer j are exclusively logic program facts, and that these facts correspond to the ones in the narrative already obtained by the algorithm.

i=0) It is easy to see that the program rule (6.49) adds the nonpertinence atoms for all the symbols whereas $atoms(\sigma_0)$ completely fix the initial state. The rules (6.1) and (6.3) can be deleted by failure (there are no simultaneous values for the same symbol in a state

σ_0). Also, rule (6.52) is deleted by failure, since no atom $holds(a, v, 0)$ is head of any rule. Finally, rule (6.49) with $I = 0$ is deleted by positive reduction, since no atom $pert(p, \mathtt{p}, 0)$ is head of any program rule. So, we can obtain an equivalent program that exclusively contains the atoms of the narrative as only rules with head for situation 0.

i-1) We assume proved up to situation $i - 1$ and try to prove for situation i.

i, j=0) All the action atoms are directly obtained from **do** (α_i, i). For those performed actions, we also get $pert(a, \mathtt{p}, i)$ by applying success in (6.5). The rest of instances for (6.5) are deleted by failure (there is no $holds(a, v, i)$ for those actions). As for the fluents, note that there cannot be any program rule with head $pert(f, \mathtt{p}, i)$ and fluent f at layer 0. Thus, we can apply positive reduction in (6.49) for all the fluents at layer 0, obtaining that they are nonpertinent. This also means that we can apply success to all the instances of (6.48) for which the previous value of the fluent has been already obtained, and so we obtain that the fluent value persists. The rest of instances of (6.48) are deleted by failure. So, again we reduce the program rules at this layer exactly to the set of atoms obtained by the algorithm.

i,j-1) We assume proved up to layer $j - 1$ and proceed to prove for layer j.

i,j) Let us consider any rule E **if** C **after** D with $layer(symb(E)) = j$. Thanks to induction hypothesis, it is easy to show that the rule is applicable at this layer iff some of the program rules for its translation can be reduced by success, obtaining the value and pertinence atoms for the $\mathcal{P}$-rule effect. The rest of program rules in the translation can be deleted by failure. As for non-applicable $\mathcal{P}$-rules, it is also easy to see that they can be deleted by failure. Then, the instances of (6.49) for the obtained pertinent fluents can be deleted by negative reduction, whereas for those fluents that have not been concluded as pertinent, we can apply positive reduction to (6.49) to obtain that the are nonpertinent. Finally, we can apply success to (6.48) to obtain the persistence of their value, and failure for the rest of instances of (6.48).

It must be observed that when the algorithm stops because of inconsistence, it could actually go on computing facts until reaching the last situation and layer. It is easy to see that the algorithm detects inconsistence if and only if the program transformation method yields atom $\perp$ as a rule fact. Finally, as we have shown that the final program exclusively contains program facts, this means that we have obtained the complete WFM (otherwise, we could only guarantee that the WFM is a superset of these facts). □

Proof of theorem 10

Stable models:
It is clear that, given some σ_0 and $\vec{\alpha}$, the stable model M of $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$ is a stable model of $P' = P(R) \cup P_{gen}$. To check it, just note that P^M and P'^M are the same programs: when doing the modulo w.r.t. M, the rules in P_{gen} generate exactly the observations in $atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$.

So, we must actually prove that for any stable model M of $P' = P(R) \cup P_{gen}$ there exists some σ_0 and $\vec{\alpha}$ such that M is stable model of $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$. We prove first that M contains a unique atom $holds(f, v, 0)$ for each fluent f. Assume it contains no value for that fluent. Then in P'^M, the rule (6.53) (included in P_{gen}) would be transformed into the fact $holds(f, v, 0)$ and so M would not be model of P'^M (and so, neither stable model), since it does not contain this atom. Assume that, instead, it contains more than one value for fluent f. Then, in the modulo P'^M, all the rules (6.53) for fluent f would be deleted, since their bodies contain *not* $holds(f, v_j, i)$ for all the values in $range(f)$ but one, and M contains at least two of them. Looking at the rest of the rules in P', it is easy to see that no other rule has some atom $holds(f, v, 0)$ as head. Therefore, the least model of P'^M will not contain atoms of that shape (i.e., as M contains them, it is not a stable model).

An analogous reasoning can be done for action executions: for each action a and situation $i > 0$, M must contain either some unique atom $holds(a, v, i)$ or the atom $pert(a, \mathtt{n}, i)$. We do not provide the proof, whose only variation is that we must also consider rule (6.52), which completes the negative cycle for the case of nonexecution of the action.

Therefore, we can extract from the atoms in M the corresponding (complete) initial situation σ_0 and a (complete) sequence of actions $\vec{\alpha}$. Finally, we have to prove that M is stable model of $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$. This is straightforward since, again, $P'^M = P^M$.

Supported models:

As happened with stable models, we begin proving that given some σ_0 and $\vec{\alpha}$, the supported model M of $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$ is supported model of $P' = P(R) \cup P_{gen}$. To see this, consider COMP$[P]$ and COMP$[P']$. It is easy to see that, in the case of P, completion can be just applied to $P(R)$:

$$\text{COMP}[P] = \text{COMP}[P(R)] \cup atoms(\sigma_0) \cup \ \mathbf{do}\ (\vec{\alpha})$$

since the rest of formulas are just atoms. In the case of P', as $P(R)$ does not contain any rule with head $holds(f, v, 0)$ nor $holds(a, v, i)$ we also have that:

$$\text{COMP}[P'] = \text{COMP}[P(R)] \cup \text{COMP}[P_{gen}]$$

where P_{gen} contains the formulas:

$$\begin{aligned} holds(F, v, 0) &\equiv \ not\ holds(F, v_1, 0), \ldots, not\ holds(F, v_m, 0) \\ holds(A, u, I) &\equiv \ not\ holds(A, u_1, I), \ldots, not\ holds(A, u_m, I), not\ pert(A, \mathtt{n}, I) \end{aligned}$$

for any $I \in [1, n]$, for any fluent F, action A, any $v \in range(F)$ with $\{v_1, \ldots, v_m\} = range(F) - \{v\}$, and any $u \in range(A)$ with $\{u_1, \ldots, u_m\} = range(A) - \{u\}$.

Since M is model of COMP$[P]$, it is also model of COMP$[P(R)]$, so we only have to prove that it is model of COMP$[P_{gen}]$. But COMP$[P_{gen}]$ asserts that any fluent is assigned an unique value at situation 0 whereas for actions, it asserts that either they are assigned a unique value, or they are nonpertinent. Since M is model of P', by correspondence theorem (9), it contains the same set of atoms than the narrative in the operational behavior, and so, it must satisfy COMP$[P_{gen}]$.

Now, we have to prove the other direction, that is, given any supported model of P', there exists a σ_0 and a $\vec{\alpha}$ such that M is supported model of $P = P(R) \cup atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$. As we have seen, satisfaction of COMP$[P_{gen}]$ implies that any model M defines a complete initial situation σ_0 and a complete actions execution $\vec{\alpha}$. As M is model of COMP$[P']$ it must also be model of COMP$[P(R)]$ and clearly $M \models atoms(\sigma_0) \cup$ **do** $(\vec{\alpha})$. Therefore, $M \models$ COMP$[P]$. $\square$

Proof of theorem 11

Some initial remarks. First, remember that we implicitly understood that nonquantified variables are actually universally quantified. Besides, we handle the standard axiomatization for the integer sort, including arithmetic operations and inequalities.

To show the equivalence, we will prove that both:

(i) $(8.42) \wedge (8.43) \models (8.44) \wedge (8.45)$

(ii) $(8.44) \wedge (8.45) \models (8.42) \wedge (8.43)$

Proof of (i)

Showing that $(8.42) \wedge (8.43) \models (8.45)$ is straightforward, by simply applying in (8.43) the substitutions $I_1 = I$ and $I_3 = I+1$ and observing that, for integer numbers, there does not exist any I_2 strictly between I and $I+1$.

For proving (8.44), for any fluent F, consider the formula:

$$\neg\exists A, I_2, V'.\ \big(0 < I_2 \wedge I_2 < I+1 \wedge \mathit{happens}(A, I_2) \wedge \mathit{set}(A, F, V', I_2)\big) \tag{A-1}$$

If we assume (A-1) is true, from (8.42) with $I_3 = I+1$ we get:

$$\mathit{holds}(F, V, 0) \supset \mathit{holds}(F, V, I+1)$$

which, by lemma 2, is equivalent to:

$$\mathit{holds}(F, V, 0) \equiv \mathit{holds}(F, V, I+1) \tag{A-2}$$

But, in the same way, as (A-1) covers situations strictly lower than $I+1$, it is also true for those strictly lower than I. Therefore, taking $I_3 = I$ in (8.42) and following analogous steps, we obtain:

$$\mathit{holds}(F, V, 0) \equiv \mathit{holds}(F, V, I) \tag{A-3}$$

And now, from (A-2) and (A-3) we finally get:

$$\mathit{holds}(F, V, I) \equiv \mathit{holds}(F, V, I+1)$$

which directly implies (8.44), since it is its consequent.

Now, assume that (A-1) is false instead. Then, we can take an action, let us call it B, in some situation I_1 between 0 and $I+1$ that happens and sets some value v for F. Besides, for all those actions, we can freely take B as the (chronologically) latest one, so that:

$$\mathit{happens}(B, I_1) \wedge \mathit{set}(B, F, V, I_1) \wedge \neg\exists A, I_2, V'.\ \big(I_1 < I_2 \wedge I_2 < I+1 \wedge \mathit{happens}(A, I_2) \wedge \mathit{set}(A, F, V', I_2)\big) \tag{A-4}$$

It is easy to see that we can apply (A-4) together with (8.43) both for $I_3 = I$ and $I_3 = I+1$, obtaining in this way both $\mathit{holds}(F, v, I)$ and $\mathit{holds}(F, v, I+1)$. But, due to axioms (6.2) and (6.1), this implies:

$$\mathit{holds}(F, V, I) \equiv \mathit{holds}(F, V, I+1)$$

and so (8.45) is also true.

Proof of (ii)

It is quite direct. To show (8.42) notice that if its antecedent is true, then we can inductively apply (8.44) for $I = 0 \ldots I_3 - 1$ to successively obtain $holds(F, V, 1), \ldots, holds(F, V, I_3)$. For showing (8.43), if its antecedent is true, then we can apply (8.45) with $I = I_1$ to obtain $holds(F, V, I_1 + 1)$. But now, we can apply again (8.44) inductively for $I = I_1 + 1 \ldots I_3 - 1$ to obtain $holds(F, V, I_1 + 2), \ldots, holds(F, V, I_3)$. □

Proof of theorem 14

As a proof sketch, we may use the classical logic encoding of L^2. Notice that the conjunction of the first two rules is equivalent to:

$$(\phi \wedge !\phi \supset \psi \wedge !\psi) \quad \wedge \quad (\phi \wedge !\phi \supset \gamma \wedge !\gamma)$$

and this, in its turn, is equivalent to:

$$\phi \wedge !\phi \quad \supset \quad (\psi \wedge !\psi \wedge \gamma \wedge !\gamma)$$

Finally, note that this formula entails:

$$\phi \wedge !\phi \quad \supset \quad (\psi \wedge \gamma \wedge (!\psi \vee !\gamma))$$

which corresponds to:

$$\psi \wedge \gamma \quad \Leftarrow \quad \phi$$

□

Proof of theorem 15

Again, we may use the classical encoding of L^2. As we had seen, $A \Leftarrow B \vee C$ is equivalent to:

$$(B \vee C) \wedge (!B \vee !C) \supset A \wedge !A$$

which corresponds to the conjunction of the four implications:

$$B \wedge !B \quad \supset \quad A \wedge !A \tag{A-5}$$
$$B \wedge !C \quad \supset \quad A \wedge !A \tag{A-6}$$
$$C \wedge !B \quad \supset \quad A \wedge !A \tag{A-7}$$
$$C \wedge !C \quad \supset \quad A \wedge !A \tag{A-8}$$

Now, simply notice that (A-5) and (A-8) are respectively equivalent to $A \Leftarrow B$ and $A \Leftarrow C$. □

Appendix B

PAL examples

B-1 Newton's formula

(page 1)

```
#file force.pal
options
  not concurrent, solutions=1;
sets
  int = [0,20];
fluents
  f,m,a : int;
actions
  apply_force : int;
rules
  f:=apply_force;
  a:=f/m if m!=0;

initially
  m:=3,a:=0,f:=0;
do {apply_force:=15;}
```

Notice how the causal rule transforms $\mathtt{f} = \mathtt{m} \cdot \mathtt{a}$ into $\mathtt{a} = \mathtt{f}/\mathtt{m}$, since the real effect is the acceleration a. The output for the proposed execution would be:

```
1)
apply_force:=15
f:=15
a:=5
```

Note also how the mass m is not shown (it remains unchanged). We can also compute the force knowing the acceleration, proposing a planning problem:

```
initially
  m:=3,a:=0,f:=0;
```

```
query
  ; a=5?
```

The output of PAL interpreter would be in this case:

```
Solution 1:
1)
apply_force:=15
f:=15
a:=5
1 solution
```

which, although it seems the same result as before, it is actually proposing the hypothetical execution of `apply_force:=15` without changing the current state. If we remove the option `solutions=1` we actually obtain more solutions, since the causal rule handles an integer division and so, values 16 and 17 are also correct answers.

Computing the mass in terms of the acceleration and the force is still not possible using the PAL interpreter, since temporal explanation problems for completing the initial state are not available yet.

B-2 Lin's suitcase

(page 1)

```
# file suitcase.pal
sets
  lock={1,2};
fluents
  open: boolean;
  up: lock -> boolean;
actions
  toggle: lock -> event;
vars
  L:lock;
rules
  up(L):=not prev(up(L)) if toggle(L);
  open if up(1) and up(2);

initially
  not up(1),up(2),not open;
do{ toggle(1); }
```

The proposed execution corresponds to the typical Lin's suitcase problem. The output from PAL interpreter is the expected one:

```
1)
toggle(1):=true
open:=true
up(1):=true
```

that is, the suitcase results open.

B-3 Yale Shooting Problem

(chapter 1, page 4)

The Yale Shooting Problem can be encoded in PAL as follows:

```
#file yale.pal
actions
  shoot,load : event;

fluents
  loaded,alive: boolean;

rules
  loaded     if load;
  not alive  if shoot and prev(loaded);
  not loaded if shoot and prev(loaded);

initially
  alive, not loaded;
do {
  load;
  ;
  shoot;
}
```

whose resulting output is:

```
1)
load:=true
loaded:=true
2)
3)
shoot:=true
loaded:=false
alive:=false
```

B-4 Lamp circuit

(chapter 1, page 9)
The lamp circuit domain can be simply formalized in PAL as follows:

```
#file lamp.pal
sets
  switch={1,2};
fluents
```

```
  light: boolean;
  sw: switch -> boolean;
actions
  toggle: switch -> event;
vars
  S:switch;
rules
  sw(S):=not prev(sw(S)) if toggle(S);
  light:=sw(1) and sw(2);
```

The four cases commented in the introduction would correspond to the transitions:

```
# case 1: opening sw(1) while sw(2) was closed
initially sw(1), sw(2), light;
do { toggle(1); }

# case 2: closing sw(2) while sw(1) was open
initially not sw(1), not sw(2), not light;
do { toggle(2); }

# case 3: opening sw(1) and closing sw(2) simultaneously
initially sw(1), not sw(2), not light;
do { toggle(1),toggle(2); }

# case 4: perform no actions, having sw(1) open and sw(2) closed
initially not sw(1), sw(2), not light;
do { ; }
```

These are the respective results for the four transitions:

```
1)
toggle(1):=true
light:=false
sw(1):=false

Restart
1)
toggle(2):=true
light:=false
sw(2):=true

Restart
1)
toggle(1):=true
toggle(2):=true
light:=false
sw(1):=false
sw(2):=true
```

```
Restart
1)
```

Notice that the resulting state is the same one in all cases, but we can see different outputs because we are only shown the pertinent facts, with respect to each of the four initial states. Note also how the result for case 3 has made `light` to become pertinent although it was also previously false. This, as was explained in the introduction, is can be interpreted as the possibility of a momentary flash in the light, depending on the accuracy of the simultaneous movement of the switches.

B-5 Combinatorial circuit

(page 41)

```
#file circuit.pal
fluents
  a,b,c,d : boolean;
actions
  set_c, set_a : boolean;

rules
  d:=a and (c or not c);
  b:=not a;
  a:=set_a;
  c:=set_c;

initially
  a:=true, c:=false, b:=false, d:=true;

do { set_c:=true; }
```

The output we obtain after setting `c` true is:

```
1)
set_c:=true
c:=true
d:=true
```

that is, fluent `d` is also caused true, although it was also previously true. Notice how the intervention in `c` has affected `d` even though the presence of `c` in the rule for `d` is a classical tautology. On the other hand, fluent `b` has no dependence on `c` and so, it does not result affected, persisting false as before.

B-6 Account balance

(example 2, page 12)

The domain for computing the balance account can be formalized as:

```
#file balance.pal
sets
  int = [-50,50];

fluents
  transac,balance: int;

actions
  deposit,withdraw: int;

rules
  transac:=deposit;
  transac:=-withdraw;
  balance:=prev(balance)+transac;
```

It must be noticed that we are free to add new rules or to modify the existing one for computing the last transaction to be included in the balance *without needing to modify at all* the rule for fluent `balance`. To see an example of how pertinence is important in this domain, consider the execution:

```
initially
  balance:=0,transac:=0;

do {
  deposit:=35;
  withdraw:=22;
  ;
  deposit:=10;
  deposit:=10;
  ;
  deposit:=10;
}
```

The obtained output is the following one:

```
1)
deposit:=35
transac:=35
balance:=35
2)
withdraw:=22
transac:=-22
balance:=13
3)
4)
deposit:=10
transac:=10
balance:=23
```

```
5)
deposit:=10
transac:=10
balance:=33
6)
7)
deposit:=10
transac:=10
balance:=43
```

The key point here is that fluent `transac` has value 10 in situations 4 to 7, but situation 6 is not taken into account, because in that case the fluent `transac` *has persisted.* In other words, the `balance` is modified only when the `transac` is pertinent.

B-7 The gong example

(example 5, page 66)
The gong scenario can be encoded in PAL practically without any variation with respect to the elementary syntax of $\mathcal{P}$-language:

```
#file gong.pal
fluents
  gong, dancing: boolean;
actions
  finish, strike: event;
rules
  gong if strike;
  dancing if gong;
  not dancing if finish;

initially
  not gong, not dancing;
do { strike; finish; }
```

The sequential execution of the two actions above leads to:

```
1)
strike:=true
gong:=true
dancing:=true
2)
finish:=true
dancing:=false
```

Notice how, although `gong` remains true at situation 2, the rule `dancing if gong` is not applied, because `gong` is not pertinent.

B-8 The alarm problem

(example 6, page 78)

As happened with the gong example, the representation in PAL of the alarm scenario is practically the same as in $\mathcal{P}$-rules:

```
#file alarm.pal
fluents
  in, active, ring;
actions
  enter, disconnect, connect;

rules
  active if connect;
  not active if disconnect;
  in if enter;
  ring if in and active and not pert(active);

initially
  active, not in, not ring;
do {enter;}

initially
  active, not in, not ring;
do {enter,disconnect; connect;}
```

In this file, we have included two proposed temporal projections problems. For the first case, we simply check that when someone enters the building while the alarm is active the bell is caused to ring, that is:

```
1)
enter:=true
in:=true
ring:=true
```

In the second execution, while the person enters the building, this time we disconnect the alarm simultaneously. Afterwards, we connect the alarm again, once the person was already inside:

```
Restart
1)
enter:=true
disconnect:=true
in:=true
active:=false
2)
connect:=true
active:=true
```

Notice that `ring` persists false without any change since the initial situation. Connecting the alarm does not cause the bell to ring because this only happens when `in` is caused true and `active` persists true, but is not caused.

B-9 The gear wheels

(example 15, page 97)

The first version of the gear wheels is simply represented as:

```
#file wheels.pal
sets
  wheel = {1,2};
fluents
  turn : wheel -> boolean;
actions
  start, stop : wheel -> event;
vars
  W : wheel;

rules
  turn(W) if start(W);
  not turn(W) if stop(W);
  turn(1):=turn(2);
  turn(2):=turn(1);
```

Just as an example, the execution:

```
initially not turn(W);
do { ; start(1); ; stop(2); start(2); }
```

yields the output:

```
1)
2)
start(1):=true
turn(1):=true
turn(2):=true
3)
4)
stop(2):=true
turn(1):=false
turn(2):=false
5)
start(2):=true
turn(1):=true
turn(2):=true
```

Now, consider the first variation, that is, we add the coupling mechanism as in the set of $\mathcal{P}$-rules (7.34)-(7.34):

```
#file coupled.pal
sets
  wheel = {1,2};
fluents
  turn : wheel -> boolean;
  coupled: boolean;
actions
  start, stop : wheel -> event;
  uncouple, couple : event;
vars
  W : wheel;

rules
  turn(W) if start(W);
  not turn(W) if stop(W);
  coupled if couple;
  not coupled if uncouple;
  turn(1):=turn(2) if coupled;
  turn(2):=turn(1) if coupled;

initially
  turn(1), not turn(2), not coupled;
do { couple; }
```

When we use the WFS interpretation, the execution above yields the output:

```
1)
couple:=true
turn(1):=?   Unfounded values={ }
turn(2):=?   Unfounded values={ }
coupled:=true
```

whereas we obtain no stable models.

The second variation (file `coupled2.pal`) solves the problem by adding the rule:

```
not turn(W) if coupled;
```

which corresponds to (7.34)-(7.35), so that coupling also causes that both wheels are stopped. The same execution yields now:

```
1)
couple:=true
turn(1):=false
turn(2):=false
coupled:=true
```

Finally, the last variation consists in:

```
#file coupled3.pal
sets
  wheel = {1,2};
fluents
  turn : wheel -> boolean;
  uncouple, coupled: boolean;
actions
  start, stop : wheel -> event;
  couple : event;
vars
  W : wheel;

rules
  turn(W) if start(W);
  not turn(W) if stop(W);
  coupled if couple;
  not coupled if uncouple;
  turn(1):=turn(2) if coupled and not pert(coupled);
  turn(2):=turn(1) if coupled and not pert(coupled);
  false if couple and prev(turn(W));
```

so that we avoid coupling wheels that are not stopped and, additionally, we require that the connection between the rules is only active when `coupled` *persists true*. For instance, when we execute:

```
initially
  not turn(1), not turn(2), not coupled;
do { couple; }
```

we actually obtain:

```
1)
couple:=true
coupled:=true
```

that is, the fluents `turn(W)` *are not affected* by coupling the wheels.

B-10 Shanahan's relay

(figure 7.4, page 101)

In this case, we can simplify the $\mathcal{P}$-rules representation, directly assigning boolean expressions to fluents `light` and `relay`:

```
#file relay2.pal
sets
  lock={1,2,3};
fluents
  light,relay: boolean;
```

```
    sw: lock -> boolean;
  actions
    toggle: lock -> event;
  vars
    L:lock;
  rules
    sw(L):=not prev(sw(L)) if toggle(L);
    light:=sw(1) and sw(2);
    relay:=sw(1) and sw(2) and sw(3);
    not sw(2) if relay;
```

The problem in the circuit design becomes evident using the execution:

```
initially
  not sw(1),sw(2),sw(3),not relay, not light;
do {toggle(1);}
```

which yields no stable model whereas, in WFS, we obtain:

```
1)
toggle(1):=true
light:=?   Unfounded values={ }
relay:=?   Unfounded values={ }
sw(1):=true
sw(2):=?   Unfounded values={ }
```

B-11 The soup bowl

(example 16, page 106)

The soup bowl scenario can be captured using non-pertinence of an action for testing its non-occurrence:

```
#file soup.pal
sets
  side = {left,right};

actions
  lift: side -> event;

fluents
  spilled: boolean;

vars
  S1,S2 : side;

rules
  spilled if S1!=S2 and lift(S1) and not pert(lift(S2));
```

Thanks to the use of variables, we can summarize the behavior in a quite straightforward single rule. The executions:

```
initially not spilled;
do { lift(right); }

initially not spilled;
do { lift(left),lift(right); }
```

yield the output:

```
1)
lift(right):=true
spilled:=true

Restart
1)
lift(left):=true
lift(right):=true
```

that is, we must lift both sides in order to avoid spilling the soup.

As an example of query, we could ask a way of lifting some side while avoiding to spill the soup:

```
initially not spilled;
query ; not spilled and lift(S1) ?
```

We obtain two answers, depending on how we instantiate variable `S1`:

```
Solution 1:
S1=left
1)
lift(left):=true
lift(right):=true

Solution 2:
S1=right
1)
lift(left):=true
lift(right):=true

2 solutions
```

although the "plan" is the same in both cases: lift both sides simultaneously.

B-12 Thielscher's relay

(example 17, page 123)

The representation of Thielscher's relay using PAL is very similar to the one for Shanahan's relay we saw before:

```
#file relay.pal
sets
  lock={1,2,3};
fluents
  light,relay: boolean;
  sw: lock -> boolean;
actions
  toggle: lock -> event;
vars
  L:lock;
rules
  sw(L):=not prev(sw(L)) if toggle(L);
  light:=sw(1) and sw(2);
  relay:=sw(1) and sw(3);
  not sw(2) if relay;
```

The configuration depicted in figure 8.1 corresponds to the initial state:

```
initially
  not sw(1), sw(2), sw(3), not relay, not light;
```

and, after performing `do {toggle(1);}` we obtain:

```
1)
toggle(1):=true
light:=false
relay:=true
sw(1):=true
sw(2):=false
```

that is, the relay results connected and the light is caused to be off, although it was already off before.

B-13 The trapdoor

(example 18, page 124)

The representation in PAL that directly corresponds to Thielscher's formulation for the trapdoor scenario is:

```
#file trapdoor.pal
fluents
  trapdoor_open, at_trap, alive: boolean;
actions
  open, entice: event;
rules
  trapdoor_open if open and not prev(trapdoor_open);
  at_trap if entice and not prev(at_trap) and prev(alive);
  alive if entice and not prev(at_trap) and prev(alive);
```

```
    not alive if trapdoor_open and at_trap and not pert(at_trap);
    false if trapdoor_open and at_trap and alive;
```

Assume that the trap is closed and the turkey alive but not at the trap.

```
initially
  not at_trap, not trapdoor_open, alive;
```

If we perform:

```
do { entice; open; }
```

the turkey is finally killed:

```
1)
entice:=true
at_trap:=true
alive:=true
2)
open:=true
trapdoor_open:=true
alive:=false
```

However, changing the order of the actions for the same initial situation:

```
do { open; entice; }
```

prevents the turkey to be actually enticed:

```
Restart
1)
open:=true
trapdoor_open:=true
Inconsistence (rule 5, line 12, applied with 'false' head).
```

This representation, however, does not seem the most direct one for the considered domain. For instance, one could think about a third action `push` that forces the turkey to be `tat_trap` even when the trapdoor is open. In this case, the turkey should result killed too, but this time, because of a change in `at_trap`. Therefore, a more appropriated formulation could perhaps be:

```
#file trapdoor2.pal
fluents
  trapdoor_open, at_trap, alive: boolean;
actions
  open, entice, push: event;
rules
  trapdoor_open if open;
  at_trap if push;
  at_trap if entice;
  false if entice and prev(trapdoor_open);
  not alive if trapdoor_open and at_trap;
```

This representation yields the same result for the two previous prediction problems, but also allows:

```
initially
  not at_trap, not trapdoor_open, alive;
do { open; push; }
```

so that the turkey is finally killed by a change in `at_trap`:

```
1)
open:=true
trapdoor_open:=true
2)
push:=true
at_trap:=true
alive:=false
```

Bibliography

[1] M. Abadi and Z. Manna. Temporal logic programming. *Journal of Symbolic Computation*, 8:277–295, 1989.

[2] J. J. Alferes. *Semantics of Logic Programs with Explicit Negation.* PhD thesis, Facultade de Ciências e Tecnologia, Universidade Nova de Lisboa, 1993.

[3] J. J. Alferes, L. M. Pereira, and T. C. Przymusinski. 'classical' negation in nonmonotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20(1):107–142, 1998.

[4] A. R. Anderson and N. D. Belnap. *Entailment: The Logic of Relevance and Necessity*, volume 1. Princeton University Press, Princeton, USA, 1975.

[5] G. Antoniou. *Nonmonotonic Reasoning.* The MIT Press, 1997.

[6] K. R. Apt and R. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 19-20:9–71, 1994.

[7] B. Ari. *Mathematical Logic for Computer Science.* Prentice Hall, 1993.

[8] A. B. Baker. Nonmonotonic reasoning in the framework of the situation calculus. *Artificial Intelligence*, 49(1–3):5–23, 1991.

[9] C. Baral and M. Gelfond. Reasoning about effects of concurrent actions. *Journal of Logic Programming*, 31(1-3):85–117, 1997.

[10] C. Baral, M. Gelfond, and A. Provetti. Reasoning about actions: Laws, observations and hypotheses. *Journal of Logic Programming*, 31, 1997.

[11] C. Baral, J.Lobo, and J.Minker. Generalized disjunctive well-founded semantics for logic programs. *Annals of Math and Artificial Intelligence*, 5:89–132, 1992.

[12] C. Baral and J. Lobo. Defeasible specifications in action theories. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1441–1446, Nagoya, Japan, 1997.

[13] C. Baral and L. Tuan. Reasoning about actions in a probabilistic setting. In *Proc. of the Fifth Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense 2001)*, 2001.

[14] M. Baudinet. A simple proof of the completeness of temporal logic programming. In L. Fariñas del Cerro and M. Penttonen, editors, *Intensional Logics for Programming*, pages 51–83. Clarendon Press, Oxford, 1992.

[15] K. Van Belleghem, M. Denecker, and D. T. Dupré;. Ramifications in an event-based language. In W. Daelemans K. Van Marcke, editor, *Proceedings of the Ninth Dutch Conference on Artificial Intelligence (NAIC'97)*, pages 227–236, November 1997.

[16] N. Bidoit and C. Froidevaux. Minimalism subsumes default logic and circumscription. In *Proc. of the IEEE Symposium on Logic in Computer Science (LICS-87)*, pages 89–97, 1987.

[17] S. Brass, J. Dix, B. Freitag, and U. Zukowski. Transformation-based bottom-up computation of the well-founded model. *Theory and Practice of Logic Programming*, to appear, 2001. (Draft version available at `http://www.cs.man.ac.uk/~jdix/Papers/01\_TPLP.ps.gz`).

[18] G. Brewka. *Nonmonotonic Reasoning: Logical Foundations of Commonsense.* Cambridge University Press, 1991.

[19] P. Cabalar. Temporal answer sets. In *Proceedings of the Joint Conference on Declarative Programming (APPIA-GULP-PRODE'99)*, L'Aquila, Italy, September 1999.

[20] P. Cabalar. Well founded semantics as two-dimensional here-and-there. In *Proceedings of the Workshop on Answer Set Programming (ASP'01). 2001 AAAI Spring Symposium Series.*, Stanford, California, March 2001.

[21] P. Cabalar, M. Cabarcos, and R. P. Otero. PAL: Pertinence action language. In *Proceedings of the 8th Intl. Workshop on Non-Monotonic Reasoning NMR'2000 (Collocated with KR'2000)*, Breckenridge, Colorado, USA, april 2000. (`http://xxx.lanl.gov/abs/cs.AI/0003048`).

[22] P. Cabalar, R. P. Otero, and S. G. Pose. Temporal constraint networks in action. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence*, Berlin, Germany, August 2000. IOS Press.

[23] CCALC (causal calculator) web page:
`http://www.cs.utexas.edu/users/tag/cc` .

[24] CHAFF web page:
`http://www.ee.princeton.edu/~chaff/` .

[25] K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 241–327. Plenum, 1978.

[26] J. M. Crawford. Ntab location at crawford's web page:
`http://www.cirl.uoregon.edu/crawford/ntab.tar` .

[27] J.M. Crawford and L.D. Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81(1):31–57, 1996.

[28] M. Denecker, D. Theseider, and K. van Belleghem. An inductive definition approach to ramifications. *Linköping Electronic Articles in Computer and Information Science*, 3(7), 1998. `URL: http://www.ep.liu.se/ea/cis/1998/007/` .

[29] J. Dix. Classifying semantics of disjunctive logic programs. In K. R. Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, Washington, 1992. ALP, MIT Press.

[30] J. Dix. Semantics of logic programs: Their intuitions and formal properties. an overview. In A. Fuhrmann and Hans Rott, editors, *Logic, Action and Information— Essays on Logic in Philosophy and Artificial Intelligence*, pages 241–327. De Gruyter, 1993.

[31] Jürgen Dix, Luis Pereira, and Teodor Przymusinski. Prolegomena to Logic Programming for Non-Monotonic Reasoning. In J. Dix, L. Pereira, and T. Przymusinski, editors, *Non-monotonic Extensions of Logic Programming*, LNAI 1216, pages 1–36. Springer, Berlin, 1997.

[32] DLV web page
`http://www.dbai.tuwien.ac.at/proj/dlv/`.

[33] P. Doherty, J. Gustafsson, L. Karlsson, and J. Kvarnström. Temporal action logics language specification and tutorial. *Linköping University Electronic Press, Series in Computer and Information Science*, 15, 1998.

[34] R. E. Fikes and N. J. Nilsson. Strips: A new approach to theorem proving in problem solving. *Artificial Intelligence Journal*, 2:189–208, 1971.

[35] M. Fitting. A kripke-kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[36] D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors. *Handbook of Logic in Artificial Intelligence and Logic Programming. Nonmonotonic Reasoning and Uncertain Reasoning*, volume 3. Oxford Science Publications, 1994.

[37] H. Geffner. Causal theories for nonmonotonic reasoning. In *Proceedings AAAI-90*, pages 524–530, 1990.

[38] M. Gelfond and V. Lifschitz. The stable models semantics for logic programming. In *Proc. of the 5th Intl. Conf. on Logic Programming*, pages 1070–1080, 1988.

[39] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proc. of the 7th Intl. Conf. on Logic Programming*, pages 579–597, 1990.

[40] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[41] M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *The Journal of Logic Programming*, 17:301–321, 1993.

[42] M. Gelfond and V. Lifschitz. Action languages. *Linköping Electronic Articles in Computer and Information Science*, 3(16), 1998. (`http://www.ep.liu.se/ea/cis/1998/016`).

[43] M. Gelfond, V. Lifschitz, H. Przymusińska, and M. Truszczyński. Disjunctive defaults. In *Proc. of the 2nd Intl. Conf. on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 230–237, 1991.

[44] L. Giordano, A. Martelli, and C. B. Schwind. Dealing with concurrent actions in modal action logic. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, pages 537–541, 1998.

[45] E. Giunchiglia, J. Lee, V. Lifschitz, and H. Turner. Causal laws and multi-valued fluents. (unpublished draft) `http://www.cs.utexas.edu/users/vl/mypapers/clmvf-long.ps`.

[46] E. Giunchiglia and V. Lifschitz. An action language based on causal explanation: Preliminary report. In *Proc. of the AAAI-98*, pages 623–630, 1998.

[47] J. Gustafsson and P. Doherty. Embracing occlusion in specifying the indirect effects of actions. In *Proc. of the 5th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, 1996.

[48] S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artificial Intelligence Journal*, 33:379–413, 1987.

[49] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, volume II (Extensions of Classical Logic)*, pages 497–604. Kluwer Academic Publishers, Dordrecht (NL), 1984.

[50] B. A. Haugh. Simple causal minimizations for temporal persistence and projection. In *Proceedings of the 6th National Conference of Artificial Intelligence*, pages 218–223, 1987.

[51] A. Kakas, R. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 6?(2?):719–770, 1993.

[52] H. Kautz. The logic of persistence. In *Proceedings of the 5th National Conference of Artificial Intelligence*, pages 401–405, 1986.

[53] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[54] V. Lifschitz. Computing circumscription. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 121–127, 1985.

[55] V. Lifschitz. Pointwise circumscription: Preliminary report. In *Proceedings of the 5th National Conference of Artificial Intelligence*, pages 406–411, 1986.

[56] V. Lifschitz. Formal theories of action (preliminary report). In *Proc. of the 10th IJCAI*, pages 966–972, Milan, Italy, 1987.

[57] V. Lifschitz. Frames in the space of situations. *Artificial Intelligence*, 46:365–376, 1990.

[58] V. Lifschitz. Circumscription. In C.J. Hogger D.M. Gabbay and J.A. Robinson, editors, *Handbook of Logic in AI and Logic Programming*, volume 3, pages 298–352. Oxford University Press, 1993.

[59] V. Lifschitz. Foundations of logic programming. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 69–127. CSLI Publications, Stanford, California, 1996.

[60] V. Lifschitz. On the logic of causal explanation. *Artificial Intelligence Journal*, 96:451–465, 1997.

[61] V. Lifschitz. Action languages, answer sets and planning. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[62] V. Lifschitz. Success of default logic. In *Logical Foundations for Cognitive Agents: Contributions in Honour of Ray Reiter*, pages 208–212. Springer Verlag, 1999.

[63] V. Lifschitz. M. Shanahan, solving the frame problem. *Artificial Intelligence*, 123(1-2):265–268, 2000.

[64] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2000.

[65] F. Lin. Embracing causality in specifying the indirect effects of actions. In C. S. Mellish, editor, *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, Montreal, Canada, August 1995. Morgan Kaufmann.

[66] J.W. Lloyd. *Foundations of Logic Programming (2nd ed).* Springer-Verlag, 1987.

[67] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming.* The MIT Press, 1992.

[68] D. Lorenzo. *Learning non-monotonic logic programs to reason about actions and change.* PhD thesis, Facultade de Informática, Universidade da Coruña, 2001. (to appear).

[69] N. McCain and H. Turner. A causal theory of ramifications and qualifications. In C. S. Mellish, editor, *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 1978–1984, Montreal, Canada, August 1995. Morgan Kaufmann.

[70] N. McCain and H. Turner. Causal theories of action and change. In *Proc. of the AAAI-97*, pages 460–465, 1997.

[71] N. McCain and H. Turner. Satisfiability planning with causal theories. In *Proc. of the 6th Intl. Conf. of Knowledge Representation and Reasoning*, 1998.

[72] N. C. McCain. *Causality in commonsense reasoning about actions.* PhD thesis, University of Texas at Austin, 1997.

[73] J. McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, 1959.

[74] J. McCarthy. Circumscription: A form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[75] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 26(3):89–116, 1986.

[76] J. McCarthy. What is AI?, 1997. Available at:
`http://www-formal.stanford.edu/jmc/whatisai.html`.

[77] J. McCarthy. Elaboration tolerance. In *Proc. of the 4th Symposium on Logical Formalizations of Commonsense Reasoning (Common Sense 98)*, pages 198–217, London, UK, 1998. Updated version at
`http://www-formal.stanford.edu/jmc/elaboration.ps`.

[78] J. McCarthy. Approximate objects and approximate theories. In *Proc. of the 7th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pages 519–526. Morgan Kaufmann, 2000.

[79] J. McCarthy. M. Shanahan, solving the frame problem. *Artificial Intelligence*, 123(1-2):269–270, 2000.

[80] J. McCarthy and T. Costello. Combining narratives. In *Proc. of the 6th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pages 48–59, Trento, Italy, 1998. Morgan Kaufmann.

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

[82] The Medtool project and related activities are described in web documents `http://www.dc.fi.udc.es/ai/medtool.html`, 2001.

[83] M. Moskewicz, C. Madigana, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *39th Design Automation Conference*, 2001.

[84] M. Otero and R. P. Otero. Using causality for diagnosis. In *Proc. of 11th Int. Workshop on Principles of Diagnosis (DX-00)*, pages 171–176, Morelia, Mexico, 2000.

[85] R. P. Otero. Pertinence logic for reasoning about actions and change. Technical Report TR-AI-97-01, AI Lab., Dept. of Computer Science, University of A Coruña, 1997.

[86] R. P. Otero and P. Cabalar. Pertinence and causality. In *Proc. of the Nonmonotonic Reasoning Actions and Change Workshop (NRAC), at the Intl. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 111–119, Stockholm, Sweden, 1999.

[87] PAL web page:
`http://www.dc.fi.udc.es/ai/~cabalar/pal/`.

[88] J. Pearl. *Causality*. Cambridge University Press, 2000.

[89] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligence (ECAI'92)*, pages 102–106, Montreal, Canada, 1992. John Wiley & Sons.

[90] J. Pinto. Causality in theories of action. In *Fourth Symposium on Logical Formalizations of Commonsense Reasoning*, pages 349–364, London, U.K., 1998.

[91] J. Pinto. Causality, indirect effects and triggers (preliminary report). In *Seventh International Workshop on Non-monotonic Reasoning*, Trento, Italy, 1998.

[92] T. Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.

[93] T. C. Przymusinski. Stationary semantics for disjunctive logic programs and deductive databases. In *Proceedings of the North American Logic Programming Conference*, pages 40–59, Austin, Texas, 1990. MIT Press.

[94] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[95] R. Reiter. Narratives as programs. In *Proc. of the 7th Intl. Conf. on Principles of Knowledge Representation and Reasoning*, pages 99–108, Breckenridge, Colorado, USA, 2000. Morgan Kaufmann.

[96] E. Sandewall. Filter preferential entailment for the logic of action in almost continuous worlds. In C. S. Mellish, editor, *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 894–899. Morgan Kaufmann, 1989.

[97] E. Sandewall. *Features and Fluents. A Systematic Approach to the Representation of Knowledge about Dynamical Systems.* Oxford University Press, 1994.

[98] E. Sandewall. Transition cascade semantics and first assessments results for ramification. In Oliviero Stock, editor, *Spatial and Temporal Reasoning.* Kluwer Publishing Company, 1997.

[99] E. Sandewall. M. Shanahan, solving the frame problem. *Artificial Intelligence*, 123(1-2):271–273, 2000.

[100] Camilla Schwind. Causality in action theories. *Linköping University Electronic Press, Series in Computer and Information Science*, 4(4), May 1999. `http://www.ep.liu.se/ea/cis/1999/004/`.

[101] M. Shanahan. *Solving the Frame Problem.* The MIT Press, 1997.

[102] M. Shanahan. M. Shanahan, solving the frame problem. *Artificial Intelligence*, 123(1-2):275, 2000.

[103] Murray P. Shanahan. The ramification problem in the event calculus. In *Proc. of the Intl. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 140–146, 1999.

[104] Y. Shoham. Chronological ignorance: time, nonmonotonicity, necessity and causal theories. In *Proceedings of the 5th National Conference of Artificial Intelligence*, pages 389–393, 1986.

[105] SMODELS web page `http://www.tcs.hut.fi/software/smodels/`.

[106] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[107] M. Thielscher. Ramification and causality. *Artificial Intelligence Journal*, 1-2(89):317–364, 1997.

[108] M. Thielscher. Reasoning about actions: Steady versus stabilizing state constraints. *Artificial Intelligence*, 104:339–355, 1998.

[109] H. Turner. Representing actions in logic programs and default theories: A situation calculus approach. *Journal of Logic Programming*, 31:245–298, 1997.

[110] H. Turner. *Causal Action Theories and Satisfiability Planning.* PhD thesis, University of Texas at Austin, 1998.

[111] Hudson Turner. A logic of universal causation. *Artificial Intelligence*, 113(1–2):87–123, 1999.

[112] M. H. van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23:733–742, 1976.

[113] A. van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[114] H. Zhang. SATO web page:
`http://www.cs.uiowa.edu/~hzhang/sato/`.

[115] H. Zhang. SATO: An efficient propositional prover. In *Conference on Automated Deduction*, pages 272–275, 1997.