Reasoning about Actions in Transaction Logic

Martín Rezk

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

Fakultät für Informatik | **Facoltà di Scienze e Tecnologie informatiche** | **Faculty of Computer Science**

For further information about FUB-CS publications, please contact

Facoltà di Scienze e Tecnologie Informatiche	Fakultät für Informatik	Faculty of Computer Science
Libera Università di Bolzano	Freie Universität Bozen	Free University of Bozen-Bolzano
Piazza Domenicani 3	3 Domenikanerplatz	Piazza Domenicani 3
39100 Bolzano	39100 Bozen	39100 Bozen-Bolzano
Italia	Italien	Italy

tel: +39 0471 016 000
fax: +39 0471 016 009
e-mail: cs-secretariat@unibz.it
homepage: http://www.unibz.it/inf

Reasoning about Actions in Transaction Logic

PhD Thesis in Computer Science

Martín Rezk

Dissertation advisors:

Prof. Michael Kifer
Department of Computer Science
Stony Brook University
New York, U.S.A.

Prof. Werner Nutt
KRDB Research Centre
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy

Thesis Evaluated by:

Prof. José Alferes
Department of Computer Science
Faculdade de Cincias e Tecnologia
Universidade Nova de Lisboa
Lisbon, Portugal

Dr. Marco Montali
KRDB Research Centre
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy

Examination Committee:

Prof. Luigi Palopoli
Dipartimento di Elettronica,
Informatica e Sistemistica
Universita degli Studi della Calabria
Rende, Italy

Prof. Sandro Morasca
Department of Sciences of Culture,
Politics and Information
Universita degli Studi dell'Insubria
Como, Italy

Prof. Francesco Ricci
Faculty of Computer Science
Free University of Bozen-Bolzano
Bozen-Bolzano, Italy

Date of public defense: 23/04/2012

Abstract

This thesis introduces $\mathcal{TR}^{PAD}$ (*Transaction Logic with Partially Defined Actions*)—an expressive formalism for reasoning about the effects of complex actions. $\mathcal{TR}^{PAD}$ is largely based on a subset of Transaction Logic, but extends it with special *premise*-formulas that generalize the data and transition formulas of the original Transaction Logic. We develop a sound and complete proof theory for $\mathcal{TR}^{PAD}$ and illustrate the formalism on a number of non-trivial examples. In addition, we show that most of $\mathcal{TR}^{PAD}$ is reducible to ordinary logic programming and that, in a well-defined sense, this reduction is sound and complete.

We also augment $\mathcal{TR}^{PAD}$ with default negation, and along the way we define a well-founded semantics for $\mathcal{TR}^{PAD}$, which, to the best of our knowledge, has never been done before. Finally, as an application, we use $\mathcal{TR}^{PAD}$ to give a declarative semantics to the combination of *production systems* and *ontologies*.

The results obtained in this thesis significantly advance the state of the art in the important subfield of *Artificial Intelligence* devoted to reasoning about actions. We expect these results to find applications in intelligent agent systems, semantic web services, question answering systems, and other areas.

Contents

Acknowledgments

> Each of us has his cowardice.
> Each of us is afraid to lose, afraid
> to die. But hanging back is the
> way to remain a coward for life.
> The Way to find courage is to seek
> it on the field of conflict.
>
> Masutatsu Oyama

Someone said that there is not such thing as silent gratitude, and maybe s/he was right. Then one needs to sit down and come up with a list of names, and find a way to say thanks. But before one writes the first line one realizes the two main issues in this task: *(i)* the huge number of people that one is in debt to, and *(ii)* how hard it is to thank without sounding like if it was just a protocolar duty. These two problems leads us to the following—general—claim: Greetings are not scalable. The more people we thank, the more it looks that they did not contribute so much to the final result. However, in my case, I truly owe a lot to many people, and that is because to be fair I would need to thank not only those who helped me in the last four years, but in the last twenty-nine. The people I want to mention here taught me how to overcome my own cowardice, tackle each battle, learn from my losses, and prevail.

My family gave me the very first tools, principles, curiosity, and support. It is so simple, and yet irreplaceable. My friends, spread all around the world: Argentina, USA, Europe, Asia; have always been a safe shelter on this journey, giving me advice, listening, shaping different views, opening new doors. My teachers and professors showed me how and where to seek for knowledge, and they put an order in the huge mess of ideas that I had in my head, they taught me how to learn. In particular, Emmanuel (Kyokushin Karate instructor) taught me something that became decisive to reach this point: The Japanese word "OSU". It means "to persevere whilst pushing oneself to the absolute limit".

He taught me that as long as we can stand up, we are not over, and that we must stand up, always, independently of the pain, or the challenge. There were very hard times during my PhD studies, many more than I expected. But I was not alone. Enrico Franconi and Diego Calvanese were two of the persons who cared, and asked, and sought for solutions, intensively. And it was Michael Kifer who, in the worse moment, helped me to stand up, taught me how to continue, backed me up. And as it was not enough, he went beyond science, and showed me how to think out of the box, not to judge too soon, not to listen too late.

Last but not least I would like to thank all the people here in Bolzano and USA who helped me to complete this journey: my local supervisor Werner Nutt, the reviewers of my thesis Marco Montali and Jose Alferes, and the members of the group, colleagues, and friends Mariano (The tool) Rodriguez, Vlad (it is ok) Ryzhikov , Elena (She knows) Botoeva, Babak Bagheri, Yazmin (the cook) Ibanez, Ola (Hola) Kerhet, Manuel (THE german) Kirschner, Inanc (Wing) Seylan, Tim (yellow) Knapik, Paul Fodor, Eddie (sensei) Cumming, Laura Alonso, Mario (Mochi) Assis, Alessandro Mosca, Maria Keet, Simon Razniewski, and the rest of the KRDB members.

Chapter 1

Introduction

> And the reason I am so nervous is that everything I do now is leading me to one of three possible futures... Which one will it be? Time alone will tell. But still I know that writing this diary can perhaps provide the answer; it may even help produce the right future.
>
> Adolfo Bioy Casares,
> The Invention of Morel

Artificial Intelligence (AI) is the area of computer science that studies the problem of how to make intelligent machines, especially intelligent computer programs. One of the key issues in this field is how to represent knowledge about a given domain, and how machines and systems (such as *autonomous agents*) can use this representation to make decisions, and infer new information. In this thesis we deal with one of the hardest aspects of the knowledge representation problem: *reasoning about the agents' actions.* To perform this task we need to reason about change in dynamic domains. Thus, we must be able to specify [65]:

Base scenario: object identities, static properties, space.

Time: time-varying properties, time itself.

Actions: preconditions of elementary actions, effects of elementary actions, nondeterministic actions, indirect effects of actions, and the cumulative effects of *complex actions.*

A complex action is an action composed out of simpler constituent actions, sometimes in rather complex ways. Let us present the following example to clarify the concepts introduced above :

1.0.1. EXAMPLE. [Block World] Suppose we need to model a robotic arm that can move a block from the top of one block, to the top of another if the tops of both blocks are clear and have the same color. The robotic arm can also be idle. If the top of any of the blocks is not free, then the robotic arm *recursively* clears the top of the blocks before moving them. The constituent actions of the complex action *move* are to *lift* and *drop* the blocks.

Once this information has been specified, we could consider a concrete setting with two violet blocks, blk_1 and blk_2 over a table and check if blk_2 can be stacked on the top of blk_1 (c.f. Figure 1.1). □

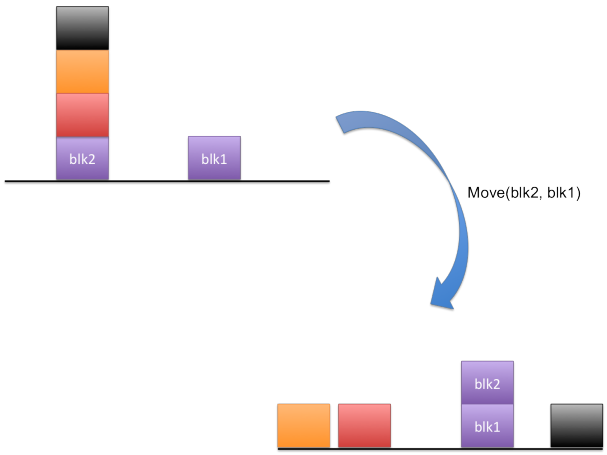

Figure 1.1: Block World

Now, let us point out several key issues that must be specified to check if blk_2 can be stacked on the top of blk_1.:

object identity: We must represent the fact that blk_1, blk_2, and the table are not the same object.

static properties: We must represent the color of the blocks.

space: We must be able to deal with locations. We must represent the knowledge that if blk_1 and blk_2 are in the table, then blk_2 is not on the top of blk_1.

time-varying properties: We should represent properties that change with time, like clearness of block tops.

time: We must represent the initial state where both blocks are in the table, and the successor states resulting from the actions taken by the robotic arm.

preconditions: We need to represent the restriction that the robotic arm can only lift a block if its top is free, and it can stack a block only if the blocks being stacked have the same color.

effects of elementary actions: We need to model that when the robotic arm lifts a block, such block is not on the table anymore.

nondeterministic actions: We should be able to express that the robotic arm can either move a block, or wait.

indirect effects: We must express that if a block breaks, it cannot be lifted anymore.

cumulative effects of complex actions: We should represent the *sequence* of action effects that define the complex action *move*.

Finally, to check if we can stack $blk2$ over $blk1$ we need to be able to *reason* over the initial state using our knowledge about the world and the effects of actions.

The standard and most promising tool for representing knowledge and reasoning is *logic*. Initially, it was first order logic and the situation calculus [61] was one of the first methodologies for reasoning about actions in that framework. However, many researchers felt that the situation calculus was a cumbersome and only partial solution to the problem. In particular, situation calculus did not offer an elegant way to work with *default reasoning* [65]. Default reasoning allows to infer properties of the world without a complete description of it. Some features of the world are *by default* assumed to be true or false. For instance, in our example if we do not have information about a block $blk3$, we assume that such block does not exist. That is, we jump to conclusions given the lack of information. If new information becomes available that invalidates those conclusions, then we must also retract those conclusions. Non-monotonic logics offered a simple and yet effective way to perform such kind of reasoning. Thus, a number of advanced logical theories for reasoning about actions and change based on non-monotonic formalisms were developed over the years. Some of the best known theories are: Fluent Calculus and Flux [76, 62], Event Calculus [51], $\mathcal{A}$[35], $\mathcal{L}_1$[8], $\mathcal{C}$[36], $\mathcal{ALM}$[46]. However, existing approaches have limitations such as the inability to define complex actions [35, 8, 36], performs complex hypothetical tests [56, 76, 62, 35, 8], post-conditions for actions, and recursive actions [35, 8, 36, 51] (further details can be found in Chapter 4). One of the non-obvious consequences of relying on a logic is the need for *axioms of inertia* (a.k.a. frame axioms). The issue here is that properties of an object

do not normally change without a cause (e.g. the color of a block remains the same after the block is moved), and this is not a "built-in" amenity in most logics. This problem has been called *the frame problem* [61]. Axioms of inertia are logical statements intended to solve this problem. Different logics solve the frame problem using different set of axioms, and this is a never-ending debate about the merits of the different solutions [14].

In this thesis we tackle the problem of developing an expressive declarative language that allows sophisticated and yet simple ways of describing and reasoning about actions and change, allowing:

- complex actions,
- recursion,
- post-conditions for actions,
- complex hypothetical tests,
- rich domain domain descriptions, etc.

As applications of our formalism, we show how it benefits the planning problem (c.f. Chapter 4) and also describe its use as a logic language that gives a declarative semantics to *production systems* and rule-based ontologies (c.f. Chapter 5). However, these two contributions, regarding planning and production systems, are to illustrate the power of our language and it is not the main focus of this thesis.

The language proposed in this thesis is based on *Transaction logic* ($\mathcal{TR}$). Transaction logic [1, 11, 12] is a promising logic language that overcomes many of the limitations of the existing approaches. It was intended as a formalism for declarative specification of complex state-changing transactions in logic programming; and it has been used for planning [11], knowledge representation [13], active databases [11], event processing [3], workflow management and Semantic Web services [23, 24, 73, 27], and as a declarative alternative to non-logical features in Prolog, like the *assert* and *retract* operators [12]. In particular, in the database area, $\mathcal{TR}$ serves as a declarative language for programming transactions, for defining active rules, and for updating database views. In addition, it is worth mentioning that $\mathcal{TR}$ has several implementations [43, 44, 75, 33, 82, 50].

The contributions provided in this thesis significantly advance the state of the art in the important subfield of *Artificial Intelligence* dedicated to reasoning about actions. We expect these results to find applications in intelligent agent systems, semantic Web services, question answering systems, and other areas.

Modeling Actions in Transaction Logic

The idea behind $\mathcal{TR}$ is that by defining a new logical connective for *sequencing* of actions and by giving it a model-theoretic semantics over sequences of states, one gets a purely logical formalism that combines declarative and procedural knowledge.

1.0.2. EXAMPLE. As a motivating example, consider the US health insurance regulations. The complexity of these laws makes it difficult to determine whether a particular action, like information disclosure, or contacting a patient, is compliant. To help along with this problem, [54] formalized a fragment of these regulations in Prolog, but could not formalize temporal, state-changing regulations. For instance, [54] had statements to express the fact that, to be compliant with the law, a DNA test requires a doctor's prescription and a patient's consent, but it was awkward to declaratively express the order in which these two independent actions are to be performed. The sequencing operator of $\mathcal{TR}$ enables these kinds of statements naturally. However, it has limitations to reason about the effects of these actions. □

Although $\mathcal{TR}$ was created to program state-changing transaction, [10] demonstrated that $\mathcal{TR}$ can do basic, yet interesting reasoning about actions. However, [10] was unable to develop a complete proof theory, and the fragment of $\mathcal{TR}$ studied there was not expressive enough for modeling many problems in the context of action languages, for instance the Turkey Hunting problem [40]. A number of sophisticated logical theories to reason about actions have been developed over the years, including $\mathcal{A}$[35], $\mathcal{L}_1$[8], $\mathcal{C}$[36], $\mathcal{ALM}$[46]. Unfortunately, most of such languages have their weak points along with the strong ones, and none of them is sufficient as a logical foundation for agents.

In this thesis we develop a full-fledged theory—depicted in Figure 1.2—*Transaction Logic with Partially Defined Actions* ($\mathcal{TR}^{PAD}$), for programming and reasoning about actions over states. This theory extends $\mathcal{TR}$ with *premise*-formulas, which generalize $\mathcal{TR}$'s data and transition oracles, making the formalism more suitable for specifying partial knowledge about actions. The data oracles in $\mathcal{TR}$ specifies a set of primitive database *queries* , i.e., the static semantics of states; and the transition oracle specifies a set of primitive database *updates*, i.e., the dynamic semantics of states.

In our example, we use $\mathcal{TR}^{PAD}$ premises to express information about the states. For instance, to express that in a state $\mathbf{D}_2$ the patient consents to a DNA test or that executing the action `do_dna` in state $\mathbf{D}_1$ leads to state $\mathbf{D}_2$.

It is worth noticing that $\mathcal{TR}^{PAD}$ subsumes Horn-$\mathcal{TR}$ [1, 11, 12], a well-studied and expressive fragment of Transaction Logic. The theory developed here, $\mathcal{TR}^{PAD}$, has a great deal of sophistication in action composition, enabling

hypothetical, recursive, and non-deterministic actions. This allows to reason about actions in highly complex scenarios, and not only about the effects of future actions, but also about the causes and preconditions holding in the past. For instance, coming back to our example, in $\mathcal{TR}^{PAD}$ we can program an action "*do_dna*" that performs a DNA test if the patient gives an ok, but (assuming that the hospital was in compliance) if the test was administered we can also infer that the patient must have given her prior consent.

To carry out this kind of reasoning, we provide a sound and complete proof system for this new formalism. We also show that, when we restrict the shape of partial actions, such reasoning can be done by a reduction to ordinary logic programming. This last contribution provides an easy way to implement and experiment with the formalism, although a better implementation should be using the proof theory directly, similarly to the implementation of the serial-Horn subset of *TR* in FLORA-2 [50].

Finally, to boost the non-monotonic capabilities of $\mathcal{TR}^{PAD}$, we extend $\mathcal{TR}^{PAD}$ with *default* negation (a.k.a. negation as failure). Default negation allows a logic system to conclude the negation of any atom that the system unsuccessfully finishes exploring all possible proofs.

Our main focus in this thesis is the development of the formalism itself and illustration of its capabilities.

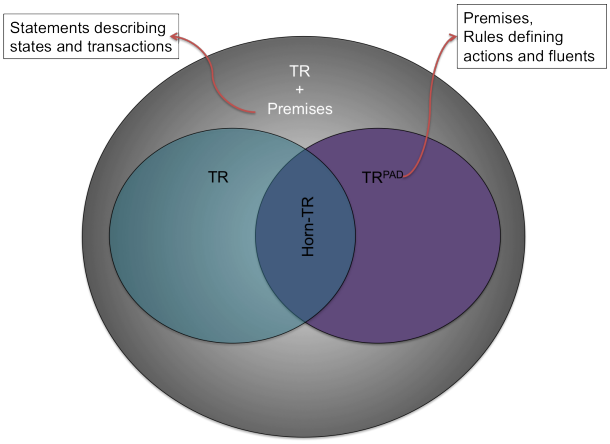

Figure 1.2: $\mathcal{TR}^{PAD}$

Applying $\mathcal{TR}^{PAD}$

As an application, Chapter 5 develops a semantics for *production systems* augmented with DL Ontologies, and embed it in $\mathcal{TR}^{PAD}$.

Production systems (PSs) are one of the oldest knowledge representation paradigms in artificial intelligence, and are still widely used today. They are applied to examine the health of data centers and networks [74], to enforce constraints over databases [15], to test product's quality [20], accounting [49], to model human behavior [84], etc.

Such systems consist of a set of *production rules* that rely on forward chaining reasoning to update the underlying database, called *working memory*. Traditionally, PSs have had only operational semantics, where satisfaction of rule conditions is checked using pattern matching, and rule actions produce assertions and deletions of facts from the working memory. PSs syntax and semantics have been standardized as W3C's Production Rule Dialect of the Rule Interchange Format (RIF-PRD) [80]. The RIF-PRD specification has a number of limitations, however. First, it omits certain important primitives that are found in many commercial production systems such as IBM's *JRules* [49]. The *FOR*-loop and the *while*-loop constructs are examples of such an omission. Second, RIF-PRD still does not integrate with ontologies [6, 42]. Here, by ontology we mean a formal representation of a domain of interest, expressed in terms of concepts and roles, which denote classes of objects and binary relations between classes of objects, respectively.

The integration of two knowledge representation languages with such different semantics raises many questions. In particular, while a working memory can be seen as a single interpretation structure (closed world semantics), an ontology represents a possibly infinite set of interpretations (open world semantics), namely the models of the ontology. Thus, it is necessary to define how to apply rules to this set of models, and how we check for satisfaction/entailment of rule conditions. On top of that, the rule applications can produce inconsistencies between the working memory and the ontology that need to be solved.

To answer these questions we need to define a precise semantics (both model-theoretic and computational) to the combination of rules, ontologies, and production systems.

We provide both, *(i)* a new semantics for production systems augmented with *DL* ontologies that includes looping-rules, and can handle inconsistency; *(ii)* a sound embedding of the combination of PSs and *rule-based* ontologies into $\mathcal{TR}^{PAD}$, which provides a model-theoretic semantics to the combination.

Our formalization is significantly more general than RIF-PRD or other existing formalizations of production rules in that it supports wider ontology integration and covers important extensions that exist in commercial systems such as the aforesaid *FOR*-loop.

Summary of the Contributions

Our contribution in this thesis are as follows:

1. extension of $\mathcal{TR}$ with *premise*-formulas to express information about the states, making the formalism more suitable for specifying partial knowledge about actions;
2. defining a subset of the formalism, called $\mathcal{TR}^{PAD}$, and demonstrating its expressive power for high-level descriptions of the behavior of complex actions;
3. development of a sound and complete proof theory for $\mathcal{TR}^{PAD}$;
4. sound and complete reduction of the serial-Horn fragment of $\mathcal{TR}$ and the deterministic subset of $\mathcal{TR}^{PAD}$ to regular logic programming;
5. extension of $\mathcal{TR}^{PAD}$ with default negation, and definition of a well-founded semantics [79] for $\mathcal{TR}^{PAD}$,
6. relationship of the modeling and reasoning capabilities of $\mathcal{TR}^{PAD}$ and the action language $\mathcal{L}_1$, and a brief discussion about the relation between $\mathcal{TR}^{PAD}$ and the action languages Situation Calculus, Event Calculus, Fluent Calculus, $\mathcal{C}$ and $\mathcal{ALM}$;
7. new operational semantics for production systems augmented with *DL* ontologies;
8. sound embedding of such combination (restricted to *rule-based* ontologies) into $\mathcal{TR}^{PAD}$, giving in this way a model-theoretic semantics to the combination.

Structure of the Thesis

This thesis is organized as follows.

Chapter 2 presents the necessary background needed for this thesis. That is, Transaction Logic needed to define $\mathcal{TR}^{PAD}$, First-Order Logic, and Logic Programming needed in Chapter 4.

Chapter 3 defines an appropriate subset of $\mathcal{TR}$, $\mathcal{TR}^{PAD}$, develops a sound and complete proof theory for it, and provides numerous examples of the use of $\mathcal{TR}^{PAD}$ and its proof theory for complex reasoning tasks about actions. It also proves that the frame axioms proposed in that chapter *behave as expected*. That is, they correctly model the inertia laws. In

addition, it introduces a reduction from a fragment of $\mathcal{TR}^{PAD}$ to Horn logic programs and presents soundness and completeness results for this reduction. Finally, it extends $\mathcal{TR}^{PAD}$ with default negation and shows the relation between the frame axioms with and without default negation.

Chapter 4 identifies and compares the modeling and reasoning capabilities of $\mathcal{TR}^{PAD}$ and the action language $\mathcal{L}_1$, reduces $\mathcal{L}_1$ to $\mathcal{TR}^{PAD}$, discusses planning problems in both formalisms, and compares $\mathcal{TR}^{PAD}$ with other popular action languages: Situation Calculus, Event Calculus, $\mathcal{ALM}$, $\mathcal{C}$, etc.

Chapter 5 presents the necessary background on production systems and Description Logic needed for this Chapter. It explores the space of design options for combining the traditional closed world semantics of PSs with the open world semantics of DL and propose a *new* operational semantics for such combination. Finally, it formalizes in $\mathcal{TR}^{PAD}$ the semantics for Production systems combined with rule-based ontologies.

Chapter 6 concludes the thesis.

All proofs are given in the appendices.

Chapter 2

Preliminaries

> My undertaking is not difficult, essentially. I should only have to be immortal to carry it out.
>
> Jorge Luis Borges, Pierre Menard, Author of The Quixote

2.1 First-Order Logic

The alphabet of a first-order language $\mathcal{L}$ includes a countably infinite disjoint sets of variables $\mathcal{V}$, constant symbols $\mathcal{C}$, and predicate symbols $\mathcal{P}$. A *term* is a constant or a variable. Each predicate symbols has an *arity* n, which is a non-negative integer. In this thesis we will use First-Order logic (FOL) (and Description Logic) in Chapter 5–where we combine productions systems and ontologies—as an ontology language. Thus, we will not include function symbols here since neither description logics nor production systems use them.[1]

Our language, $\mathcal{L}$, includes all the usual first-order operators $\vee, \wedge, \neg, \forall, \exists, \rightarrow, =$ plus the symbol **neg** , which represents the *explicit negation* (also sometimes called *strong* negation) [66]. The **neg** symbol applies only to atoms. In the actual use in this thesis, $\neg$ will appear only in ontologies, while **neg** will be used both in ontologies and in Transaction Logic. In a certain sense, which will be made clear later, **neg** f will imply $\neg f$, but not vice versa.

[1] Many production systems do use built-in and external functions and so do some DLs. However, this does not bring any new or interesting issues in our context, so we disregard functions (in FOL) in order to simplify the exposition.

Formulas are defined recursively as usual in first-order logic. A literal is either an atomic formula f, or a formula of the form **neg** f where f is an atomic formula. Atoms are also called **positive literals**. A **negative literal** is an atom preceded with the symbol **neg** (e.g., **neg** $p(X)$).

Since in Chapter 5 we will be integrating ontologies with production systems, we will need to use Herbrand domains and the unique name assumption. Therefore, we will use the *Herbrand semantics* from the outset. As is well-known, this semantics is equivalent to the general one for universal clausal form [59]. The semantics defines *semantic structures.* The domain of a Herbrand semantic structure is called the Herbrand universe $\mathcal{U}$; in our restricted case it is just the set of all constants $\mathcal{C}$ in the language $\mathcal{L}$. The Herbrand base $\mathcal{B}$ is a set of all ground literals in the language. Note that the Herbrand universe and Herbrand base are infinite, fixed, and depend only on the language $\mathcal{L}$.

2.1.1. Definition. [Semantic Structure] A *semantic structure* $\mathcal{M}$ is a triple $\langle \Delta, \mathbf{B}, \sigma \rangle$, where

- Δ is the Herbrand universe.
- $\mathbf{B}$ is a subset of the Herbrand Base $\mathcal{B}$.
- σ is a **variable assignment**, i.e., a mapping $\mathcal{V} \longrightarrow \Delta$. □

We now define satisfaction of formulas by semantic structures. If ϕ is a formula of $\mathcal{L}$, $\mathcal{M}$ is a structure for $\mathcal{L}$, then satisfaction of ϕ in $\mathcal{M}$ is denoted $\mathcal{M} \models \phi$. Given a structure $\mathcal{M}$, and a term (i.e., constant or variable) t, we define $t^{\mathcal{M}}$ as: $t^{\mathcal{M}} = \sigma(t)$ if t is a variable and $t^{\mathcal{M}} = t$ if t is a constant. Note that this implies the *unique name assumption* (UNA). That is, if $c_1, c_2 \in \mathcal{C}$ are two distinct constants then $c_1^{\mathcal{M}} \neq c_2^{\mathcal{M}}$.

The relation $\models$ is defined recursively as follows:

- If t_1 and t_2 are terms, then $\mathcal{M} \models t_1 = t_2$ if and only if $t_1^{\mathcal{M}} = t_2^{\mathcal{M}}$.
- If P is an n-place predicate letter in $\mathcal{L}$ and $t_1, \ldots, t_n$ are terms, then $\mathcal{M} \models P(t_1 \ldots t_n)$ if and only if $P(t_1^{\mathcal{M}} \ldots t_n^{\mathcal{M}}) \in \mathbf{B}$.
- If P is an n-place predicate letter in $\mathcal{P}$ and $t_1, \ldots, t_n$ are terms, then $\mathcal{M} \models$ **neg** $P(t_1 \ldots t_n)$ if and only if **neg** $P(t_1^{\mathcal{M}} \ldots t_n^{\mathcal{M}}) \in \mathbf{B}$.
- $\mathcal{M} \models \neg\phi$ if and only if it is not the case that $\mathcal{M} \models \phi$.
- $\mathcal{M} \models (\phi \wedge \psi)$ if and only if $\mathcal{M} \models \phi$ and $\mathcal{M} \models \psi$.
- $\mathcal{M} \models (\phi \vee \psi)$ if and only if $\mathcal{M} \models \phi$ or $\mathcal{M} \models \psi$.

- $\mathcal{M} \models \forall v : \phi$ if and only if $\mathcal{M}' \models \phi$ for every structure $\mathcal{M}'$ that agrees with $\mathcal{M}$ except possibly on the variable v.
- $\mathcal{M} \models \exists v : \phi$ if and only if $\mathcal{M}' \models \phi$ for some structure $\mathcal{M}'$ that agrees with $\mathcal{M}$ except possibly on the variable v.

A formula ϕ is *valid*, if $\mathcal{M} \models \phi$, for every structure $\mathcal{M}$.
A formula ϕ is *satisfiable* if there is a structure $\mathcal{M}$ such that $\mathcal{M} \models \phi$.
If Γ is a set of sentences and if $\mathcal{M} \models \phi$ for each sentence ϕ in Γ, then we say that $\mathcal{M}$ is a *model* of Γ. So a set of sentences is satisfiable if it has a model.
We say that Γ *entails* ϕ, written $\Gamma \models \phi$, if and only if $\Gamma \cup \{\neg\phi\}$ is not satisfiable.

2.2 Logic Programs

In this section we briefly remind the basic notions from standard logic programming [60], which will be needed in this thesis.

Syntax.

The language $\mathcal{L}$ in traditional logic programming consists of:

- A countably infinite set of variables $\mathcal{V}$.
- A countably infinite set of function symbols $\mathcal{F}$, where constants are treated as 0-arity function symbols.
- A countably infinite set of predicates $\mathcal{P}$.
- The symbols $\{\forall, \exists, \wedge, \rightarrow, \mathbf{neg}, \mathbf{not}\}$

Terms are defined as usual in first order logic. We denote the set of all atoms in the language as $\mathbf{A}$. Atoms will be also called positive literals.
The symbol **neg** will be used to represent the strong negation [66]. The symbol **not** will be used for *default* negation.
A literal is either an atom or it has one of the following negated forms:

$$\mathbf{neg}\, f, \quad \mathbf{not}\, f, \ \mathbf{not}\, \mathbf{neg}\, f$$

where f is an atom. Literals that do not mention **not** are said to be **not**-free. Otherwise we say they are **not**-literals.
A ***logic program*** is a collection of statements (called ***rules***) of the form

$$\forall X : (l_0 \leftarrow l_1 \wedge \cdots \wedge l_m \wedge \mathbf{not}\, l_{m+1} \wedge \ldots \wedge \mathbf{not}\, l_n) \tag{2.1}$$

where each l_i is a literal and l_0 is **not**-free. The literal l_0 is called the ***head*** of the rule r. The set of literals $\{l_1, \ldots, l_n\}$ is called the ***body*** of r. If the body is empty, then $\leftarrow$ can be dropped, and the it is a *fact*. By a ***clause*** we mean either a rule or a fact.

Given a rule r of the form (2.1), the sets $\{l_0\}$, $\{l_1 \ldots l_m\}$, and $\{l_{m+1} \ldots l_n\}$ are referred to as $head(r)$, $pos(r)$ and $neg(r)$ respectively. The set $lit(t)$ stands for $head(r) \cup pos(r) \cup neg(r)$.

Following a standard convention, (2.1) will be simply written as:

$$l_0 \leftarrow l_1, \ldots, l_m, \mathbf{not}\, l_{m+1}, \ldots, \mathbf{not}\, l_n) \tag{2.2}$$

An expression is called *ground* if it does not contain any variable.

Queries are statements of the form $\exists \bar{X} : l_1 \wedge ... \wedge l_m$, where $l_1, ..., l_m$ are literals and $\bar{X}$ are all the variables mentioned in $l_1, ..., l_m$. The existential quantifier is usually omitted and comma is used often in lieu of the conjunction symbol $\wedge$.

A ***Horn logic program*** is a logic program that do not contain **not**-literals. Analogously, a ***Horn query*** is a query that do not contain **not**-literals.

Semantics.

This Section reviews the main concepts of the *well-founded semantics* for logic programs. We will follow the approach in [68]. However we will modify the presentation to ease the understanding of the well-founded semantics for $\mathcal{TR}^{PAD}$.

First let us introduce some notation. Let $\mathbf{P}$ be a logic program. The domain of $\mathbf{P}$ is the Herbrand universe $\mathcal{U}$ of $\mathcal{L}$. The Herbrand base of $\mathbf{P}$, denoted $\mathcal{B}_P$,[2] is the set of all instantiations of atoms in $\mathbf{P}$ using the terms from $\mathcal{U}$.

This semantics uses three truth values, $\mathbf{u}$, $\mathbf{t}$ and $\mathbf{f}$, which stand for *true*, *false*, and *undefined* and are ordered as follows: $\mathbf{f} < \mathbf{u} < \mathbf{t}$. In addition, we will use the following operator $\sim$: $\sim \mathbf{t} = \mathbf{f}$, $\sim \mathbf{f} = \mathbf{t}$, $\sim \mathbf{u} = \mathbf{u}$.

2.2.1. Definition. [Partial Herbrand interpretation] A ***Partial Herbrand interpretation*** is mapping from $\mathcal{B} \mapsto \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ that assigns a truth value, $\mathbf{f}, \mathbf{u}$, or $\mathbf{t}$, to every formula ϕ in $\mathcal{B}$. □

2.2.2. Definition. [Satisfaction] Let $\mathbf{I}$ be a partial Herbrand interpretation, f a ground **not**-free literal, and $l_1 \ldots l_n$ literals. We define **truth valuations** with respect to the path structure $\mathbf{I}$ as follows:

- $\mathbf{I}(f)$ was already defined as part of the definition of partial Herbrand interpretation.
- $\mathbf{I}(l_1 \wedge l_2) = min(\mathbf{I}(l_1), \mathbf{I}(l_2))$

[2] We will omit the subindex when it is clear from the context.

- $\mathbf{I}(\mathbf{not}\,\phi) = \sim \mathbf{I}(\phi)$
- $\mathbf{I}(f \leftarrow l_1 \wedge \cdots \wedge l_n) = \mathbf{t}$ iff $\mathbf{I}(f) \geq \mathbf{I}(l_1 \wedge \cdots \wedge l_n)$

We write $\mathbf{I} \models \phi$ and say that ϕ is **satisfied** in the structure $\mathbf{I}$ if $\mathbf{I}(\phi) = \mathbf{t}$. □

2.2.3. Definition. [Model of $\mathbf{P}$] A partial interpretation $\mathbf{I}$ is a model of a program $\mathbf{P}$ if and only if for all ground instances of the form (2.2) in $\mathbf{P}$

$$\mathbf{I}(l_0) \geq \mathbf{min}\{\mathbf{I}(l_1), \ldots, \mathbf{I}(l_k)\}$$

□

There are *two natural orderings* between interpretations, one of them, $\preceq^c$, is called the *truth ordering*. The other one, $\leq^c$ is called the *information ordering* and coincides with set-theoretic inclusion.

Given two *partial* Herbrand interpretations $\mathbf{N}_1$ and $\mathbf{N}_2$, we say that

- $\mathbf{N}_1 \leq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$.
- $\mathbf{N}_1 \preceq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_2$ are true in $\mathbf{N}_1$.

2.2.4. Definition. [Least Model] A model $\mathbf{M}$ of a program $\mathbf{P}$ is **minimal** with respect to $\preceq^c$ iff for any other model, $\mathbf{N}$, of $\mathbf{P}$, if $\mathbf{N} \preceq^c \mathbf{M}$ then $\mathbf{N} = \mathbf{M}$. The **least** model of $\mathbf{P}$, denoted LPM($\mathbf{P}$), is a minimal model that is unique. □

The following definition is key to the notion of well-founded models.

2.2.5. Definition. [Quotient Operator] Let $\mathbf{P}$ be a logic program, and $\mathbf{I}$ be any partial interpretation. By the **quotient** of $\mathbf{P}$ **modulo** $\mathbf{I}$ we mean a new program, $\frac{\mathbf{P}}{\mathbf{I}}$, which is obtained from $\mathbf{P}$ by replacing in every clause of $\mathbf{P}$ all *negative* premises **not** l which are true (respectively undefined; respectively false) in $\mathbf{I}$ by their corresponding truth value $\mathbf{t}$ (respectively, $\mathbf{u}$, $\mathbf{f}$). □

2.2.6. Proposition. *[69] Let $\mathbf{P}$ be a logic program, and $\mathbf{I}$ be any partial interpretation. Then $\frac{\mathbf{P}}{\mathbf{I}}$ is* **not***-free and therefore it has a unique least partial model.*

The least partial model of a program can be obtained as the least fixed point of the immediate consequence operator $\hat{T}$, defined below. The following definition assumes that the program has already been grounded.

2.2.7. Definition. [Consequence Operator][69] Let $\mathbf{P}$ be a logic program and let $\mathbf{I}$ be an interpretation. We define $\hat{T}(\mathbf{I})$ as follows:

$$\hat{T}(\mathbf{I}) = \mathrm{LPM}(\frac{\mathbf{P}}{\mathbf{I}})$$

The ordinal powers of the consequence operator $\hat{T}$ are then defined inductively as follows:

- $\hat{T}^{\uparrow 0}(\mathbf{I}_\emptyset) = \mathbf{I}_\emptyset$
- $\hat{T}^{\uparrow n}(\mathbf{I}_\emptyset) = \hat{T}(\hat{T}^{\uparrow n-1}(\mathbf{I}_\emptyset))$, if n is a successor ordinal
- $\hat{T}^{\uparrow \omega}(\mathbf{I}_\emptyset) = \bigcup_{j \leq \omega} \hat{T}^{\uparrow j}(\mathbf{I}_\emptyset)$, if ω is a limit ordinal □

2.2.8. Proposition. *The sequence of interpretations $\hat{T}^{\uparrow n}$ described in Definition 2.2.7 is monotonic and thus has a fixed point $\hat{T}^{\uparrow n}$ with the property that*

$$\hat{T}^{\uparrow n} = \hat{T}^{\uparrow n+1}$$

Next we give the constructive definition of the well-founded models of logic programs as iterated fixed points of the consequence operator $\hat{T}$.

2.2.9. Definition. [Well-founded] The **well-founded** model of a logic program $\mathbf{P}$, written WFM($\mathbf{P}$), is defined as the limit of the sequence of interpretations $\hat{T}^{\uparrow n}$ described in Definition 2.2.7 □

2.2.10. Theorem. *[69] $WFM(\mathbf{P})$ is the least model of $\mathbf{P}$.*

Let $\mathbf{P}$ be a program and q a query. For simplicity we assume that q is a ground formula. We say that the program $\mathbf{P}$'s answer to q is *yes* if $WFM(\mathbf{P})(q) = \mathbf{t}$, *no* if $WFM(\mathbf{P})(\mathbf{not}\, q) = \mathbf{f}$, and *unknown* otherwise.

Stable model semantics for Logic Programs

In this section, we review the main concepts of *stable model semantics* [34] needed in Chapter 4. For the sake of simplicity, we assume that logic rules are written as

$$l_0 \leftarrow l_1, \ldots, l_n, \mathbf{not}\, l_{n+1}, \ldots, \mathbf{not}\, l_k, \tag{2.3}$$

where $l_1 \ldots l_k$ are **not**-free.

2.2.11. Definition. [Herbrand interpretation] A ***Herbrand interpretation***, $\mathcal{M}$, is consistent a subset of the Herbrand base. □

Observe that under stable model semantics, interpretations are 2-valued. Satisfaction of a formula ϕ by Herbrand interpretation, $\mathcal{M}$, denoted $\mathcal{M} \models \phi$, is defined as follows:

- $\mathcal{M} \models l$, where l is a (**not**-free) literal, iff $l \in \mathcal{M}$.
- $\mathcal{M} \models \phi_1 \wedge \phi_2$, iff $\mathcal{M} \models \phi_1$ and $\mathcal{M} \models \phi_2$.
- $\mathcal{M} \models \mathbf{not}\, \phi$, iff it is not the case that $\mathcal{M} \models \phi$.
- $\mathcal{M} \models r$, where r is a ground rule of the form (2.3), iff $l_0 \in \mathcal{M}$ whenever $\mathcal{M} \models l_1 \wedge \ldots l_n$ and $\mathcal{M} \models \mathbf{not}\, (l_{n+1} \wedge \ldots l_k)$.

Given a **not**-free program $\mathbf{P}$, we write $\mathcal{M} \models \mathbf{P}$ if $\mathcal{M} \models r$ for every rule $r \in \mathbf{P}$. In this case we say that $\mathcal{M}$ is a ***stable model*** (a.k.a. answer set) of $\mathbf{P}$. It is known that every **not**-free program $\mathbf{P}$ has a unique ***least model*** [4]—a model $\mathcal{M}_0$ such that for any other model $\mathbf{N}$ of $\mathbf{P}$, $l \in \mathcal{M}_0$ implies $l \in \mathbf{N}$ for any $l \in \mathcal{B}_{\mathbf{P}}$.

To extend the definition of stable model (answer set) to arbitrary programs, take any program $\mathbf{P}$, and let $\mathcal{I}$ be a Herbrand interpretation in $\mathcal{L}$. The **reduct**, $\mathbf{P}(\mathcal{S})$, of $\mathbf{P}$ relative to $\mathcal{S}$ is obtained from $\mathbf{P}$ by first dropping every rule of the form (2.3) such that $\{l_{n+1}, \ldots, l_k\} \cap \mathcal{I} = \emptyset$; and then dropping the $\{l_{n+1}, \ldots, l_k\}$ literals from the bodies of all remaining rules. Thus $\mathbf{P}(\mathcal{I})$ is a program without default negation.

2.2.12. Definition. [Stable Model] A Herbrand interpretation $\mathcal{M}$ is an **stable model** for $\mathbf{P}$ if $\mathcal{M}$ is an answer set for $\mathbf{P}(\mathcal{I})$. □

Observe that no every program has stable models, for instance

$$p \leftarrow \mathbf{not}\, p$$

has no stable models.

2.2.13. Definition. [Entailment] A program $\mathbf{P}$ entails a ground literal l, written $\mathbf{P} \models l$, if l is satised by every stable model of $\mathbf{P}$. □

Let $\mathbf{P}$ be a program and q is a query. For simplicity we assume that q is a **not**-free literal. We say that the program $\mathbf{P}$'s answer to q is *yes* if $\mathbf{P} \models q$, *no* if $\mathbf{P} \models \mathbf{not}\, q$, and *unknown* otherwise.

Splitting Sets: In the following, we will work with *stratified* programs. Intuitively, a program $\mathbf{P}$ is stratified if it can be partitioned into (disjoint) strata $P_1 \ldots P_n$ such that: *(i)* $\mathbf{P} = P_1 \cup \cdots \cup P_n$, *(ii)* P_0 is **not**-free, and *(iii)* all the negative literals in P_i $(0 < i \leq n)$ are only allowed to refer to predicates that

are already defined in P_{i-1}. Intuitively, in a *stratified* program $\mathbf{P}$, the *intended model* is obtained via a sequence of bottom-up derivation steps. In the first step, $\mathbf{P}$ is split into stratums. The first stratum is a bottom part that does not contain negation as failure. Since this subprogram is positive, it has a unique stable model. Having substituted the values of the bottom predicates in the bodies of the remaining rules, $\mathbf{P}$ is reduced to a program with fewer strata. By applying the splitting step several times, and computing every time the unique stable model of a positive bottom, we will arrive at the *intended model* of $\mathbf{P}$. Further details can be found in [67].

2.2.14. Definition. [Splitting Set [57]] A **splitting set** for a program $\mathbf{P}$ is any set U of literals such that for every rule $r \in \mathbf{P}$, if $head(r) \cap U \neq \emptyset$ then $lit(r) \subset U$. If U is a splitting set for $\mathbf{P}$, we also say that U **splits** P. The set of rules $r \in \mathbf{P}$ such that $lit(r) \subset U$ is called the **bottom** of $\mathbf{P}$ relative to the splitting set U and denoted by $b_U(\mathbf{P})$. The subprogram $\mathbf{P} \setminus b_U(\mathbf{P})$ is called the **top** of $\mathbf{P}$ relative to U. □

2.2.15. Definition. [Partial Evaluation] The **partial evaluation** of a program $\mathbf{P}$ with splitting set U with respect to a set of literals X, is the program $e_U(\mathbf{P}, X)$ defined as follows. For each rule $r \in \mathbf{P}$ such that

$$(pos(r) \cap U) \subset X \quad \text{and} \quad (neg(r) \cap U) \cap X = \emptyset$$

put in $e_U(\mathbf{P}, X)$ all the rules r' that satisfy the following property

$$\begin{array}{lcl} head(r') & = & head(r) \\ pos(r') & = & pos(r) \setminus U \\ neg(r') & = & neg(r) \setminus U \end{array}$$

□

2.2.16. Definition. [Solution] Let U be a splitting set for a program $\mathbf{P}$. A **solution** to $\mathbf{P}$ with respect to U is a pair (X, Y) of literals such that

- X is an stable model for $b_U(\mathbf{P})$
- Y is an stable model for $e_U(\mathbf{P} \setminus b_U(\mathbf{P}), X)$
- $X \cup Y$ is consistent. □

2.2.17. Example. [8] Consider the following program $\mathbf{P}$:

$$\begin{array}{l} a \leftarrow b, \mathbf{not}\, c \\ b \leftarrow c, \mathbf{not}\, a \\ c \leftarrow \end{array}$$

The set $U = \{c\}$ splits $\mathbf{P}$; the last rule of $\mathbf{P}$ belongs to the bottom and the first two rules from the top. Clearly, the unique stable model for the bottom of $\mathbf{P}$ is $\{c\}$. The partial evaluation of the top part of $\mathbf{P}$ consists in dropping its first rule, because the negated subgoal c makes it useless, and in dropping the trivial positive subgoal c in the second rule. The result of simplification is the program consisting of one rule

$$b \leftarrow \mathbf{not}\, a \tag{2.4}$$

The only stable model for P can be obtained by adding the only stable model for (2.4), which is $\{b\}$, to the stable model for the bottom used in the evaluation process, $\{c\}$. □

2.2.18. PROPOSITION. *[57] Let U be a splitting set for a program $\mathbf{P}$. A set S of literals is a consistent stable model for $\mathbf{P}$ if and only if $S = X \cup Y$ for some solution (X, Y) of $\mathbf{P}$ with respect to U.*

2.3 Transaction Logic

This section briefly reviews the syntax and model theory of a subset of Transaction Logic, which we call $\mathcal{TR}^-$, that is necessary for understanding the results of this thesis. One restriction on $\mathcal{TR}^-$ are that it uses only the *explicit* negation **neg** (sometimes called strong negation [66]). This negation is a weaker form of classical negation, and it applies only to fluents, not actions. Another restriction is that $\mathcal{TR}^-$ uses only one particular type of database states and update operators, known as the "*relational oracle*". In Sections 3.5 and 3.6, these restrictions will enable us to reduce various subsets of interest of $\mathcal{TR}^-$ to ordinary logic programming.

Syntax.

The alphabet of a language, $\mathcal{L}_{\mathcal{TR}}$, of $\mathcal{TR}^-$ consists of:

- A countably infinite set of variables $\mathcal{V}$.
- A countably infinite set of function symbols $\mathcal{F}$, where constants are treated as 0-arity function symbols.
- A countably infinite set of predicates $\mathcal{P}$. This set is further partitioned into two countably infinite subsets, $\mathcal{P}_{fluents}$ and $\mathcal{P}_{actions}$. The former will be used to represent facts in database states and the latter transactions that change those states. Querying a fluent can be viewed as an action that does not change the underlying database state.

- Logical connectives $\wedge$, $\vee$, the implication connectives $\leftarrow$ and $\rightarrow$, sequential conjunction $\otimes$, and hypothetical operator $\Diamond$.
- The explicit negation connective **neg**.
- Quantifiers $\forall$ and $\exists$.

Terms are defined as usual in first-order logic. Formulas in Transaction Logic are called *transaction formulas*; they extend the syntax of first-order logic as defined next.
A ***transaction formula*** is defined recursively as follows.

- An ***atomic formula*** is an expression of the form $p(t_1 \ldots t_n)$ where $p \in \mathcal{P}$ is a predicate symbol, and $t_1, \ldots, t_n$ are terms. Atoms are also called ***positive literals***. A ***negative literal*** is an atom preceded with the symbol **neg**. A literal whose predicate symbol is in $\mathcal{P}_{fluents}$ will be referred to as a ***fluent literal***. If the predicate symbol is in $\mathcal{P}_{action}$, the literal will be called ***transactional*** (or ***action***) ***literal***. Action literals are always positive and *cannot* be preceded with **neg** (but fluent literals can).
- If ϕ and ψ are transaction formulas then so are the following expressions:
  - *"Classical" conjunction and disjunction*: $\phi \wedge \psi$, $\phi \vee \psi$
  - *Left and right implication*: $\phi \leftarrow \psi$, $\psi \rightarrow \phi$
  - *Serial conjunction*: $\phi \otimes \psi$
  - *Hypothetical execution*: $\Diamond\phi$
  - *Quantification*: $\forall X\phi, \exists X\psi$, where X is a variable.
- A ***transaction*** is a statement of the form $\texttt{?-}\,\exists\bar{X}\phi$, where $\phi = l_1 \otimes \cdots \otimes l_k$ is a serial conjunction of literals (both fluent and action literals) and $\bar{X}$ is the list of all the variables that occur in ϕ.

  Transactions in $\mathcal{TR}^-$ are analogous to (and generalize) the notion of queries in ordinary logic programming.
- A ***transaction base*** is a set of transaction formulas.

Informally, a serial conjunction of the form $\phi \otimes \psi$ is an action composed of an execution of ϕ followed by an execution of ψ. When ϕ and ψ are conjunctions of fluents, the serial and classical conjunctions behave identically, i.e.,

$$f^1 \wedge \ldots \wedge f^n \equiv f^1 \otimes \ldots \otimes f^n$$

We will also be often using the usual De Morgan's laws, such as $\mathbf{neg}\,\mathbf{neg}\,f \equiv f$ and $\mathbf{neg}\,(f \wedge g) \equiv \mathbf{neg}\,f \vee \mathbf{neg}\,g$. This allows us to apply **neg** to complex formulas and not just the atomic ones. For example,

$$\begin{array}{l}\mathbf{neg}\,(f_1 \wedge f_2) \equiv \mathbf{neg}\,f_1 \vee \mathbf{neg}\,f_2\\ \mathbf{neg}\,(f_1 \vee f_2) \equiv \mathbf{neg}\,f_1 \wedge \mathbf{neg}\,f_2\\ \mathbf{neg}\,(\mathbf{neg}\,f_1 \vee \mathbf{neg}\,f_2) \equiv f_1 \wedge f_2\end{array}$$

A hypothetical formula, $\Diamond\phi$, represents an action where ϕ is tested *hypothetically* whether it can be executed at the current state. However, no actual changes to the current state takes place. Implications, that can be written $\phi \leftarrow \psi$ and $\psi \rightarrow \phi$, can be understood as statements that ϕ is a call to a *complex action* (a.k.a. compound actions or complex transaction) and ψ is the definition of (the actual course of action for) that transaction. In Section 3.1 we will see another use of the implication $\rightarrow$ for partial action definitions. We assume that the set of all fluent predicates is partitioned into ***base fluents*** and ***derived fluents***. Base fluents can appear only as facts, while derived fluents can appear in the heads of fluent rules, but they cannot appear as facts. In addition to the user defined predicate symbols, in $\mathcal{TR}^-$ there are built-in actions called elementary transitions for basic manipulation of states. These include $delete(f)$ and $insert(f)$ for every ground base fluent literal f. Intuitively, *delete* and *insert* deletes and inserts facts respectively.

The following examples illustrates the above concepts. We will follow the usual logic programming convention whereby lowercase symbols represent constants, function, and predicate symbols, and the uppercase symbols represent variables that are universally quantified outside of the rules. Universal quantifiers are omitted, as usual.

2.3.1. Example. [Block World, continued] Consider Example 1.0.1 with the following additional feature: a block can be moved only if it is *light*. In the rules, below, *move*, *free*, *put_on_table*, *delete*, and *insert* represent actions and *on*, *clear*,

light, *color*, *not_broken*, and *weight* are fluents.

$$
\begin{array}{rlll}
(i) & \mathit{move}(X,Y) & \leftarrow & \mathit{on}(X,Z) \otimes \mathit{clear}(X) \otimes \mathit{not\_broken}(X) \otimes \\
 & & & \mathit{clear}(Y) \otimes \mathit{not\_broken}(Y) \otimes \mathit{light}(X) \otimes \mathit{color}(X,W) \otimes \\
 & & & \mathit{color}(Y,W) \otimes \mathit{delete}(\mathit{on}(X,Z)) \otimes \mathit{insert}(\mathit{on}(X,Y)) \\
 & & & \otimes \mathit{delete}(\mathit{clear}(Y)) \\
(ii) & \mathit{move}(X,Y) & \leftarrow & \mathit{on}(Z,X) \otimes \mathit{free}(Z,X) \otimes \mathit{move}(X,Y) \\
(iii) & \mathit{move}(X,Y) & \leftarrow & \mathit{on}(Z,Y) \otimes \mathit{free}(Z,Y) \otimes \mathit{move}(X,Y) \\
(iv) & \mathit{free}(X,Y) & \leftarrow & \mathit{clear}(X) \otimes \mathit{clear}(Y) \otimes \mathit{not\_broken}(X) \otimes \\
 & & & \mathit{put\_on\_table}(X) \otimes \mathit{delete}(\mathit{on}(X,Y)) \otimes \mathit{insert}(\mathit{clear}(Y)) \\
(v) & \mathit{free}(X,Y) & \leftarrow & \mathit{on}(W,X) \otimes \mathit{free}(W,X) \otimes \mathit{free}(X,Y) \\
(vi) & \mathit{light}(X) & \leftarrow & \mathit{weight}(X,W) \otimes \mathit{limit}(L) \otimes W < L \\
(vii) & \mathit{put\_on\_table}(X) & \leftarrow & \mathit{insert}(\mathit{on}(X,\mathit{table})) \\
(viii) & & ?- & \mathit{move}(blk1,blk2) \otimes \mathit{move}(\mathit{SomeBlk},blk1)
\end{array}
$$

Rules $(i), (ii)$ and (iii) define the complex action for moving a block from the top of one block, to the top of another.

This action is defined in terms of the built-in elementary updates *insert* and *delete* and the complex action *free* that recursively clear the tops of the blocks. The action *free* is defined in the rules (iv) and (v). Rule (vi) defines the fluent *light*, which is used in the definition of *move*. That rule consists exclusively of fluents and thus is a regular logic programming rule. Since all the literals involved in the definition of *light* are fluents, they cause no state transitions and the use of serial conjunction $\otimes$ in that rule is equivalent to the use of classical conjunction $\wedge$. Thus, rule (vi) could also be written as

$$\mathit{light}(X) \leftarrow \mathit{weight}(X,W) \wedge \mathit{limit}(L) \wedge W < L$$

Rule (vii) defines the action *put_on_table*. The last statement in the example is an update transaction, which moves block $blk1$ from its current position to the top of $blk2$ and then finds some other block and moves it on top of $blk1$. For instance, if the current database state is

$$
\begin{aligned}
\mathbf{D}_1 = \{ & \mathit{clear}(blk1), \mathit{clear}(blk2), \mathit{clear}(blk3), \mathit{on}(blk1,\mathit{table}), \mathit{on}(blk3,\mathit{table}), \\
& \mathit{on}(blk2,\mathit{table}), \mathit{color}(blk1,red), \mathit{not\_broken}(blk1), \\
& \mathit{color}(blk2,red), \mathit{color}(blk3,red), \mathit{not\_broken}(blk2), \mathit{not\_broken}(blk3)\}
\end{aligned}
$$

then execution of the transaction *move* changes the database state to

$$\mathbf{D}_2 = \mathbf{D}_1 \setminus \{\mathit{clear}(blk2), \mathit{on}(blk1,\mathit{table})\} \cup \{\mathit{on}(blk1,blk2)\}$$

(assuming that all the blocks involved satisfy the predicate *light* above) and then, instantiating $\mathit{SomeBlk}$ to $blk3$, to

$$
\begin{aligned}
\mathbf{D}_3 = \mathbf{D}_2 \setminus \{\mathit{clear}(blk1), \mathit{on}(blk3,\mathit{table})\} \cup \\
\{\mathit{on}(blk3,blk1)\}
\end{aligned}
$$

□

Model Theory.

In $\mathcal{TR}^-$, truth of a transaction is defined over sequences of states, called *execution paths* (or simply *paths*). When the user executes a transaction, the underlying database may change, going from the initial state to some other state. In doing so, the execution may pass through any number of intermediate states.

2.3.2. DEFINITION. [State] A **database state** (or simply *state*) is a set of **ground** (i.e., variable-free) fluent literals. □

For example the execution of $insert(a) \otimes insert(b) \otimes insert(\mathbf{neg}\, c)$ takes a relational database from an initial state $\mathbf{D}$ through the intermediate states $\mathbf{D}_1 = \mathbf{D} \cup \{a\} \setminus \{\mathbf{neg}\, a\}$ and $\mathbf{D}_2 = \mathbf{D}_1 \cup \{b\} \setminus \{\mathbf{neg}\, b\}$, to the final state $\mathbf{D}_3 = \mathbf{D}_2 \cup \{\mathbf{neg}\, c\} \setminus \{c\}$.

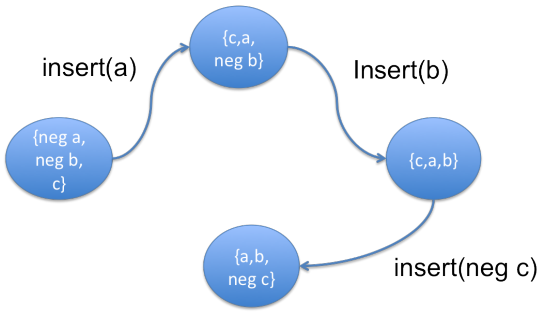

Figure 2.1: Actions in $\mathcal{TR}^-$

In this thesis, we will use only the Herbrand semantics for $\mathcal{TR}^-$. The semantics defines *path structures*, which generalize the usual first-order semantic structures (also called *interpretations*). As in first-order logic, the domain of Herbrand path structures is called the Herbrand universe $\mathcal{U}$; it is the set of all ground first-order terms that can be constructed from the function symbols in the given language $\mathcal{L}_{\mathcal{TR}}$. The Herbrand base $\mathcal{B}$ is a set of all ground literals in the language. A *classical* Herbrand structure is a subset of $\mathcal{B}$. Note that the Herbrand structure and Herbrand base are infinite, fixed, and depend only on the language $\mathcal{L}_{\mathcal{TR}}$, not on the transaction base. Since this thesis deals with Herbrand path structures only, we shall often omit the adjective "Herbrand."

A central feature in the semantics of $\mathcal{TR}^-$ is the notion of (execution) *paths* and the associated operation of *splitting* of paths into subpaths.

2.3.3. DEFINITION. [Path and split] An **execution path** of length k, or a **k-path**, is a finite sequence of states, $\pi = \langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $k \geq 1$. A *split* of π

is a pair of subpaths, π_1 and π_2, such that $\pi_1 = \langle \mathbf{D}_1 \ldots \mathbf{D}_i \rangle$ and $\pi_2 = \langle \mathbf{D}_i \ldots \mathbf{D}_k \rangle$ for some i $(1 \leq i \leq k)$. In this case, we write $\pi = \pi_1 \circ \pi_2$. □

It is worth noting that $\mathcal{TR}^-$ distinguishes between a database state $\mathbf{D}$ and the path $\langle \mathbf{D} \rangle$ of length 1. Intuitively, $\mathbf{D}$ represents the facts stored in the database, whereas $\langle \mathbf{D} \rangle$ represents the superset of $\mathbf{D}$ that can be derived from $\mathbf{D}$ and the rules in the transaction base.

$\mathcal{TR}$ is parametrized by data and transition *oracles*. The data oracles specifies a set of primitive database *queries*, i.e., the static semantics of states; and the transition oracle specifies a set of primitive database *updates*, i.e., the dynamic semantics of states. However, the formalism developed in Chapter 3, $\mathcal{TR}^{PAD}$, does not use *any* oracles—data or transition. Instead, the $\mathcal{TR}^{PAD}$ provides statements called *premises* that generalize these oracles.

2.3.4. Definition. [Herbrand Path Structures] A **Herbrand path structure**, $\mathcal{M}$, is a mapping that assigns a classical Herbrand structure to every path. This mapping is subject to the following restrictions, for all states $\mathbf{D}$, $\mathbf{D}_1$, $\mathbf{D}_2$ and base fluent p:

1. $\mathbf{D} \subseteq \mathcal{M}(\langle \mathbf{D} \rangle)$
2. $insert(p) \in \mathcal{M}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)$ iff $\mathbf{D}_2 = \mathbf{D}_1 \cup \{p\} \setminus \{\mathbf{neg}\, p\}$.
3. $delete(p) \in \mathcal{M}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)$ iff $\mathbf{D}_2 = \mathbf{D}_1 \setminus \{p\} \cup \{\mathbf{neg}\, p\}$.

Note that $delete(p)$ is equivalent to $insert(\mathbf{neg}\, p)$. □

Intuitively, Herbrand path structures in $\mathcal{TR}$ have the same role a transition functions in temporal logics like CTL or μ-Calculus [29]. That is, they are relations between states and actions. However, while transition functions take a state and an action and return a set of states, a Herbrand path structure takes paths of the form $\langle \mathbf{D}_1 \ldots \mathbf{D}_n \rangle$ and returns the set of actions that are executable at $\mathbf{D}_1$ and for which at least one execution ends in state $\mathbf{D}_n$ (actions in $\mathcal{TR}$ can be non-deterministic).

The following definition formalizes the idea that truth of $\mathcal{TR}^-$ formulas is defined on paths. Intuitively, each atom that is true on a path represents a transaction whose execution causes the state changes specified by the path. As in classical logic, to define the truth value of quantified formulas we use the notion of variable assignment. A ***variable assignment*** (or an ***instantiation***) is a mapping $\nu : \mathcal{V} \longrightarrow \mathcal{U}$, which takes a variable as input and returns a Herbrand term as output. We extend the mapping from variables to terms in the usual way: $\nu(f(t_1, \ldots, t_n)) = f(\nu(t_1), \ldots, \nu(t_n))$. The mapping can be extended to literals in a similar fashion.

2.3.5. DEFINITION. [Satisfaction] Let $\mathcal{M}$ be a Herbrand path structure, π be a path, and ν be a variable assignment.

1. **Base case:** If p is a literal, then $\mathcal{M}, \pi \models_\nu p$ if and only if $\nu(p) \in \mathcal{M}(\pi)$.
2. **"Classical" conjunction and disjunction:** $\mathcal{M}, \pi \models_\nu \phi \wedge \psi$ iff $\mathcal{M}, \pi \models_\nu \phi$ and $\mathcal{M}, \pi \models_\nu \psi$. Similarly, $\mathcal{M}, \pi \models_\nu \phi \vee \psi$ iff $\mathcal{M}, \pi \models_\nu \phi$ or $\mathcal{M}, \pi \models_\nu \psi$.
3. **Implication:** $\mathcal{M}, \pi \models_\nu \phi \leftarrow \psi$ (or $\mathcal{M}, \pi \models_\nu \psi \rightarrow \phi$) iff whenever $\mathcal{M}, \pi \models_\nu \psi$ then also $\mathcal{M}, \pi \models_\nu \phi$.
4. **Serial conjunction:** $\mathcal{M}, \pi \models_\nu \phi \otimes \psi$ iff $\mathcal{M}, \pi_1 \models_\nu \phi$ and $\mathcal{M}, \pi_2 \models_\nu \psi$ for *some* split $\pi_1 \circ \pi_2$ of path π.
5. **Universal and existential quantification:** $\mathcal{M}, \pi \models_\nu (\forall X)\phi$ iff $\mathcal{M}, \pi \models_\mu \phi$ for *every* variable assignment μ that agrees with ν everywhere except on X.

   $\mathcal{M}, \pi \models_\nu (\exists X)\phi$ iff $\mathcal{M}, \pi \models_\mu \phi$ for *some* variable assignment μ that agrees with ν everywhere except on X.
6. **Executional possibility:** $\mathcal{M}, \pi \models_\nu \Diamond\phi$ iff π is a 1-path of the form $\langle \mathbf{D} \rangle$, for some state $\mathbf{D}$, and $\mathcal{M}, \pi' \models_\nu \phi$ for some path π' that begins at $\mathbf{D}$. □

As in classical logic, the variable assignment can be omitted for *sentences*, *i.e.*, for formulas with no free variables. From now on, we will deal only with sentences, unless explicitly stated otherwise. If $\mathcal{M}, \pi \models \phi$, then we say that sentence ϕ is *satisfied* (or is *true*) on path π in structure $\mathcal{M}$.

2.3.6. DEFINITION. [Model] A path structure, $\mathcal{M}$, is a **model of a formula** ϕ if $\mathcal{M}, \pi \models \phi$ for every path π. In this case, we write $\mathcal{M} \models \phi$. A path structure is a **model of a set of formulas** if it is a model of every formula in the set. □

Executional Entailment.

A $\mathcal{TR}^-$ program consists of two distinct parts: a transaction base $\mathbf{P}$ and an initial database state $\mathbf{D}$. The database is a set of fluents and the transaction base is a set of transaction formulas. With this in mind we can define *executional entailment*, a concept that relates the semantics of $\mathcal{TR}^-$ to the notion of execution.

2.3.7. DEFINITION. [Executional entailment] Let $\mathbf{P}$ be a transaction base, ϕ a transaction formula, and let $\mathbf{D_0} \ldots \mathbf{D_n}$ be a sequence of databases. Then the following statement

$$\mathbf{P}, \mathbf{D_0} \ldots \mathbf{D_n} \models \phi \tag{2.5}$$

is said to be true if and only if $\mathcal{M}, \langle \mathbf{D_0} \ldots \mathbf{D_n} \rangle \models \phi$ for every model $\mathcal{M}$ of $\mathbf{P}$. Related to this is the following statement

$$\mathbf{P}, \mathbf{D_0} \text{---} \models \phi$$

which is true if and only if there is a database sequence $\mathbf{D_0} \ldots \mathbf{D_n}$ that makes (2.5) true. □

Intuitively (2.5) says that a successful execution of transaction ϕ, over the transaction base $\mathbf{P}$, can change the database from state $\mathbf{D}_0$ to $\mathbf{D}_1$ ... to $\mathbf{D}_n$.

Serial-Horn Transaction Bases.

One particular well-studied subset of Transaction Logic consists of so-called serial-Horn rules. This subset has a sound and complete SLD-style proof theory, and Section 3.5 shows that under certain assumptions this subset is reducible to ordinary logic programming.

Serial-Horn $\mathcal{TR}^-$, is the fragment of $\mathcal{TR}^-$ that consists of *serial-Horn* rules. A ***serial-Horn rule*** is a statement of the form

$$c \leftarrow c_1 \otimes \cdots \otimes c_n \tag{2.6}$$

where the body of the rule is *a serial-Horn goal*, c is an atom and $n \geq 0$. If the rule head is a fluent literal then we require that all the body literals are also fluents. We will refer to this last type of rules as *fluent rules*. A ***serial-Horn goal*** has the form $c_1 \otimes \cdots \otimes c_n$, where each c_i is a literal or an hypothetical-serial goal. Recall that a literal can be either a fluent or an action, and action literals are always positive. A serial-Horn rule can be understood as a statement that c is an invocation sequence of a complex action and $c_1 \otimes \cdots \otimes c_n$ is a definition of the actual course of action to be performed by that transaction. It is worth noticing that c can be defined by more than one serial-Horn rule.

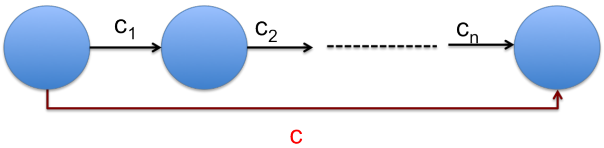

Figure 2.2: Horn-Rules

Consider a rule of the form (2.6). We will say that c is a ***compound action*** (a.k.a. complex action) if c is an action atom. We will say that c is a ***defined fluent*** if c is a fluent literal. A ***serial-Horn transaction base*** is a finite set of serial-Horn rules. Note that Example 2.3.1 is serial-Horn.

Chapter 3

Transaction Logic With Partially Defined Actions

> There are no moral or intellectual merits. Homer composed the Odyssey; if we postulate an infinite period of time, with infinite circumstances and changes, the impossible thing is not to compose the Odyssey, at least once.
>
> Jorge Luis Borges
> The Immortal

Transaction Logic ($\mathcal{TR}$) [1, 11, 12] was intended as a formalism for declarative specification of complex state-changing transactions in logic programming.

Although $\mathcal{TR}$ was created to program state-changing transactions, [10] demonstrated that $\mathcal{TR}$ can do basic, yet interesting reasoning about actions. However, [10] was unable to develop a complete proof theory, and the fragment of $\mathcal{TR}$ studied there was not expressive enough for modeling many problems in the context of action languages (cf. Example 3.2.3). In this section we continue that investigation and develop a full-fledged theory, *Transaction Logic with Partially Defined Actions* ($\mathcal{TR}^{PAD}$), for reasoning about actions over states *in addition* to programming the actions.

$\mathcal{TR}^{PAD}$ deviates from $\mathcal{TR}^{-}$ in that: *(i)* $\mathcal{TR}^{PAD}$ includes *premise*-formulas; *(ii)* it does not make use of the data or transition relational oracles, since they are generalized using premises (c.f. Section 3.1); *(iii)* it does not have built-in actions, since they can be represented using partially defined actions; and *(iv)* it allows default negation.

Our main focus in this chapter is the development of the formalism itself and illustration of its capabilities. $\mathcal{TR}^{PAD}$ has a great deal of sophistication in action composition, enabling hypothetical, recursive, and non-deterministic actions. In particular, compared with other actions languages like [39, 35, 8, 17, 38, 16], $\mathcal{TR}^{PAD}$ supports more general ways of describing actions and can be more selective in when and whether the fluents are subject to the laws of inertia.

Our contribution in this chapter is many-fold: *(i)* extension of $\mathcal{TR}$ with *premise*-formulas, which make $\mathcal{TR}$ more suitable for specifying partial knowledge about actions; *(ii)* defining a subset of the resulting formalism, called $\mathcal{TR}^{PAD}$, and demonstrating its expressive power for high-level descriptions of the behavior of complex actions; *(iii)* development of a sound and complete proof theory for $\mathcal{TR}^{PAD}$; *(iv)* a sound and complete reduction of the definite subset of $\mathcal{TR}^{PAD}$ to regular logic programming; and *(v)* an extension of $\mathcal{TR}^{PAD}$ with default negation under a variant of the well-founded semantics [79] for $\mathcal{TR}^{PAD}$. The reduction of the definite subset of $\mathcal{TR}^{PAD}$ to regular logic programming provides an easy way to implement and experiment with the formalism.

3.1 Partially Defined Actions and Incomplete Information

This section extends $\mathcal{TR}^{-}$ making it suitable for representing commonsense knowledge about actions and for reasoning about their effects in the presence of incomplete information. We introduce so-called *premise* formulas, which generalize the relational data and transition oracles, and then propose a sublanguage of the resulting extended formalism. The new formalism, called $\mathcal{TR}^{PAD}$, is a substantial generalization of the serial-Horn subset of $\mathcal{TR}^{-}$, which was studied in [1, 11, 12] and briefly described in Section 2.3. It has a sound and complete proof theory, is much more expressive, and better lends itself to complex representational and reasoning tasks about actions.

The alphabet of a language, $\mathcal{L}_{\mathcal{TR}^{PAD}}$, of $\mathcal{TR}^{PAD}$ is defined as in $\mathcal{TR}^{-}$. For convenient reference we will repeat it here. $\mathcal{L}_{\mathcal{TR}^{PAD}}$ consists of:

- A countably infinite set of variables $\mathcal{V}$.
- A countably infinite set of function symbols $\mathcal{F}$, where constants are treated as 0-arity function symbols.
- A countably infinite set of predicates $\mathcal{P}$. This set is further partitioned into two countably infinite subsets, $\mathcal{P}_{fluents}$ and $\mathcal{P}_{actions}$. As before, fluents are further partitioned into **base fluents** and **derived fluents**. Recall that derived fluents are those fluents that appear in the head of some fluent rule. Base fluents are the fluents which are not defined by Horn-rules.

- Logical connectives $\wedge$, $\vee$, the implication connectives $\leftarrow$ and $\rightarrow$, sequential conjunction $\otimes$, and hypothetical operator $\diamond$.
- The explicit negation connective **neg**.
- Quantifiers $\forall$ and $\exists$.

$\mathcal{TR}^{PAD}$ consists of *serial-Horn rules* (including *fluent rules*, c.f. Section 2.3), *partial action definitions* (PADs), and certain statements about action execution, which we call *premises*. A premise is a new kind of formula that was not in the original Transaction Logic (and thus not in $\mathcal{TR}^{-}$).

A ***partial action definition*** (or a ***PAD***) is a statement of the form:

$$b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \tag{3.1}$$

where b_1 and b_2 are conjunctions of fluent literals, b_3 and b_4 are conjunctions of *base* fluent literals, and α is an action atom. The serial conjunction $\otimes$ binds stronger than the implication, so the above PAD statement should be interpreted as $(b_1 \otimes \alpha \otimes b_2) \rightarrow (b_3 \otimes \alpha \otimes b_4))$. We will say that b_1 is a ***precondition*** of the action α and b_4 is its ***effect***. In addition, b_2 will be called ***post-condition*** and b_3 is a ***pre-effect***. Intuitively, (3.1) means that whenever we know that b_1 holds before executing α and b_2 holds after, we can conclude that b_3 must have held before executing α and b_4 must hold as a result of α (c.f. Figure 3.1). Observe that in absence of pre-effects and post-conditions, PADs can be seen as actions in STRIPS [30].

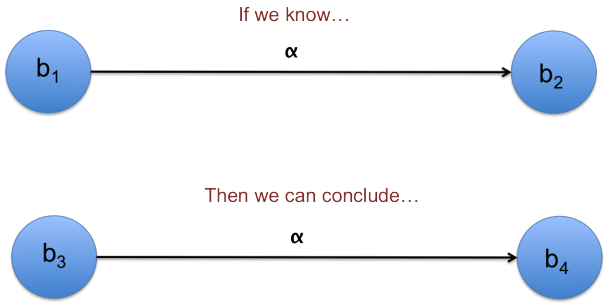

Figure 3.1: PADs

For instance, the PAD,

$$\textit{alive\_turkey} \otimes \textit{shoot} \otimes \textbf{neg}\ \textit{alive\_turkey} \rightarrow \textit{loaded} \otimes \textit{shoot}$$

states that if a turkey is alive before firing the gun and is dead after the shooting, then we can conclude that the gun was loaded initially.

Since b_1, b_2, b_3, and b_2 are conjunctions of fluents, the serial and classical conjunctions behave identically, since for fluents the semantics of $\mathcal{TR}^-$ guarantees that

$$f^1 \wedge ... \wedge f^n \equiv f^1 \otimes ... \otimes f^n$$

Each individual conjunct in b_1 will also be called a ***primitive precondition*** and in b_2 a ***primitive post-condition***. Similarly, each individual conjunct in b_3 will be referred to as a ***primitive pre-effect*** and in b_4 as ***primitive effect***.

Like $\mathcal{TR}^-$, $\mathcal{TR}^{PAD}$ uses only relational underlying states, i.e., they are simply sets of fluents. It is crucial to note that $\mathcal{TR}^{PAD}$ does not use *any* oracles—data or transition—because they are "black boxes" for the logic. Moreover, $\mathcal{TR}^{PAD}$ makes no use of the built-in actions $insert(f)$ and $delete(f)$ defined by transition oracles, since they can be axiomatized by the following PADs:

$$\begin{array}{l} insert(f) \rightarrow insert(f) \otimes f \\ g \otimes insert(f) \rightarrow insert(f) \otimes g \quad \text{where } g \neq f \text{ and } g \neq \mathbf{neg}\, f \\ delete(f) \rightarrow delete(f) \otimes \mathbf{neg}\, f \\ g \otimes delete(f) \rightarrow delete(f) \otimes g \quad \text{where } g \neq f \text{ and } g \neq \mathbf{neg}\, f \end{array}$$

Intuitively, transition oracles are replace by *run*-premises of the form

$$\mathbf{d}_1 \overset{insert(f)}{\rightsquigarrow} \mathbf{d}_2$$

Observe that in $\mathcal{TR}^-$ we were unable to refer explicitly to the facts in a given state, and moreover, relational data oracles need complete information about the database states to be well-defined. Thus, $\mathcal{TR}^-$ was not suitable for describing domains with incomplete information. To overcome this problem we make use of state premises, defined below. States premises can describe partially the database states. All these concepts will be illustrated with the examples in Section 3.2.

To sum up, in $\mathcal{TR}^{PAD}$ we will not distinguish built-in actions in any way. However, we will be distinguishing between ***partially defined actions*** (abbr., *pda*) and ***complex actions***. Partially defined actions cannot be defined by Horn rules—they can be defined by PADs only. In contrast, complex actions will be defined by Horn rules only, not by PADs. An important point is that *pda*s can appear in the rule bodies that define complex actions and, in this way, $\mathcal{TR}^{PAD}$ can be used to create larger action theories out of smaller ones in a modular way.

A $\mathcal{TR}^{PAD}$ ***transaction base*** is a set of serial-Horn rules and partial action definitions.

One key addition that $\mathcal{TR}^{PAD}$ brings to $\mathcal{TR}$ is the notion of *premises*. In premises, states are referred to with the help of special constants called ***state***

identifiers. We will be usually using boldface lowercase letters $\mathbf{d}$, $\mathbf{d}_1$, $\mathbf{d}_2$, to represent them. In $\mathcal{TR}$, state identifiers are not part of the language, since $\mathcal{TR}$ formulas never refer to database states explicitly. In the following, for the sake of simplicity, we will refer to state identifiers just as states.

3.1.1. DEFINITION. [Premise] A **premise** is a statement that has one of the following forms:

- A **state**-premise: $\mathbf{d} \rhd f$, where f is a fluent and $\mathbf{d}$ a database identifier. Intuitively, it means that f is known to be true at state $\mathbf{d}$.
- A **run**-premise: $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$, where α is a partially defined action. Intuitively it says that execution of action α in state represented by $\mathbf{d}_1$ is known to lead to state denoted by $\mathbf{d}_2$ (among others).[1]

A $\mathcal{TR}^{PAD}$ **specification**—depicted in Figure 3.2—is a pair $(\mathbf{P}, \mathcal{S})$ where $\mathbf{P}$ is a $\mathcal{TR}^{PAD}$ transaction base, and $\mathcal{S}$ is a set of premises. □

Intuitively, the set of premises can be viewed as a transition system, where $\rightsquigarrow$ is the transition relation.

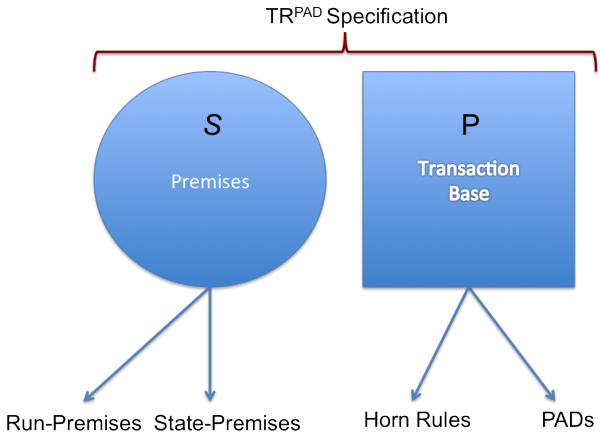

Figure 3.2: $\mathcal{TR}^{PAD}$ Specification

Usually, premises are statements about the initial and the final database states, and statements about some possible executions of partially defined actions. Typically these are partial descriptions so several different database states may satisfy the state-premises and several execution paths may satisfy the run-premises. Let us now turn to the semantics of $\mathcal{TR}^{PAD}$ specifications.

[1] In general, an action can be non-deterministic and may non-deterministically move to any one of a number of states.

3.1.2. DEFINITION. [Herbrand Path Structures] A **Herbrand path structure**, $\mathcal{M}$, is a mapping that assigns a classical Herbrand structure to every path. This mapping must satisfy the following condition for every state $\mathbf{D}$:

$$\mathbf{D} \subseteq \mathcal{M}(\langle \mathbf{D} \rangle)$$

In addition, $\mathcal{M}$ includes a mapping of the form $\Delta_{\mathcal{M}}$: *State identifiers* $\longrightarrow$ *Database states*, which associates states to state identifiers. We will usually omit the subscript in $\Delta_{\mathcal{M}}$.

A ***path abstraction*** is a finite sequence of state identifiers. If $\langle \mathbf{d}_1 \ \ldots \ \mathbf{d}_k \rangle$ is a path abstraction then $\langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $\mathbf{D}_i = \Delta(\mathbf{d}_i)$, is an execution path. We will also sometimes write $\mathcal{M}(\langle \mathbf{d}_1 \ \ldots \ \mathbf{d}_k \rangle)$ meaning $\mathcal{M}(\langle \Delta(\mathbf{d}_1) \ \ldots \ \Delta(\mathbf{d}_k) \rangle)$. □

3.1.3. DEFINITION. **(Models)** Let $\mathbf{P}$ be a transaction base, and $\mathcal{M}$ be a Herbrand path structure, such that $\mathcal{M} \models \mathbf{P}$, and let σ be a premise statement. We say that $\mathcal{M}$ **satisfies** σ, denoted $\mathcal{M} \models \sigma$, iff:

- σ is a run-premise of the form $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ and $\mathcal{M}, \langle \Delta(\mathbf{d}_1)\Delta(\mathbf{d}_2) \rangle \models \alpha$.
- σ is a state-premise $\mathbf{d} \rhd f$ and $\mathcal{M}, \langle \Delta(\mathbf{d}) \rangle \models f$.

$\mathcal{M}$ is a **model** of a set of premises $\mathcal{S}$ if it satisfies every statement in $\mathcal{S}$. □

3.1.4. DEFINITION. [Entailment] Let $\mathbf{P}$ be a $\mathcal{TR}^{PAD}$ transaction base, ϕ a transaction formula, and let $\mathcal{S}$ be a set of premises. We write

$$\mathbf{P}, \mathcal{S}, \mathbf{d_1} \ldots \mathbf{d_n} \models \phi \tag{3.2}$$

if and only if for every model $\mathcal{M}$ of $\mathbf{P}$ and $\mathcal{S}$, we have $\mathcal{M}, \langle \mathbf{\Delta}(\mathbf{d_1}) \ldots \mathbf{\Delta}(\mathbf{d_n}) \rangle \models \phi$. □

3.2 Representing Actions with $\mathcal{TR}^{PAD}$

We will now show how $\mathcal{TR}^{PAD}$ can be used to represent complex scenarios that arise in reasoning about actions. We will discuss which conclusions are desired in each case, but the machinery needed to do the actual reasoning will be developed in subsequent sections.

3.2.1. EXAMPLE. [Health Insurance] Consider the US health insurance regulations scenario discussed in the introduction. Suppose we want to formalize the following regulations:
(i) The AIDS and DNA tests ($\mathit{aids\_t}(T)$ and $\mathit{dna\_t}(T)$), require *prior* consent of

the patient ($need\_consent(T)$).
(ii) To perform a test T prescribed by doctor D to patient P in compliance with the law ($do\_cmplnt\_test(T, P, D)$), T must be done ($do\_t(T, P, D)$) only *after* D prescribed the test T ($do\_presc(D, T)$), which in turn must be done *after* receiving the consent of P ($rcv\_consent(P, T)$).
In $\mathcal{TR}^{PAD}$, this is expressed as follows:

(1) $need\_consent(T) \leftarrow aids\_t(T)$

(2) $need\_consent(T) \leftarrow dna\_t(T)$

(3) $do\_cmplnt\_test(T, P, D) \leftarrow rcv\_consent(P, T) \otimes consent(P, T) \otimes do\_presc(T, P, D) \otimes presc(T, P, D) \otimes do\_t(T, P, D)$

In the rules above, *do_cmplnt_test*, *rcv_consent*, *do_presc* and *do_t* are actions, while *need_consent*, *dna_t*, *aids_t*, *consent* and *presc* are fluents. Rules (1) and (2) define the fluent *need_consent*. They consist exclusively of fluents so they are regular logic programming rules that do not cause state transitions. Moreover, serial conjunction of fluents is equivalent to the use of the classical conjunction, since fluents do not cause state transitions. Rules (1) and (2) formalize regulation (i). Rule (3) defines the *compound* action *do_cmplnt_test* which formalizes regulation (ii). The three actions in Rule (3) will be defined in Example 3.2.2. They are *partially defined actions*, which we will define in the following chapter. Note that compound actions like *do_cmplnt_test* cannot be expressed in action languages like [35, 8, 38].

The next statement is an update transaction, where wb, s, and m are constants.

$$?\text{-}\ aids\_t(wb) \otimes do\_cmplnt\_test(wb, m, s) \otimes negative(m, wb)$$

It first queries the database to check if Western Blot (wb) is an aids test. If it is, the transaction executes the compound action *do_cmplnt_test* to perform a complaint test wb for the patient Mark (m) prescribed by Dr. Smith (s). If the test finishes successfully, the transaction checks that the result is negative and all is well. Note that if after executing *do_cmplnt_test* the transaction fails, for example because Mark's consent was not received, actions are "backtracked over," and the underlying database state remains unchanged. □

3.2.2. Example. [Health Insurance, continued] Consider Example 3.2.1, and let us now present the three PADs that were left undefined. We also add the fluents *dr*, *matching*, and *finished*.

$$\mathbf{P} = \begin{cases} \mathbf{neg}\ finished(P, T) \otimes \mathbf{neg}\ matching(P, T) \otimes do\_t(T, P, D) \rightarrow \\ \qquad do\_t(T, P, D) \otimes finished(P, T) \otimes negative(P, T) \\ patient(P) \otimes need\_consent(T) \otimes rcv\_consent(P, T) \rightarrow \\ \qquad rcv\_consent(P, T) \otimes consent(P, T) \\ dr(D) \otimes do\_presc(T, P, D) \rightarrow do\_presc(T, P, D) \otimes presc(D, P, T) \end{cases}$$

The first PAD states that the result of the test is negative if the test is still in

process (i.e., not finished) and there is no match with the patient's sample. The second and third rules define the actions *rcv_consent* and *do_presc*. Suppose that Mark (m) got a PCR DNA test (pr) prescribed by Doctor Smith (s); and we know that the result of the test did not match the sample and the test finished successfully. The premises for the problem at hand are as follows:

$$\mathcal{S} = \begin{cases} \mathbf{d}_1 \overset{\textit{rcv\_consent}(m,pr)}{\rightsquigarrow} \mathbf{d}_2 & - \ m\text{'s consent is received at } \mathbf{d}_1, \text{ which leads to } \mathbf{d}_2 \\ \mathbf{d}_2 \overset{\textit{do\_presc}(m,pr,s)}{\rightsquigarrow} \mathbf{d}_3 & - \ \text{The prescription is received at } \mathbf{d}_2 \text{ leading to } \mathbf{d}_3 \\ \mathbf{d}_3 \overset{\textit{do\_t}(m,pr,s)}{\rightsquigarrow} \mathbf{d}_4 & - \ m\text{'s test is made at state } \mathbf{d}_3 \text{ and it results in } \mathbf{d}_4 \\ \mathbf{d}_1 \rhd \mathbf{neg}\ \textit{finished}(m,pr) & - \ \text{The test is not finished at state } \mathbf{d}_1 \\ \mathbf{d}_1 \rhd \textit{dna\_t}(pr) & - \ \text{PCR is a DNA test} \\ \mathbf{d}_1 \rhd \textit{patient}(m) & - \ \text{Mark is a patient} \\ \mathbf{d}_1 \rhd \textit{dr}(s) & - \ \text{Smith is a doctor} \\ \mathbf{d}_3 \rhd \mathbf{neg}\ \textit{matching}(m,pr) & - \ \text{There is no match with } m\text{'s sample} \\ \mathbf{d}_4 \rhd \textit{finished}(m,pr) & - \ \text{The test was performed successfully} \end{cases}$$

We would like the logic to infer that the result of the *compliant* PCR test for Mark was negative. That is,

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \text{---} \models \textit{do\_cmplnt\_test(pr,m,s)} \otimes \textit{negative}(m,pr)$$

□

Let us now consider a popular example in action languages, called the Turkey Hunting Problem, which is used in [35, 8, 38] among others.

3.2.3. Example. [The Turkey Shoot Problem [39]] A pilgrim goes turkey-hunting. If he fires a loaded gun, the turkey is dead in the next state. The turkey can die only by being shot. Assuming that the turkey is alive initially and dead afterwards, we want to be able to infer that the gun was loaded initially. For this problem, the fluents are *loaded* and *alive*, and the actions are *load* and *shoot*. The set of premises is:

$$\mathcal{S} = \begin{cases} \mathbf{d}_1 \overset{\textit{shoot}}{\rightsquigarrow} \mathbf{d}_2 \\ \mathbf{d}_1 \rhd \textit{alive} \\ \mathbf{d}_2 \rhd \mathbf{neg}\ \textit{alive} \end{cases}$$

The PADs for the above problem are as follows:

$$\begin{aligned} \textit{load} &\rightarrow \textit{load} \otimes \textit{loaded} \\ \textit{loaded} \otimes \textit{shoot} &\rightarrow \textit{shoot} \otimes \mathbf{neg}\ \textit{alive} \\ \textit{shoot} &\rightarrow \textit{shoot} \otimes \mathbf{neg}\ \textit{loaded} \end{aligned}$$

The above premises state that a shooting action has occurred at some state $\mathbf{D}_1$ $(= \Delta(\mathbf{d}_1))$, that the turkey was alive then, and that it was not alive after the action. The PADs describe the effects of loading and shooting. Our requirement is that the logic be strong enough to prove that the gun was loaded initially:

$$\mathbf{P}, \mathbf{d}_1 \models \mathit{loaded}$$

In general, there is not enough information to prove that in all models where *shoot* makes a transition from $\mathbf{D}_1$ to $\mathbf{D}_2$ $(= \Delta(\mathbf{d}_2))$, the following is impossible:

$$\mathbf{D}_1 = \{\mathbf{neg}\ \mathit{loaded}, \mathit{alive}\} \qquad \mathbf{D}_2 = \{\mathbf{neg}\ \mathit{loaded}, \mathbf{neg}\ \mathit{alive}\}$$

However, common sense reasoners, due to the inertia law, would normally reject transitions from such $\mathbf{D}_1$ to $\mathbf{D}_2$ because the fluent alive changes without a cause. □

To solve the problem highlighted in the above example, we need to be able to state the so-called *commonsense law of inertia* (or *frame axioms*), which says that things stay the same unless there is an explicitly stated cause for a change. However, the following example shows that there are situations where assuming that things change only due to a direct effect of an action (and remain the same otherwise) is inappropriate.

3.2.4. Example. [The Turkey Shoot Problem #2] Consider Example 3.2.3 with the following additional features:

- the gun can be loaded only if the pilgrim has bullets
- the pilgrim can only hunt during the day and
- In the third state the night falls

To represent this, we introduce two new fluents, *daylight* and *bullets*, and the following premises:

$$\mathcal{S} = \begin{cases} \mathbf{d}_1 \overset{shoot}{\rightsquigarrow} \mathbf{d}_2 \\ \mathbf{d}_2 \overset{load}{\rightsquigarrow} \mathbf{d}_3 \\ \mathbf{d}_1 \rhd \mathit{daylight} \\ \mathbf{d}_1 \rhd \mathit{alive} \\ \mathbf{d}_2 \rhd \mathbf{neg}\ \mathit{alive} \\ \mathbf{d}_3 \rhd \mathbf{neg}\ \mathit{loaded} \\ \mathbf{d}_3 \rhd \mathbf{neg}\ \mathit{daylight} \end{cases}$$

The PADs for the above problem are as follows:

$$\begin{aligned} \mathit{bullets} \otimes \mathit{load} &\rightarrow \mathit{load} \otimes \mathit{loaded} \\ \mathit{daylight} \wedge \mathit{loaded} \otimes \mathit{shoot} &\rightarrow \mathit{shoot} \otimes \mathbf{neg}\ \mathit{alive} \\ \mathit{shoot} &\rightarrow \mathit{shoot} \otimes \mathbf{neg}\ \mathit{loaded} \end{aligned}$$

These premises state that a shooting occurs at some state represented by $\mathbf{d}_1$ and then a load action at $\mathbf{d}_2$. Also, initially the turkey was alive and there was daylight, but following the shooting, the turkey was not alive. After shooting and loading took place, the gun was found to be unloaded, and it was dark outside. The PADs describe the effects of the loading and shooting actions We want our logic to conclude that the gun was loaded initially and after shooting the pilgrim must have run out of bullets:

$$\mathbf{P}, \mathbf{d}_1 \models \mathit{loaded}$$
$$\mathbf{P}, \mathbf{d}_2 \models \mathbf{neg}\ \mathit{bullets}$$

A subtle point here is that *daylight* is not a direct effect of an action, so a simplistic law of inertia would conclude

$$\mathbf{P}, \mathbf{d}_2 \models \mathbf{neg}\ \mathit{daylight} \quad \text{and} \quad \mathbf{P}, \mathbf{d}_1 \models \mathbf{neg}\ \mathit{daylight}$$

Clearly, this is not what we want in this case. □

3.2.5. Example. [The Turkey Shoot Problem #3] Consider again the scenario described in Example 3.2.3. Assuming that the gun is unloaded initially and the turkey is dead afterwards, we want to be able to infer that the turkey was not alive initially. We keep the same set of fluents and actions as in Example 3.2.3, and the premises are

$$\mathcal{S} = \begin{cases} \mathbf{d}_1 \overset{shoot}{\rightsquigarrow} \mathbf{d}_2 \\ \mathbf{d}_1 \rhd \mathbf{neg}\ \mathit{loaded} \\ \mathbf{d}_2 \rhd \mathbf{neg}\ \mathit{alive} \end{cases}$$

The above states that a shooting action has occurred at some state $\mathbf{d}_1$, that the gun was not loaded initially, and that the turkey was not alive after shooting. The PADs describe the effects of loading and shooting. Our requirement is that the logic be strong enough to prove that the turkey was not alive initially:

$$\mathbf{P}, \mathbf{d}_1 \models \mathbf{neg}\ \mathit{alive}$$

□

The following example illustrates the use of complex actions.

3.2.6. Example. [The Turkey Shoot Problem #4] Again we take Example 3.2.3 as a point of departure. We extend the set of fluents with *hidden* and *bird_in_range* and add the actions *find_location*, *hunt* and *hide*. The action *hunt* is a *complex* action composed of several partially defined actions and fluents. The pilgrim can *hunt* if he finds a good spot to shoot, manages to hide, the gun is loaded, and when he shoots he kills the turkey. The set of premises is:

$$\mathcal{S} = \begin{cases} \mathbf{d}_1 \overset{find\_location}{\rightsquigarrow} \mathbf{d}_2 \\ \mathbf{d}_2 \overset{hide}{\rightsquigarrow} \mathbf{d}_3 \\ \mathbf{d}_3 \overset{shoot}{\rightsquigarrow} \mathbf{d}_4 \\ \mathbf{d}_5 \overset{shoot}{\rightsquigarrow} \mathbf{d}_4 \\ \mathbf{d}_3 \rhd \mathit{loaded} \\ \mathbf{d}_2 \rhd \mathit{alive} \\ \mathbf{d}_4 \rhd \mathbf{neg}\ \mathit{alive} \end{cases}$$

The above premises state that shooting in state $\mathbf{d}_3$ or $\mathbf{d}_5$ leads to $\mathbf{d}_4$, hiding in state $\mathbf{d}_2$ leads to $\mathbf{d}_3$, and finding a location in $\mathbf{d}_1$ brings in state $\mathbf{d}_2$. We also know that the gun was loaded in $\mathbf{d}_3$, and that the turkey was not alive in $\mathbf{d}_4$, but it was alive in $\mathbf{d}_2$. (The index number should not be construed as implying a temporal order among the states $\mathbf{d}_1$, ..., $\mathbf{d}_5$.)

The PADs for the above problem describe the effects of loading, shooting, etc. They are as follows:

$$\begin{aligned} \mathit{load} &\rightarrow \mathit{load} \otimes \mathit{loaded} \\ \mathit{loaded} \otimes \mathit{shoot} &\rightarrow \mathit{shoot} \otimes \mathbf{neg}\ \mathit{alive} \\ \mathit{shoot} &\rightarrow \mathit{shoot} \otimes \mathbf{neg}\ \mathit{loaded} \\ \mathit{hide} &\rightarrow \mathit{hide} \otimes \mathit{hidden} \\ \mathit{find\_location} &\rightarrow \mathit{find\_location} \otimes \mathit{bird\_in\_range} \end{aligned}$$

The rules for the complex actions and derived fluents in the transaction base are shown below. The first rule defines a fluent, *correct_location*, and the second defines the complex action *hunt*.

$$\begin{aligned} &\mathit{correct\_location} \leftarrow \mathit{hidden} \otimes \mathit{bird\_in\_range} \\ &\mathit{hunt} \leftarrow \mathit{find\_location} \otimes \mathit{hide} \otimes \mathit{correct\_location} \otimes \mathit{loaded} \otimes \mathit{shoot} \end{aligned}$$

Our requirement is that the logic must be strong enough to prove that the turkey was not alive after executing *hunt* in $\mathbf{d}_1$:

$$\mathbf{P}, \mathbf{d}_1 \texttt{---} \models \mathit{hunt} \otimes \mathbf{neg}\ \mathit{alive} \qquad \square$$

Examples 3.2.3, 3.2.4,3.2.5, and 3.2.6 illustrate the need for additional axioms to express the common-sense inertia laws.

It is worth noting that the problem described in Examples 3.2.2, 3.2.3, 3.2.4, etc., cannot be expressed in the action language previously cited. For instance, the action language $\mathcal{A}$ [35], does not allow defined fluents, and neither $\mathcal{A}$ nor $\mathcal{AL}$ nor $\mathcal{AC}$ [35, 8, 38] support compound actions.

Note that in all previous examples we were using a restricted type of PADs of the form $b_1 \otimes \alpha \rightarrow \alpha \otimes b_2$. This restricted form is sufficient for most types

of action specification, but inertia and related laws require a more general kind. For example, a rule suitable for expressing the inertia needed in Example 3.2.3 is

$$\textbf{neg}\ \mathit{loaded} \otimes \mathit{shoot} \otimes \textbf{neg}\ \mathit{alive} \rightarrow \textbf{neg}\ \mathit{alive} \otimes \mathit{shoot}$$

It says that if shooting with an unloaded gun puts us in a state where the turkey is dead, the turkey must have been dead beforehand.

3.3 A Proof Theory for $\mathcal{TR}^{PAD}$

This section develops an inference system for proving statements about transaction execution. These statements, called ***sequents*** have the form $\mathbf{P}, \mathcal{S}, \mathbf{d} \text{---} \vdash \phi$, where ϕ is a serial-Horn goal and $(\mathbf{P}, \mathcal{S})$ a $\mathcal{TR}^{PAD}$ specification. Informally, such a sequent says that transaction ϕ can successfully execute starting at state $\mathbf{d}$. We refer to the inference system developed here as $\mathcal{F}$; it significantly generalizes the inference system $\mathcal{F}^H$ for the serial-Horn fragment of $\mathcal{TR}^-$ presented in [11].

3.3.1. Definition. [Inference System $\mathcal{F}$] Let $\mathbf{P}$ be a transaction base and $\mathcal{S}$ a set of premises. The inference system $\mathcal{F}$ consists of the following axioms and inference rules, where $\mathbf{d}$, $\mathbf{d}_1$, $\mathbf{d}_2$, ... denote arbitrary database states.
Axioms:

1. *No-op:* $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash ()$

Inference rules: In the rules below, a, and α are literals, and ϕ, ψ, and b_i $(i = 1, \ldots, 4)$ are serial goals.

1. *A subset of Horn inference rules from [11, 12]:*

   (a) *Applying transaction definitions:*

   $$\frac{a \leftarrow \phi \in \mathbf{P} \qquad \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi \otimes \psi}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash a \otimes \psi}$$

   (b) *Hypothetical operations:*

   $$\frac{\mathbf{P}, \mathcal{S}, \mathbf{d}, \mathbf{d}'_1, ..., \mathbf{d}'_n \vdash \beta \qquad \mathbf{P}, \mathcal{S}, \mathbf{d}, \mathbf{d}_1, ..., \mathbf{d}_m \vdash \gamma}{\mathbf{P}, \mathbf{d}, \mathbf{d}_1, ..., \mathbf{d}_m \vdash \Diamond\beta \otimes \gamma}$$

2. *Premise rules:* For each premise in $\mathcal{S}$:

   $$\frac{\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha} \qquad\qquad \frac{\mathbf{d} \rhd f \in \mathcal{S}}{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash f}$$

3. *Forward Projection:* Suppose α is a partially defined action. Then

$$\frac{\begin{array}{c} b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \in \mathbf{P} \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_2 \vdash b_2 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha \end{array}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_3 \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_2 \vdash b_4}$$

4. *Sequencing:*

$$\frac{\begin{array}{c} \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \vdash \psi \\ \text{where } 1 \leq i \leq n \end{array}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi \otimes \psi}$$

5. *Decomposition:* Suppose ϕ and ψ are serial conjunctions of literals and hypotheticals. Then

$$\frac{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \phi \otimes \psi}{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \phi \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \psi}$$

The next theorem relates the inference system $\mathcal{F}$ to the model-theory.

3.3.2. Theorem (Soundness and completeness). *For any serial goal ϕ and a $\mathcal{TR}^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$, the executional entailment $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$ holds if and only if there is a deduction in $\mathcal{F}$ of the sequent $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$*

1. Proof. *See Appendix A.* □

3.4 Axioms of Inertia and Action Theory

We now return to the problem of inertia discussed in Examples 3.2.3, 3.2.4, and 3.2.5. Given a $\mathcal{TR}^{PAD}$ transaction base $\mathbf{P}$, we augment it with suitable frame axioms and construct a specification $\mathcal{A}(\mathbf{P})$, called the ***action theory*** of $\mathbf{P}$, where $\mathbf{P} \subseteq \mathcal{A}(\mathbf{P})$. For simplicity we present a ground version of the frame axioms. Lifting to the non-ground case is done in a standard way (cf. [11]).

For this specification to be well-defined, we impose a restriction over ***interloping*** PADs—defined below. Observe that we do not impose this restriction on $\mathcal{TR}^{PAD}$ itself—only on the particular action theory presented in this section. For instance, the inference system and the reduction to logic programming given in Section 3.6 do not rely on this assumption. Some other action languages (e.g., the $\mathcal{A}$-language of [35]) impose the same restriction. To capture the inertia laws in $\mathcal{TR}^{PAD}$ without the restriction over interloping PADs, one needs a more elaborate theory, to be presented in Section 3.7. In Section 3.8 we show how to remove the restriction over interloping PADs.

Two PADs *same* action α are said to be ***interloping*** if they share a common primitive effect. For instance, the following PADs are interloping, as they share a fluent (*loaded*):

- *has_bullets* $\otimes$ *load* $\rightarrow$ *load* $\otimes$ *loaded*
- *has_ammunition* $\otimes$ *load* $\rightarrow$ *load* $\otimes$ (*loaded* $\wedge$ *ready*)

In this section, we will assume that $\mathcal{TR}^{PAD}$ transaction bases do *not* contain interloping PADs. Recall that we will be combining several formulas into one using the usual De Morgan's laws. In addition, recall that explicit negation connective **neg** is distributive with respect to conjunctions of fluent literals (serial and classical, which are equivalent for fluents) the same way as negation distributes through the regular classical conjunction according to Morgan's laws. For instance, the PAD

$$\mathbf{neg}\,(f_1 \wedge f_2) \wedge \mathbf{neg}\,(f_3 \wedge f_4) \otimes \alpha \rightarrow g$$

is equivalent to

$$(\mathbf{neg}\,f_1 \vee \mathbf{neg}\,f_2) \wedge (\mathbf{neg}\,f_3 \vee \mathbf{neg}\,f_4) \otimes \alpha \rightarrow g$$

which in turn is equivalent to

$$\begin{array}{c}(\mathbf{neg}\,f_1 \wedge \mathbf{neg}\,f_3) \otimes \alpha \rightarrow g\\ (\mathbf{neg}\,f_1 \wedge \mathbf{neg}\,f_4) \otimes \alpha \rightarrow g\\ (\mathbf{neg}\,f_2 \wedge \mathbf{neg}\,f_3) \otimes \alpha \rightarrow g\\ (\mathbf{neg}\,f_2 \wedge \mathbf{neg}\,f_4) \otimes \alpha \rightarrow g\end{array}$$

As explained in Example 3.2.3, it is a requirement that the frame axioms must be able to model a variety of different behaviors, depending on the problem at hand. In the following we define a general set of rules, $Frame(\mathbf{P})$, that encodes different aspects of the Frame Axioms, and at the end of this section we prove that this set of rules behave as expected. For instance, in Example 3.2.4 we expect that some fluents, like *alive*, are subject to the frame axioms, while others, like *daylight*, are not. We thus introduce a predicate, *inertial*, that indicates whether a fluent is subject to inertia.[2] If a fluent, f, behaves according to the frame axioms in state $\mathbf{D}$ $(= \Delta(\mathbf{d}))$, it is assumed that $\mathcal{S}$ has a *state*-premise of the form $\mathbf{d} \rhd$ *inertial*(f).

The ***action theory*** $\mathcal{A}(\mathbf{P})$ for a transaction base $\mathbf{P}$ is defined as $\mathbf{P} \cup Frame(\mathbf{P})$, where $Frame(\mathbf{P})$ is the following set of axioms:

[2] In some cases, we can also specify *inertial* via rules and facts. For instance, if every fluent is inertial, we could just have a universal fact *inertial*(F).

Forward Inertia For each base fluent literal h and each partially defined action α such that neither h nor $\mathbf{neg}\, h$ is a primitive effect of α, the following axiom is in $Frame(\mathbf{P})$:

$$(inertial(h) \wedge h) \otimes \alpha \rightarrow \alpha \otimes h \tag{3.3}$$

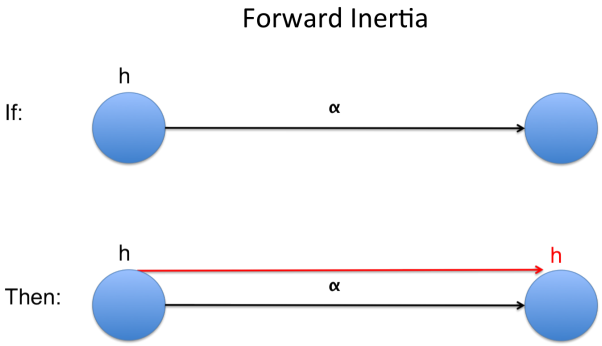

Figure 3.3: Forward Inertia

Here it is worth noting that the number of the Forward Inertia axioms are quadratic, i.e., it is proportional to the number of fluents times the number of actions. However, it is easy to replace all these axioms with just one if we use HiLog [21] and thus gain the ability to quantify over propositions. In that case, we could replace the above axiom schema with a single axiom of the form

$$(unrelated(H, Action) \wedge inertial(H) \wedge H) \otimes Action \rightarrow Action \otimes H$$

where H and $Action$ are variables and *unrelated* is a predicate that provides information on which fluents are independent of which actions.

Forward and Backward Disablement Let g or $\mathbf{neg}\, g$ be base literals and α a *pda*. Since we do not have interloping actions, there can be at most one partially defined action $\mathbf{p}_g$ with the primitive effect g and at most one *pda* $\mathbf{p}_{\mathbf{neg}\, g}$ with the primitive effect $\mathbf{neg}\, g$. Let f_g be the precondition of $\mathbf{p}_g$ and $f_{\mathbf{neg}\, g}$ the precondition of $\mathbf{p}_{\mathbf{neg}\, g}$ (if $\mathbf{p}_g$ or $\mathbf{p}_{\mathbf{neg}\, g}$ does not exist, assume that $\mathbf{neg}\, f_g$ or $\mathbf{neg}\, f_{\mathbf{neg}\, g}$ is true in every state). Then the following *forward disablement* axioms are in $Frame(\mathbf{P})$:

$$\begin{array}{lcl} (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes g \otimes \alpha & \rightarrow & \alpha \otimes g \\ (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \mathbf{neg}\, g \otimes \alpha & \rightarrow & \alpha \otimes \mathbf{neg}\, g \end{array} \tag{3.4}$$

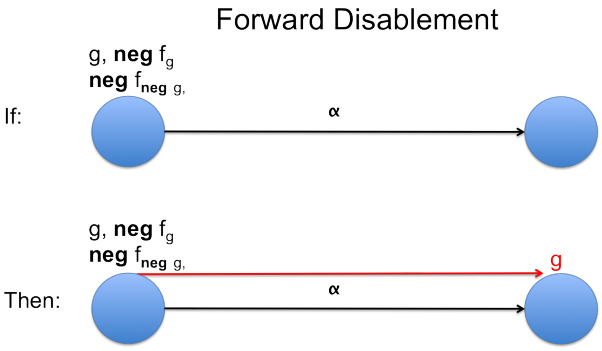

Figure 3.4: Forward Disablement

If g is a base fluent,[3] then the following *backward disablement* axioms are also in $Frame(\mathbf{P})$:

$$\begin{array}{lcl} (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \alpha \otimes g & \rightarrow & g \otimes \alpha \\ (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \alpha \otimes \mathbf{neg}\, g & \rightarrow & \mathbf{neg}\, g \otimes \alpha \end{array} \tag{3.5}$$

In other words, if the *pda*s $\mathbf{p}_g$ and $\mathbf{p}_{\mathbf{neg}\, g}$ are disabled in some state then executing α in that state does not change the truth value of the fluents g and $\mathbf{neg}\, g$.

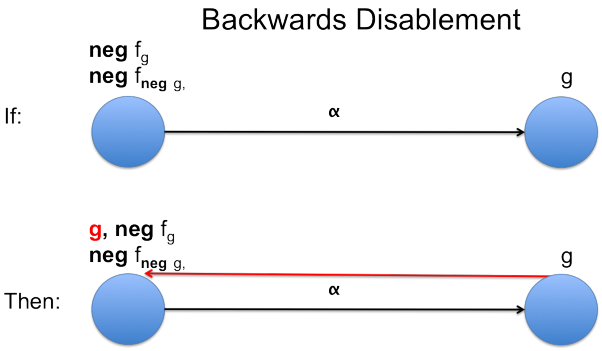

Figure 3.5: Backward Disablement

Backward Inertia For each *pda* α such that f is a base fluent which is not primitive effect of α:

$$inertial(f) \otimes \alpha \otimes f \rightarrow f \otimes \alpha \quad \in Frame(\mathbf{P}) \tag{3.6}$$

[3]Recall that base fluents are the fluents which are not defined by Horn-rules.

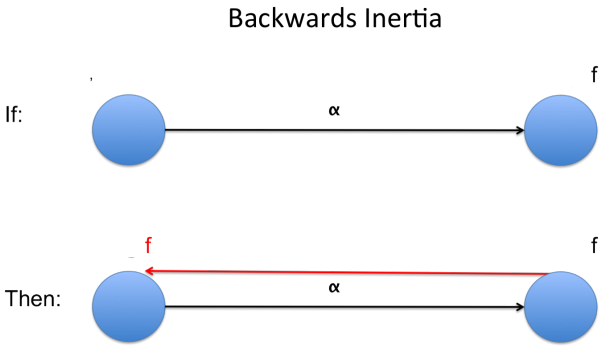

Figure 3.6: Backward Inertia

Causality For each PAD $b_1 \otimes \alpha \rightarrow \alpha \otimes b_2 \in \mathbf{P}$, and each base primitive effect b' that occurs as one of the conjuncts in b_2:

$$\mathbf{neg}\, b' \otimes \alpha \otimes b' \rightarrow b_1 \otimes \alpha \quad \in Frame(\mathbf{P}) \tag{3.7}$$

That is, if an effect of an action has been observed, the action must have been executed as prescribed by the unique (since there are no interloping PADs) PAD that specifies that effect. In particular, the precondition of that PAD must have been true.

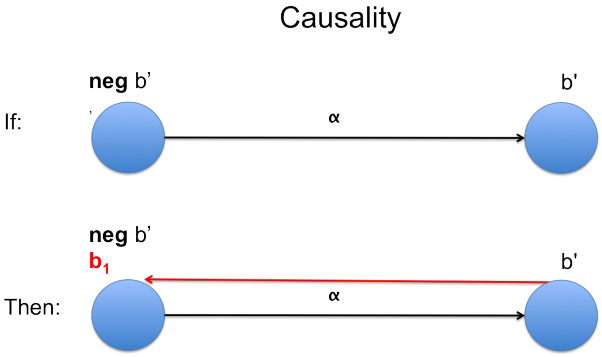

Figure 3.7: Causality

Backward Projection For each PAD in $\mathbf{P}$ of the form $(\wedge_{i=1}^{k} b_1^i) \otimes \alpha \rightarrow \alpha \otimes b_4$, and each base primitive precondition b_1^j

$$(\wedge_{i=1,i\neq j}^{k} b_1^i) \otimes \alpha \otimes \mathbf{neg}\, b_4 \rightarrow \mathbf{neg}\, b_1^j \otimes \alpha \quad \in Frame(\mathbf{P}) \tag{3.8}$$

That is, if all but one primitive preconditions hold, but the effect of the action is not observed in the next state, we must conclude that the remaining precondition was false prior to the execution.

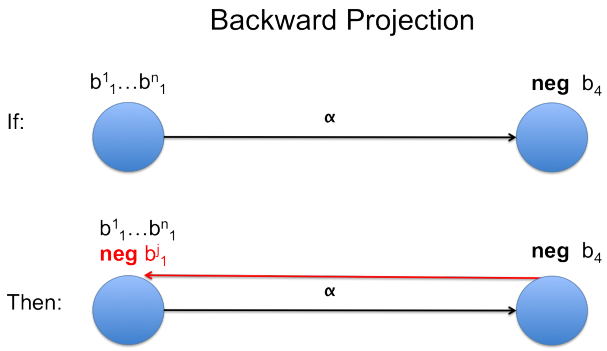

Figure 3.8: Backward Projection

We now return to our examples and show how the above action theory supports the kinds of reasoning that we desired in Section 3.2. We should point out one interesting aspect of $\mathcal{TR}^{PAD}$: the fact that the semantics of the logic and of the above action theory is completely monotonic—quite unlike the other approaches mentioned earlier. Unless we state it differently, all the fluents in this sections are inertial.

3.4.1. EXAMPLE. [Turkey Shoot, continued] The issue in Example 3.2.3 was the inability to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathit{loaded}$ (the gun was loaded initially), because $\mathcal{TR}^{PAD}$ was not sufficiently expressive to let us specify the rules of inertia. Fortunately, the PAD axioms $Frame(\mathbf{P})$ do the trick. Let $\mathcal{A}(\mathbf{P})$ be the action theory of $\mathbf{P}$. We now show how to prove $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathit{loaded}$ using the inference system $\mathcal{F}$. The relevant instance of the axioms in $Frame(\mathbf{P})$ is the causality axiom:

$$inertial(alive) \wedge \mathit{alive} \otimes \mathit{shoot} \otimes \mathbf{neg}\, \mathit{alive} \rightarrow \mathit{loaded} \otimes \mathit{shoot} \qquad (3.9)$$

Now:

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathit{alive}$	by the inference rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathit{alive}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \mathit{shoot}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathit{loaded}$	by the inference rule 3 (Forward projection), the instance (3.9) of the causality axiom and the above three sequents

The desired conclusion now follows from the soundness of $\mathcal{F}$ (Theorem 3.3.2). □

3.4.2. Example. [Turkey Shoot 2, continued] In Example 3.2.4 we wanted to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathit{loaded}$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models \mathbf{neg}\, \mathit{bullet}$, i.e., that the gun was loaded initially, and after shooting the pilgrim runs out of bullets. Furthermore, to ensure consistency, we should *not* be concluding $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathbf{neg}\, \mathit{daylight}$, i.e., that initially it was nighttime.

The proof that $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathit{loaded}$ is the same as in Example 3.2.3. Below we show how to prove $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \models \mathbf{neg}\, \mathit{bullet}$ using the inference system $\mathcal{F}$. The relevant axiom of $Frame(\mathbf{P})$ is the backward projection axiom:

$$\mathit{load} \otimes \mathbf{neg}\, \mathit{loaded} \rightarrow \mathbf{neg}\, \mathit{bullet} \otimes \mathit{load} \tag{3.10}$$

Now:

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathbf{neg}\, \mathit{loaded}$	by the inference rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2, \mathbf{d}_3 \vdash \mathit{load}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathit{bullet}$	by the inference rule 3 (Forward projection), the instance (3.10) of the backward projection axiom, and the two sequents above

The required conclusion now follows from the soundness of $\mathcal{F}$. □

3.4.3. Example. [Turkey Shoot 3, continued] The problem in Example 3.2.5 was to be able to prove $\mathbf{P}, \mathbf{d}_1 \models \mathbf{neg}\, \mathit{alive}$. We show how to prove $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathbf{neg}\, \mathit{alive}$ using the inference system $\mathcal{F}$. The relevant instance of the axioms in $Frame(\mathbf{P})$ is the Backward disablement axiom:

$$\mathit{inertial}(\mathit{alive}) \wedge \mathbf{neg}\, \mathit{loaded} \otimes \mathit{shoot} \otimes \mathbf{neg}\, \mathit{alive} \rightarrow \mathbf{neg}\, \mathit{alive} \otimes \mathit{shoot} \tag{3.11}$$

Recall that we assume that $\mathcal{S}$ contains the premise $\mathbf{d}_1 \rhd \mathit{inertial}(\mathit{alive})$. Next:

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathit{alive}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2 \vdash \mathit{shoot}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathbf{neg}\, \mathit{loaded}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathit{inertial}(\mathit{alive})$	by the inference rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathbf{neg}\, \mathit{alive}$	by the inference rule 3, the instance (3.11) of the Backward disablement axiom, and the above sequents

The required conclusion now follows from the soundness of $\mathcal{F}$. □

3.4.4. Example. [Turkey Shoot 4, continued] The problem in Example 3.2.6 was to be able to prove

$$\mathbf{P}, \mathbf{d}_1\text{---} \models \mathit{hunt} \otimes \mathbf{neg}\ \mathit{alive} \tag{3.12}$$

We show how to prove (3.12) using the inference system $\mathcal{F}$. The relevant instances of the axioms in $Frame(\mathbf{P})$ are:

$$inertial(\mathit{bird\_in\_range}) \wedge \mathit{bird\_in\_range} \otimes \mathit{hide} \rightarrow \mathit{hide} \otimes \mathit{bird\_in\_range} \tag{3.13}$$

Next:

(1)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2 \vdash \mathit{find\_location}$	by rule 2 (Premise rule)
(2)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathit{bird\_in\_range}$	by the inference rule 3 , (Forward projection) and sequent (1)
(3)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2, \mathbf{d}_3 \vdash \mathit{hide}$	by rule 2 (Premise rule)
(4)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash hidden$	by the inference rule 3 , (Forward projection) and sequent (3)
(5)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash inertial(\mathit{bird\_in\_range})$	by rule 2 (Premise rule)
(6)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathit{bird\_in\_range}$	by the inference rule 3, (Forward projection) the instance (3.13) of the Forward Inertia axiom and sequents (2) an (4)
(7)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash hidden \otimes \mathit{bird\_in\_range}$	by the inference rule 4 , (Sequencing) and sequents (4) and (6)
(8)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash correct\_location$	by the inference rule 1a , (Applying transaction definitions) and sequent (7)
(9)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathit{loaded}$	by rule 2 (Premise rule)

(10)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3, \mathbf{d}_4 \vdash$ *shoot*	by rule 2 (Premise rule)
(11)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_4 \vdash \mathbf{neg}$ *alive*	by the inference rule 3 , (Forward projection), and sequents (9) and (10)
(12)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \vdash$ *find_location*$\otimes$ *hide* $\otimes$ *correct_location* $\otimes$ *loaded* $\otimes$ *shoot*	by the inference rule 4 (Sequencing) (4 times) and sequents (1),(3),(8),
(13)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \vdash$ *hunt*	(9), and (10), and by inference rule 1a (Applying transaction, definitions) and sequent (12)
(14)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \vdash$ *hunt* $\otimes \mathbf{neg}$ *alive*	by the inference rule 4, (Sequencing), and sequents (13) and (11)

The required conclusion now follows from the soundness of $\mathcal{F}$ and the definition of entailment in $\mathcal{TR}$. □

3.4.5. Example. [Health Insurance, continued #2] The issue in Example 3.2.2 was to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \texttt{---} \models$ *do_cmplnt_test*$(pr, m, s) \otimes$ *negative*(pr, m). We now show a proof for this statement using the inference system $\mathcal{F}$. We assume that all fluents are inertial in every state. For convenience, we show the relevant instances of the axioms in $Frame(\mathbf{P})$ here:

(a) *inertial*(*finished*$(m, pr)) \otimes \mathbf{neg}$ *finished*$(m, pr) \otimes$ *rcv_consent*$(m, pr) \rightarrow$ *rcv_consent*$(m, pr) \otimes \mathbf{neg}$ *finished*(m, pr) $(ForwardInertia)$

(b) *inertial*(*dr(s)*)$\otimes$*dr(s)*$\otimes$*rcv_consent*$(m, pr) \rightarrow$ *rcv_consent*$(m, pr) \otimes$*dr(s)* $(ForwardInertia)$

The derivation is shown below:

(1)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash$ *patient*$(m) \otimes$ *need_consent*(pr)	by the inference rule Premise, App. tran. def. and Sequencing
(2)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash$ *rcv_consent*$(m, pr) \otimes$ $\otimes$*consent*(m, pr)	by the rule Premise, sequent (1), Forward Projection, and Sequencing
(3)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash$ *dr*(s)	by rule Premise and Forward projection, using instance (b) of the Unrelateness axiom

(4)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2\mathbf{d}_3 \vdash$ *do_presc*$(pr, m, s) \otimes$ *presc*(pr, m, s)	by sequent (3), rules Forward Projection and Sequencing
(5)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash$ **neg** *finished*(m, pr) **neg** *matching*(m, pr)	by rule Premise, Forward, Projection and instance (a) of Unrelateness axiom
(6)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3, \mathbf{d}_4 \vdash \otimes$*do_t*$(pr, m, s) \otimes$*negative*$(pr, m)$	by the above sequent (5) and rule Premise, Forward Projection, and Sequencing
(7)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \vdash$ *do_cmplnt_test*$(pr, m, s) \otimes$ *negative*(pr, m)	by the rule Applying Transaction Definitions and the sequents (2),(4),(6)

The required conclusion now follows from the soundness of $\mathcal{F}$ and the definition of entailment in $\mathcal{TR}$. □

In the rest of this section we will show that $\mathcal{TR}^{PAD}$ generalizes Horn-$\mathcal{TR}^{-}$. This implies that the frame axioms in the action theory *behave as expected* in the relational case. That is, they correctly model the inertia laws behind Horn-$\mathcal{TR}^{-}$. Furthermore, the results in Chapter 4 guarantee that the frame axioms introduced above are correct.

First we define a $\mathcal{TR}^{PAD}$ specification that corresponds to a serial-Horn program $\mathbf{P}$ with the initial database $\mathbf{D}$, which we will denote by $(\mathbf{P}, \mathbf{D})$.

3.4.6. DEFINITION. [Relational specifications for serial-Horn programs] A $\mathcal{TR}^{PAD}$ specification $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ is a ***relational specification*** of a serial-Horn program $(\mathbf{P}, \mathbf{D})$ if and only if:

Initial State for every ground base `fluent`-literal f such that $f \in \mathbf{D}$, $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ has these premise formulas:

$$\mathbf{d}_0 \rhd f$$
$$\mathbf{d}_0 \rhd inertial(f)$$

Transaction Base

$$\begin{aligned}\mathbf{Q} = \ & \mathbf{P} \\ & \cup \{insert(f) \rightarrow insert(f) \otimes f \mid \\ & \qquad \text{for every ground base } \texttt{fluent}\text{-literal } f\} \\ & \cup \{delete(f) \rightarrow delete(f) \otimes \mathbf{neg}\, f \mid \\ & \qquad \text{for every ground base } \texttt{fluent}\text{-literal } f\}\end{aligned}$$

Plus the *action theory* of $\mathbf{Q}$.

Transitions for every elementary action α, and sequence r of elementary action, $\mathcal{S}$ contains run-premises of the form:

$$\mathbf{d}_{0,r} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_{0,r,\alpha}$$

In addition, we assume that every ground base fluent is inertial in every state.

3.4.7. Definition. [Correspondence of states] Let $(\mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Given a state identifier $\mathbf{d}$ in $\mathcal{L}_{\mathcal{TR}^{PAD}}$, let $\mathbf{D}(\mathbf{d})$ denote the following set of database fluents in the language $\mathcal{L}_{\mathcal{TR}^-}$ of Transaction Logic:

$$\mathbf{D}(\mathbf{d}) \;=\; \{f \mid f \text{ is a ground } \textit{base } \texttt{fluent}\text{-term such that} \mathbf{P}, \mathcal{S}, \mathbf{d} \models f\}$$

□

3.4.8. Proposition (State consistency and completeness). *Let* $(\mathbf{P}, \mathbf{D})$ *be a Horn-*$\mathcal{TR}^-$ *program and* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$*. Let* α *be an action and* $\mathbf{d}_1, \mathbf{d}_2$ *be state identifiers such that* $\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \alpha$*. If* $\mathbf{D}(\mathbf{d}_1)$ *is consistent then so is* $\mathbf{D}(\mathbf{d}_n)$*. If, in addition,* $\mathbf{D}(\mathbf{d}_1)$ *is complete then so is* $\mathbf{D}(\mathbf{d}_n)$*.*

2. Proof. *See Appendix B.* □

3.4.9. Theorem (Soundness). *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$ *and* h *a serial goal. Suppose that* $\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_n \models h$*. Then there are relational database states* $\mathbf{D}_1, \ldots, \mathbf{D}_{n-1}$ *(in* $\mathcal{L}_{\mathcal{TR}}$*) such that*

$$\mathbf{P}, \mathbf{D}, \mathbf{D}_1 \ldots \mathbf{D}_{n-1}, \mathbf{D}(\mathbf{d}_n) \models h$$

where $\mathbf{D}(\mathbf{d}_n)$ *is as in Definition 3.4.7.*

3. Proof. *See Appendix B.* □

3.4.10. Theorem (Completeness). *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$*, and* h *a serial goal. Suppose that* $\mathbf{P}, \mathbf{D}, \ldots \mathbf{D}_n \models h$*. Then there are state identifiers* $\mathbf{d}_1 \ldots \mathbf{d}_n$ *such that* $\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_n \models h$*.*

4. Proof. *See Appendix B.* □

3.5 Reducing Serial-Horn $\mathcal{TR}^{-}$ to Logic Programming

Before reducing $\mathcal{TR}^{PAD}$ to logic programming, we first tackle a simplified case: defining a reduction of the serial-Horn subset of $\mathcal{TR}^{-}$ to sorted Horn logic programming and prove its soundness and completeness.

The serial-Horn subset of $\mathcal{TR}^{-}$ uses only serial-Horn clauses and relational data and transition oracles. This means that, in this section, database states will be collections of $\mathcal{TR}$-fluents, i.e., facts or explicitly negated facts (e.g., like $bird(Tweety)$ or $\mathbf{neg}\, bird(John)$) and the elementary update operations are $insert(f)$ and $delete(f)$, where f is a fluent.

Given a language $\mathcal{L}_{\mathcal{TR}}$ of Transaction Logic, the corresponding language $\mathcal{L}_{LP}$ of the target logic program is a sorted language with the sorts `state`, `fluent`, `action`, `constant`, and an infinite set of variables for each sort. In addition, we assume that the sort of fluents is contained in the sort of actions so any `fluent`-variable is also an `action`-variable and `fluent`-terms are allowed wherever `action`-terms are. Recall that in Transaction Logic fluents act as trivial actions that do not change the current state. We will see that the same holds in the LP reduction.

In addition, we assume that the set of all fluent predicates is partitioned into ***base fluents*** and ***derived fluents***. Base fluents can appear only as facts, while derived fluents can appear in the heads of rules, but they cannot appear as facts.

$\mathcal{L}_{LP}$ has several distinguished predicates and function symbols, which play a special role in the reduction. The three distinguished predicates are:

- *Hold* with the signature `fluent` $\times$ `state`
- *Inertial* with the signature `fluent` $\times$ `action`
- *Execute* with the signature `action` $\times$ `state` $\times$ `state`

$\mathcal{L}_{LP}$ has no other predicates. The distinguished function symbols are as follows:

- *Result* with the signature `fluent` $\times$ `state` $\rightarrow$ `state`
- s_0, a constant of sort `state`
- *insert* with the signature `fluent` $\rightarrow$ `action`
- *delete* with the signature `fluent` $\rightarrow$ `action`
- **neg** with the signature `fluent` $\rightarrow$ `fluent`
- $\Diamond$ with the signature `action` $\rightarrow$ `action`

- $\otimes$ with the signature `action` $\times$ `action` $\rightarrow$ `action`

It is worth noticing that in this section the predicate *Inertial* is binary, and not unary as in the previous sections. That is because we specify which fluent is inertial with respect to which action. Intuitively, the fact $Inertial(f, a)$ should be understood as "f is not affected by a". In $\mathcal{TR}^{PAD}$, the dependence of the inertia of a fluent with respect to an action is built into the action theory.

For convenience, we will write the function symbols **neg** and $\Diamond$ using the prefix notation and $\otimes$ using the infix notation.

In addition to the distinguished symbols, the predicate and function symbols of the language $\mathcal{L}_{\mathcal{TR}}$ have corresponding function symbols in $\mathcal{L}_{LP}$ as explained next:

- For each n-ary predicate symbol $p \in \mathcal{P}_{fluent}$ in $\mathcal{L}_{\mathcal{TR}}$, $\mathcal{L}_{LP}$ has an n-ary function symbol p (with the same name) with the signature `constant` $\times$ $\ldots \times$ `constant` $\rightarrow$ `fluent`.

- For each n-ary predicate symbol $p \in \mathcal{P}_{action}$ in $\mathcal{L}_{\mathcal{TR}}$, $\mathcal{L}_{LP}$ has an n-ary function symbol p with the signature `constant`$\times \ldots \times$`constant` $\rightarrow$ `action`.

- For each n-ary function symbol $f \in \mathcal{F}$ in $\mathcal{L}_{\mathcal{TR}}$, $\mathcal{L}_{LP}$ has an n-ary function symbol f with the signature `constant` $\times \ldots \times$ `constant` $\rightarrow$ `constant`.

The terms that have *insert* and *delete* as the outer-most symbols are called *elementary actions*. All other terms of sort `action` are *complex actions*.

A pair of ground fluents f, g are said to be ***unrelated*** if $f \neq g$ and $f \neq \mathbf{neg}\, g$ (recall that $\mathbf{neg}\,\mathbf{neg}\, g = g$, by convention). Recall that a *base* fluent is one that can occur only in facts.

Next we list the rules that constitute the reduction of serial-Horn Transaction Logic to logic programming, which we will call *LP-reduction*. This set of rules depends on the input transaction base $\mathbf{P}$ and the initial database state $\mathbf{D}$, so this set will be denoted by $\Gamma(\mathbf{P}, \mathbf{D})$.

To avoid repeating the same statements again and again, we will use the following conventions about variables: S, S_1, S_2, ... denote `state`-variables; A, A_1, A_2, etc., will be used to denote `action`-variables; and F, F_1, F_2, etc., will stand for `fluent`-variables. The rules that belong to the reduction $\Gamma(\mathbf{P}, \mathbf{D})$ can now be formulated as follows:

Initial For each fluent $f \in \mathbf{D}$, $\Gamma(\mathbf{P}, \mathbf{D})$ has the fact $Holds(f, s_0)$.

Unfolding For each $\alpha \leftarrow \beta \in \mathbf{P}$, $\Gamma(\mathbf{P}, \mathbf{D})$ has the rule

$$Execute(\alpha, S_1, S_2) \leftarrow Execute(\beta, S_1, S_2)$$

Sequencing $\Gamma(\mathbf{P}, \mathbf{D})$ has the rule

$$Execute(A_1 \otimes A_2, S_1, S_2) \leftarrow Execute(A_1, S_1, S), Execute(A_2, S, S_2)$$

Hypothetical $Execute(\Diamond A, S, S) \leftarrow Execute(A, S, S_1)$.

Effect+ $Holds(F, Result(insert(F), S))$.

Effect- $Holds(\mathbf{neg}\, F, Result(delete(F), S))$.

Query For every ground *base* `fluent`-term f, $\Gamma(\mathbf{P}, \mathbf{D})$ includes:
$Execute(f, S, S) \leftarrow Holds(f, S)$

Frame Axiom $\Gamma(\mathbf{P}, \mathbf{D})$ includes the following rule

$$Holds(F, S_2) \leftarrow Holds(F, S_1), Execute(A, S_1, S_2), Inertial(F, A)$$

Inertial For each pair of *unrelated base* `fluent`-terms f and g:

$$Inertial(f, insert(g))$$
$$Inertial(f, delete(g))$$
$$Inertial(f, \Diamond A)$$
$$Inertial(F, A_1 \otimes A_2) \leftarrow Inertial(F, A_1), Inertial(F, A_2)$$

Execution $Execute(\alpha, S, Result(\alpha, S))$ for each *elementary* action α.

It is easy to see from the above that, for any serial-Horn transaction base $\mathbf{P}$, the reduction $\Gamma(\mathbf{P}, \mathbf{D})$ is a regular Horn logic program. By the well-known result from [78], it has a unique least Herbrand model, which can be computed via a repeated exhaustive application of the rules in $\Gamma(\mathbf{P}, \mathbf{D})$.

3.5.1. Definition. [Consistency and completeness of `state`-terms] Let $\Gamma(\mathbf{P}, \mathbf{D})$ be the LP reduction of a serial-Horn $\mathcal{TR}$ program $(\mathbf{P}, \mathbf{D})$ and let s be a ground `state`-term. We say that s is **complete** if and only if for any ground *base* `fluent`-term f

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, s) \text{ or } \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(\mathbf{neg}\, f, s)$$

We will say that s is **consistent** if and only if there is no ground *base* `fluent`-term f such that both of the following hold:

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, s) \text{ and } \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(\mathbf{neg}\, f, s)$$

□

We will now establish a number of properties of the LP-reduction.

3.5.2. Proposition (State consistency and completeness). *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a relational serial-Horn Transaction Logic program* $(\mathbf{P}, \mathbf{D})$. *Let* s *and* $\hat{s}$ *be ground* `state`*-terms such that* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, s)$ *holds, where* α *is a ground* `action`*-term. If* $\hat{s}$ *is consistent then so is* s. *If, in addition,* $\hat{s}$ *is complete then* s *is also complete.*

5. Proof. *See Chapter C.*

3.5.3. Definition. [Correspondence between states in $\mathcal{L}_{LP}$ and $\mathcal{L}_{\mathcal{TR}}$] Given a ground `state`-term t in $\mathcal{L}_{LP}$, let $\mathbf{D}(t)$ denote the following set of database fluents in the language $\mathcal{L}_{\mathcal{TR}}$ of Transaction Logic:

$$\mathbf{D}(t) \;=\; \{f \mid f \text{ is a ground } \textit{base } \texttt{fluent}\text{-term such that } \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, t)\} \quad \square$$

3.5.4. Theorem (Soundness). *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a relational serial-Horn* $\mathcal{TR}$ *program* $(\mathbf{P}, \mathbf{D})$ *and suppose that* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, s)$, *where* $\hat{s}$ *and* s *are ground* `state`*-terms and* $\hat{s}$ *is consistent. Then there exist relational database states* $\mathbf{D}_1, \ldots, \mathbf{D}_n$ *(in* $\mathcal{L}_{\mathcal{TR}}$*) such that*

$$\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n\mathbf{D}(s) \;\models\; \alpha$$

where $\mathbf{D}(\hat{s})$ *and* $\mathbf{D}(s)$ *are as in Definition 3.5.3.*

6. Proof. *See Chapter C.*

3.5.5. Theorem (Completeness). *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a relational serial-Horn* $\mathcal{TR}$ *program* $(\mathbf{P}, \mathbf{D})$. *Suppose* $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_n\bar{\mathbf{D}} \models \alpha$, *where* $\hat{\mathbf{D}} = \mathbf{D}(\hat{s})$ *for some consistent ground* `state`*-term* $\hat{s}$. *Then there is a consistent ground* `state`*-term* $\bar{s}$ *such that* $\bar{\mathbf{D}} = \mathbf{D}(\bar{s})$ *and* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.[4]

7. Proof. *See Chapter C.*

3.6 Reducing $\mathcal{TR}_D^{PAD}$ to Logic Programming

In this section we define a reduction for a large fragment of $\mathcal{TR}^{PAD}$, which we call *definite* $\mathcal{TR}^{PAD}$, $\mathcal{TR}_D^{PAD}$, to sorted Horn logic programming, and prove its soundness and completeness. This reduction provides an easy way to implement and experiment with the formalism.

The only difference between $\mathcal{TR}_D^{PAD}$ and $\mathcal{TR}^{PAD}$ is that $\mathcal{TR}_D^{PAD}$ allows neither *non-deterministic* nor *converging premises* run-premises and it requires the set of premises to be *well-founded*. These notions are defined next.

[4] $\mathbf{D}(\hat{s})$ and $\mathbf{D}(\bar{s})$ were introduced in Definition 3.5.3.

A set of run-premises is ***converging*** if it has a pair of run-premises that shares the same final state. For instance,

$$\mathbf{d}_1 \overset{shoot}{\rightsquigarrow} \mathbf{d}_2$$
$$\mathbf{d}_3 \overset{load}{\rightsquigarrow} \mathbf{d}_2$$

Two run-premises for the same partially defined action, α, are ***non-deterministic*** if they have the same initial state but different final states. For instance the following run-premises are non-deterministic:

$$\mathbf{d} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_1$$
$$\mathbf{d} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$$

We should note that the restriction about determinism of the premises concerns *pda*s only: *complex* actions defined by serial-Horn rules *can* be non-deterministic, and $\mathcal{TR}_D^{PAD}$ can represent and deal with them.

We say that a set of premises $\mathcal{S}$ is ***well-founded*** if $\mathcal{S}$ does *not* have an infinite chain of *run*-premises of the form $\mathbf{d}_1 \overset{\alpha_0}{\rightsquigarrow} \mathbf{d}_0$, $\mathbf{d}_2 \overset{\alpha_1}{\rightsquigarrow} \mathbf{d}_1$, $\mathbf{d}_3 \overset{\alpha_2}{\rightsquigarrow} \mathbf{d}_2$, ..., for any states $\mathbf{d}_0$, $\mathbf{d}_1$, $\mathbf{d}_2$, ... and *pda*s α_0, α_1, α_2, ... As a special case, this precludes circular *run*-premises. For instance, the set of premises that has the following *run*-premises is not well-founded:

$$\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \quad \mathbf{d}_2 \overset{\beta}{\rightsquigarrow} \mathbf{d}_1$$

As in Section 3.5, all states in $\mathcal{TR}_D^{PAD}$ are relational, i.e., collections of fluents. Given a language $\mathcal{L}_{\mathcal{TR}}$ of $\mathcal{TR}_D^{PAD}$, the target language $\mathcal{L}_{LP}$ for the logic programing reduction of $\mathcal{TR}_D^{PAD}$ is defined as in Section 3.5 except for the set of constants. Recall that the language of the reduct has three distinguished predicate symbols *Holds*, *Inertial*, and *Execute*; and the distinguished function symbols *Result*, $\otimes$, $\diamondsuit$, and **neg**. Section 3.5 had a *single* `state`-constant s_0, but now we will have a unique `state`-constant $s_{\mathbf{d}}$ for each database state $\mathbf{d}$.

Recall that intuitively, the atom $Holds(f, s)$ means that the fluent f holds in state s, and $Execute(\alpha, s_1, s_2)$ means that executing α in s_1 leads to state s_2. The intuition behind **neg**, $\diamondsuit$, $\otimes$ should be clear at this point: they encode negated literals, hypotheticals, and sequencing of actions. The `state`-term $Result(\alpha, s)$ represents the state resulting from executing α in the state s.

The set of LP axioms that constitute the reduction depends on the input transaction base $\mathbf{P}$ as well as on the set of premises $\mathcal{S}$. We denote this reduction by $\Gamma(\mathbf{P}, \mathcal{S})$.

As in Section 3.5, we use the following conventions: S, S_1, S_2, and so on, denote `state`-variables; the symbols A, A_1, A_2, etc., are used for `action`-variables; and F, F_1, F_2, etc., represent `fluent`-variables.

Note that in the PADs the pre- and post-conditions are conjunctions of fluents, and occasionally we will need Boolean combinations of fluents. In these cases, we will be sometimes using the usual De Morgan's laws, such as $\mathbf{neg}\,(f \wedge g) = \mathbf{neg}\,f \vee \mathbf{neg}\,g$, and we postulate that $\vee$ and $\wedge$ are distributive with respect to *Holds*. For example,

$$\begin{aligned} Holds(f_1 \wedge f_2, s) &\equiv Holds(f_1, S) \wedge Holds(f_n, S) \\ Holds(f_1 \vee f_2, s) &\equiv Holds(f_1, S) \vee Holds(f_2, S) \\ Holds(\mathbf{neg}\,(f_1 \wedge f_2), s) &\equiv Holds(\mathbf{neg}\,f_1, S) \vee \\ &\qquad Holds(f\mathbf{neg}_2, S) \\ Holds(\mathbf{neg}\,(f_1 \vee f_2), s) &\equiv Holds(\mathbf{neg}\,f_1, S) \wedge \\ &\qquad Holds(f\mathbf{neg}_2, S) \end{aligned}$$

The reduction $\Gamma(\mathbf{P}, \mathcal{S})$ of a $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$ is defined by the following set of rules and facts. First we define $db2st_{\mathcal{S}}$, as a correspondence between database states and `state`-terms, as follows:

- $db2st_{\mathcal{S}}(\mathbf{d}) = s_{\mathbf{d}}$, if $\mathbf{d}$ occurs in a *run*- or *state*-premise in $\mathcal{S}$ and $\mathcal{S}$ has *no* *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, for some state $\mathbf{d}_0$. Here $s_{\mathbf{d}}$ is the unique $\mathcal{L}_{LP}$ state constant that corresponds to the $\mathcal{TR}_D^{PAD}$ state identifier $\mathbf{d}$ and α is a pda.
- $db2st_{\mathcal{S}}(\mathbf{d}) = Result(\alpha, s)$, if $\mathcal{S}$ has a *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, and $db2st_{\mathcal{S}}(\mathbf{d}_0) = s$.

It is worth noticing that this definition is well-formed, because $\mathcal{S}$ is a well-founded set of premises.

Premises The following facts are added to $\Gamma(\mathbf{P}, \mathcal{S})$ for each premise in $\mathcal{S}$.

- For each *state*-premise $\mathbf{d} \rhd f \in \mathcal{S}$ and any state $s = db2st_{\mathcal{S}}(\mathbf{d})$:

$$Holds(f, s) \quad \in \Gamma(\mathbf{P}, \mathcal{S})$$

- For each *run*-premise $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$ and any state $s = db2st_{\mathcal{S}}(\mathbf{d}_1)$:

$$Execute(\alpha, s, Result(\alpha, s)) \quad \in \Gamma(\mathbf{P}, \mathcal{S})$$

No-op For each database $\mathbf{D}$ such that $db2st_{\mathcal{S}}(\mathbf{D})$ is non empty, and for any state $s = db2st_{\mathcal{S}}(\mathbf{D})$[5]

$$Holds((), s) \quad \in \Gamma(\mathbf{P}, \mathcal{S})$$

[5] Recall that () is an empty conjunction of fluents.

Unfolding For each $\alpha \leftarrow \beta \in \mathbf{P}$, $\Gamma(\mathbf{P}, \mathcal{S})$ has the rule

$$Execute(\alpha, S_1, S_2) \leftarrow Execute(\beta, S_1, S_2)$$

Sequencing $\Gamma(\mathbf{P}, \mathcal{S})$ has the rule

$$Execute(A_1 \otimes A_2, S_1, S_2) \leftarrow Execute(A_1, S_1, S), \; Execute(A_2, S, S_2)$$

Decomposition For every conjunction of `fluent`-terms and hypothetical serial-conjunction g and each conjunct h in g, $\Gamma(\mathbf{P}, \mathcal{S})$ includes the following rule:

$$Execute(h, S, S) \leftarrow Execute(g, S, S)$$

Hypothetical $Execute(\Diamond A, S, S) \leftarrow Execute(A, S, S_1)$.

Query If f is a ground *base* `fluent`-term or the empty conjunction (), then $\Gamma(\mathbf{P}, \mathcal{S})$ includes

$$Execute(f, S, S) \leftarrow Holds(f, S)$$

Forward Projection For each PAD $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \in \mathbf{P}$, $\Gamma(\mathbf{P}, \mathcal{S})$ has the rules:

$$\begin{aligned} Holds(b_3, S) &\leftarrow Execute(b_1, S, S), Execute(\alpha, S, Result(\alpha, S)), \\ &\qquad Execute(b_2, Result(\alpha, S), Result(\alpha, S)) \\ Holds(b_4, Result(\alpha, S)) &\leftarrow Execute(b_1, S, S), Execute(\alpha, S, Result(\alpha, S)), \\ &\qquad Execute(b_2, Result(\alpha, S), Result(\alpha, S)) \end{aligned}$$

Observe that, since b_1, b_2, b_3, and b_4 might be conjunctions of literals, application of De Morgan's laws to these rules may result in conjunctions in the rule heads and disjunctions in the body. However, such rules reduce to regular Horn rules.

Observe that $\Gamma(\mathbf{P}, \mathcal{S})$ contains one kind of LP rule for each inference rule/axiom in $\mathcal{F}$[6] plus one extra rule that interprets fluents as trivial actions that do not change states.

It follows directly from the construction of $\Gamma(\mathbf{P}, \mathcal{S})$ that it is equivalent to a set of Horn rules for any $\mathcal{TR}_D^{PAD}$ transaction base $\mathbf{P}$. Therefore, it has a unique least Herbrand model, which can be computed via a repeated exhaustive application of the rules in $\Gamma(\mathbf{P}, \mathcal{S})$ in a bottom-up fashion.

[6] Forward Projection in $\Gamma(\mathbf{P}, \mathcal{S})$ consists of two kinds of rules, one for the post-condition, and one for the pre-effect.

3.6.1. DEFINITION. [Correspondence between sets of fluents in $\mathcal{L}_{LP}$ and $\mathcal{L}_{\mathcal{TR}^{PAD}}$] Given a ground `state`-term s in $\mathcal{L}_{LP}$, we define $\mathbf{D}(s)$ to be the following set of database fluents in the language $\mathcal{L}_{\mathcal{TR}^{PAD}}$ of Transaction Logic:

$$\{f \mid f \text{ is a ground } \texttt{fluent}\text{-term such that } \Gamma(\mathbf{P}, \mathcal{S}) \models \mathit{Holds}(f, s)\}$$

□

The soundness theorem uses the following partial function from `state`-terms to database states. It relies on the fact that $\mathcal{S}$ has no non-deterministic *run*-premises in $\mathcal{TR}_D^{PAD}$ and that it is well-founded. Let *st2db* be the partial function defined as follows

3.6.2. DEFINITION. Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a relational $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. We define a partial function *st2db* from `state`-terms to database state identifiers as follows:

- $\mathit{st2db}(s_{\mathbf{d}}) = \mathbf{d}$, if $\mathbf{d}$ occurs in a *run*- or *state*-premise in $\mathcal{S}$ and $\mathcal{S}$ has no *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$ for some $\mathbf{d}_0$. If $\mathbf{d}$ does *not* occur in any *run*- or *state*-premise in $\mathcal{S}$, then $\mathit{st2db}(s_{\mathbf{d}})$ is undefined.
  Here $s_{\mathbf{d}}$ is the unique state constant that corresponds to the database state $\mathbf{d}$ and α is a pda.

- $\mathit{st2db}(\mathit{Result}(\alpha, s)) = \mathbf{d}$, if $\mathit{st2db}(s)$ exists and $\mathit{db2st}(s) \overset{\alpha}{\rightsquigarrow} \mathbf{d} \in \mathcal{S}$. Otherwise, $\mathit{st2db}(\mathit{Result}(\alpha, s))$ is undefined. □

$\mathit{st2db}(s)$ is uniquely defined and thus well-formed because $\mathcal{S}$ is well-founded and has no non-deterministic *run*-premises.

3.6.3. THEOREM (SOUNDNESS). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}_D^{PAD}$ program $(\mathbf{P}, \mathcal{S})$. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models \mathit{Execute}(\alpha, s_1, s_2)$, where s_1 and s_2 are ground* `state`*-terms and α an action. Then there are relational database states $\mathbf{d}_1, \ldots, \mathbf{d}_2$ in $\mathcal{L}_{\mathcal{TR}}$ such that the following holds:*

(1) $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \alpha$ *(2)* $\mathbf{d}_1 = \mathit{st2db}(s_1)$, $\mathbf{d}_2 = \mathit{st2db}(s_2)$

(3) $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models D(s_1)$ *(4)* $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models D(s_2)$

where $D(s)$ denotes the set of all database fluents f in the language $\mathcal{L}_{\mathcal{TR}^{PAD}}$, such that $\Gamma(\mathbf{P}, \mathcal{S}) \models \mathit{Holds}(f, s)$.

8. PROOF. *See Appendix D.* □

3.6.4. THEOREM (COMPLETENESS). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP reduction of a $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$. Then the following holds:*
– If $n = 1$, and there is a `state`*-term s_1 such that $\mathit{db2st}_{\mathcal{S}}(\mathbf{d}_1) = s_1$, then ϕ is a conjunction of fluents and hypotheticals and:*

$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_1)$$

– *If* $n > 1$, *and there are ground* `state`*-terms* s_1, s_2 *such that* $db2st_{\mathcal{S}}(\mathbf{d}_1) = s_1$ *and* $db2st_{\mathcal{S}}(\mathbf{d}_n) = s_2$, *then:*

$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$$

9. Proof. *See Appendix D.* □

In plain English, together these theorems say that every execution of an action in $\Gamma(\mathbf{P}, \mathcal{S})$ has a similar execution in $\mathcal{TR}_D^{PAD}$, and vice versa.

3.7 $\mathcal{TR}^{PAD}$ with Default Negation

In this section, we extend $\mathcal{TR}^{PAD}$ with *default* negation (a.k.a. negation as failure). Default negation has become a central ingredient in the design of logic programming languages, databases [77, 37], truth maintenance system [47], etc. Default negation allows a logic system to conclude the negation of any atom that the system unsuccessfully finishes exploring all possible proofs. That is, if we fail to prove that a fact is true, by *default*, we assume it is false.

The syntax of $\mathcal{TR}^{PAD}$ with default negation is as in $\mathcal{TR}^{PAD}$ (c.f. Section 3.1). As before, the symbol **neg** will be used to represent the explicit negation, but now we allow also the symbol **not** to represent default negation. These two symbols are applicable to fluents only. A fluent literal is either an atomic fluent or it has one of the following negated forms:

$$\mathbf{neg}\, f, \quad \mathbf{not}\, f, \quad \mathbf{not}\ \mathbf{neg}\, f$$

where f is an atomic fluent. Literals that do not mention **not** are said to be **not**-free. We allow **not**-literals to occur only as a conjunct in the pre/post-conditions of PADs and in the body of Horn rules. That is, effects and pre-effects of PADs, the head of Horn rules, and premises are **not**-free.

Observe that it is not necessary the case that explicit negation implies default negation. Therefore one should not use explicit negation as if this implication held. For instance, suppose we have the following transaction base:

$$sunny \leftarrow \mathbf{not}\, rainy$$
$$rainy \leftarrow \mathbf{not}\, sunny$$

If one assumes that explicit negation implies default negation, we could have a database where **neg** *sunny* holds, and expect to infer that *rainy* is true in such state. However, in our formalism, *sunny* and *rainy* will be undefined under the well-founded semantics. This limitation in our approach does not produce 'wrong' answers, but it is a cautious approach, that is, not expressive enough to cover some cases, such as the example above.

In this section, for the sake of clarity, we split the transaction base in two: the ***transaction base*** and the ***action base***. In the transaction base we leave *only* the serial-Horn rules. The $\mathcal{TR}^{PAD}$ ***action base***, contains the set of PADs. A $\mathcal{TR}^{PAD}$ **specification** is a tuple $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ where $\mathcal{E}$ is a $\mathcal{TR}^{PAD}$ action base, $\mathbf{P}$ is a $\mathcal{TR}^{PAD}$ transaction base, and $\mathcal{S}$ is a set of premises.

Semantics.

This semantics uses three truth values, $\mathbf{u}$, $\mathbf{t}$ and $\mathbf{f}$, which stand for *true*, *false*, and *undefined* and are ordered as follows: $\mathbf{f} < \mathbf{u} < \mathbf{t}$. In addition, we will use the following operator $\sim$: $\sim \mathbf{t} = \mathbf{f}$, $\sim \mathbf{f} = \mathbf{t}$, $\sim \mathbf{u} = \mathbf{u}$.

A **database state D** (or just a *state,* for short) is a set of ***ground*** (i.e., variable-free) fluent literals.

The semantics is based on the notion of *path structures*.

3.7.1. Definition. [Three-valued Partial Herbrand Interpretation] A **partial Herbrand interpretation** is a mapping $\mathcal{H} : \mathcal{B} \mapsto \{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ that assigns a truth value, $\mathbf{f}, \mathbf{u}$, or $\mathbf{t}$, to every formula ϕ in $\mathcal{B}$. □

Recall that a central feature in the semantics of $\mathcal{TR}$ is the notion of *execution paths*, since $\mathcal{TR}$ formulas are evaluated over paths and *not* over states like in temporal logics.

3.7.2. Definition. [Three-valued Herbrand Path Structure] A **Herbrand path structure** is a mapping $\mathbf{I}$ that assigns a partial Herbrand interpretation to every path. That is, for any path π, $\mathbf{I}(\pi)$ is an interpretation. So, for instance, $\mathbf{I}(\pi)(f)$ is a truth value for any literal f. This mapping must satisfy the restriction that for each ground base fluent f and database state $\mathbf{D}$:

$$\mathbf{I}(\langle \mathbf{D} \rangle)(f) = \mathbf{t} \text{ if } f \in \mathbf{D} \quad \text{and} \quad \mathbf{I}(\langle \mathbf{D} \rangle)(\mathbf{neg}\, f) = \mathbf{t} \text{ if } \mathbf{neg}\, f \in \mathbf{D}$$

where $\langle \mathbf{D} \rangle$ is a path that contains only one state, $\mathbf{D}$.

In addition, $\mathbf{I}$ includes a mapping of the form:

$$\Delta_{\mathbf{I}} : \textit{ State identifiers} \longrightarrow \textit{ Database states}$$

which associates states to state identifiers. We will usually omit the subscript. □

In the remainder of this section we will consider ground rules and PADs only. We can make this assumption without loosing generality because all the variables in a rule are considered to be universally quantified. In addition, we assume that the language includes the distinguished propositional constants $\mathbf{t}^{\pi}$, and $\mathbf{u}^{\pi}$ for each $\mathcal{TR}$ path π. Observe that since there is an infinite number of paths, there is an infinite number of such constants. Informally, $\mathbf{t}^{\pi}$ ($\mathbf{u}^{\pi}$) is a

proposition that has the truth value $\mathbf{t}$ (respectively $\mathbf{u}$) only on the path π, and it is false on every other path. That is, $\mathbf{I}(\pi')(\mathbf{t}^{\pi}) = \mathbf{t}$ (respectively $\mathbf{I}(\pi')(\mathbf{u}^{\pi}) = \mathbf{u}$) if and only if $\pi = \pi'$.

The following definition formalizes the idea that truth of $\mathcal{TR}$ formulas is defined on paths. To ease the intuition of this definition, we will explain the idea behind the satisfaction of a serial conjunction formula of the form $\phi \otimes \psi$. A formula $\phi \otimes \psi$ is <u>true</u> in a in a path π, with respect to path structure $\mathbf{I}$ if and only if there is a split $\pi_1 \circ \pi_2$ of π such that ϕ is true in π_1 and ψ is true in ϕ_2 with respect to $\mathbf{I}$. If for every split $\pi_1 \circ \pi_2$ of π, ϕ is false in π_1 and ψ is false in ϕ_2 with respect to $\mathbf{I}$, then $\phi \otimes \psi$ is **false** in a in π. Otherwise, $\phi \otimes \psi$ is **undefined** in π with respect to path structure $\mathbf{I}$.

3.7.3. Definition. [Satisfaction] Let $\mathbf{I}$ be a Herbrand path structure, π be a path, f a ground **not**-free literal, and G, G_1, G_2 ground serial goals. We define **truth valuations** with respect to the path structure $\mathbf{I}$ as follows:

- $\mathbf{I}(\pi)(f)$ was already defined as part of the definition of Herbrand path structures.
- $\mathbf{I}(\pi)(\phi \otimes \psi) = max\{min(\mathbf{I}(\pi_1)(\phi), \mathbf{I}(\pi_2)(\psi) \mid \pi = \pi_1 \circ \pi_2\}$
- $\mathbf{I}(\pi)(G_1 \wedge G_2) = min(\mathbf{I}(\pi)(G_1), \mathbf{I}(\pi)(G_2))$
- $\mathbf{I}(\pi)(\mathbf{not}\, \phi) = \sim \mathbf{I}(\pi)(\phi)$
- $\mathbf{I}(\pi)(\diamond\phi) = \begin{cases} max\{\mathbf{I}(\pi')(\phi) \mid \pi' \text{ is a path that starts at } \mathbf{D}, \ \} & \text{if } \pi = \langle \mathbf{D} \rangle \\ \mathbf{f} & \text{otherwise} \end{cases}$
- $\mathbf{I}(\pi)(f \leftarrow G) = \mathbf{t}$ iff $\mathbf{I}(\pi)(f) \geq \mathbf{I}(\pi)(G)$
- $\mathbf{I}(\pi)(b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4) = \mathbf{t}$ iff π has the form $\langle \mathbf{D}_1, \mathbf{D}_2 \rangle$, $\mathbf{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(\alpha) = \mathbf{t}$, and the following holds:
$$\mathsf{min}\{\mathsf{min}\{\mathbf{I}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b_1\}, \mathsf{min}\{\mathbf{I}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b_2\}\}$$
$$\leq$$
$$\mathsf{min}\{\mathsf{min}\{\mathbf{I}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b_3\}, \mathsf{min}\{\mathbf{I}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b_4\}\}$$

We write $\mathbf{I}, \pi \models \phi$ and say that ϕ is **satisfied** on path π in the path structure $\mathbf{I}$ if $\mathbf{I}(\pi)(\phi) = \mathbf{t}$. □

3.7.4. Definition. [Model] A path structure, $\mathbf{I}$, is a **model of a formula** ϕ if $\mathbf{I}, \pi \models \phi$ for every path π. In this case we write $\mathbf{I} \models \phi$. A path structure, $\mathbf{I}$, is a model of a set of formulas if it is a model of every formula in the set. A path structure, $\mathbf{I}$, is a **model of a premise-statement** σ iff:

- σ is a run-premise of the form $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ and $\mathbf{I}, \langle \mathbf{d}_1 \mathbf{d}_2 \rangle \models \alpha$; or

- σ is a state-premise $\mathbf{d} \rhd f$ and $\mathbf{I}, \langle \mathbf{d} \rangle \models f$.

$\mathbf{I}$ is a **model** of a specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ if $\mathbf{I}$ is an interpretation for $\mathcal{E}$ and it satisfies every rule in $\mathbf{P}$ and every premise in $\mathcal{S}$. □

3.7.5. Example. Consider our Health Insurance example with the following additional features:

1. A doctor can do a prescription only the prescription is not *inconsistent* with the current regulations.
2. A person who gets a prescription is unhealthy.
3. A person who agrees to get a DNA test is assumed to be healthy.

To represent this we introduce the new fluent *inconsistent*. The modified PAD for *do_presc* is as follows:

$$dr(D) \otimes do\_presc(T, P, D) \otimes \mathbf{not}\ inconsistent \rightarrow do\_presc(T, P, D) \otimes presc(D, P, T)$$

The features 2 and 3 are encoded by the following rules:

$$\mathbf{neg}\ healthy(P) \leftarrow presc(D, P, T)$$
$$healthy(P) \leftarrow dna\_t(T) \wedge consent(P, T)$$

Suppose there is an initial state $\mathbf{d}_0$ where Mark gave his consent to receive a PCR-DNA, and Dr. Smith prescribed such test. This is represented by the premises below:

$$\mathbf{d}_0 \overset{do\_presc(m,pr,s)}{\rightsquigarrow} \mathbf{d}_1$$
$$\mathbf{d}_0 \rhd dna\_t(pr)$$
$$\mathbf{d}_0 \rhd consent(m, pr)$$

The frame axioms for the problem at hand have the following form:

$$dna\_t(P) \otimes do\_presc(T, P, D) \otimes \mathbf{not}\ \mathbf{neg}\ dna\_t(P) \rightarrow do\_presc(T, P, D) \otimes dna\_t(P)$$
$$consent(A, B) \otimes do\_presc(T, P, D) \otimes \mathbf{not}\ \mathbf{neg}\ consent(A, B) \rightarrow do\_presc(T, P, D) \otimes consent(A, B)$$

Observe that now we modified the frame axioms presented in Section 3.4. Further details will be given in Section 3.8.

Now take an interpretation, $\mathcal{M}_1$, such that $\mathcal{M}_1(\mathbf{d}_1)(inconsistent) = \mathbf{f}$. From the PADs in $\mathcal{E}$ instantiated with s, pr and m, we can conclude that:

$$\mathcal{M}_1(\mathbf{d}_1)(dna\_t(pr)) = \mathbf{t}$$
$$\mathcal{M}_1(\mathbf{d}_1)(consent(m, pr)) = \mathbf{t}$$
$$\mathcal{M}_1(\mathbf{d}_1)(presc(s, m, pr)) = \mathbf{t}$$

and from the rules in the transaction base it follows that $\mathcal{M}_1(\mathbf{d}_1)(\mathit{healthy}(m)) = \mathbf{t}$ and
$\mathcal{M}_1(\mathbf{d}_1)(\mathbf{neg}\ \mathit{healthy}(m)) = \mathbf{t}$. Thus, $\mathbf{d}_1$ is inconsistent in $\mathcal{M}_1$. □

Recall that in classical logic programming based on three-valued models, given two Herbrand *partial* interpretations $\mathbf{N}_1$ and $\mathbf{N}_2$, we say that

- $\mathbf{N}_1 \leq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$. This coincides with set-theoretic *inclusion* and is called the *information ordering*.
- $\mathbf{N}_1 \preceq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_2$ are true in $\mathbf{N}_1$. This is called the *truth ordering*.

3.7.6. Definition. [Order on Path Structures] Let $\mathbf{M}_1$ and $\mathbf{M}_2$ be two Herbrand path structures, then:

- *Information ordering:* $\mathbf{M}_1 \leq \mathbf{M}_2$ if for every path, π, it holds that $\mathbf{M}_1(\pi) \leq^c \mathbf{M}_2(\pi)$.
- *Truth ordering:* $\mathbf{M}_1 \preceq \mathbf{M}_2$ if for every path, π, it holds that $\mathbf{M}_1(\pi) \preceq^c \mathbf{M}_2(\pi)$. □

These two orderings are considerably different. The *truth ordering* minimize the *amount of truth*, by minimizing the atoms that are true and maximizing the atoms that are false on each path. In contrast, the *information ordering* minimizes the amount of information by minimizing both the atoms that are true and false in each path. For instance, the smallest model with respect to $\leq$ for the program $\{\alpha \rightarrow \alpha\}$ is one where α is *undefined* on every path; in contrast, the least model with respect to $\preceq$ is one when α is false on every path.

3.7.7. Example. Consider a path structure $\mathcal{I}_2$ for the specification in Example 3.7.5 that coincides with $\mathcal{I}_1$ in the path $\langle \mathbf{d}_0 \rangle$ but differs in $\langle \mathbf{d}_1 \rangle$ as follows:

$$\mathcal{M}_2(\mathbf{d}_1)(\mathit{dna\_t}(pr)) = \mathbf{u}$$
$$\mathcal{M}_2(\mathbf{d}_1)(\mathit{inconsistent}) = \mathbf{u}$$
$$\mathcal{M}_2(\mathbf{d}_1)((\mathit{consent}(m, pr)) = \mathbf{u}$$
$$\mathcal{M}_2(\mathbf{d}_1)(\mathit{presc}(s, m, pr)) = \mathbf{u}$$

It is not hard to see that $\mathcal{I}_2$ is also a model of our specification, and moreover $\mathcal{I}_2 \preceq \mathcal{I}_1$. □

3.7.8. Definition. [Least Model] A model $\mathbf{M}$ of a specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is **minimal** with respect to $\preceq$ iff for any other model, $\mathbf{N}$, of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, if $\mathbf{N} \preceq \mathbf{M}$ then $\mathbf{N} = \mathbf{M}$. The **least** model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, denoted $\mathrm{LPM}(\mathcal{E}, \mathbf{P}, \mathcal{S})$, is a minimal model that is unique. □

The following definition is key to the notion of well-founded $\mathcal{TR}^{PAD}$ models. It is modeled after [31] with appropriate extensions for PADs.

3.7.9. DEFINITION. [$\mathcal{TR}^{PAD}$-quotient] Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification, and $\mathbf{I}$ a Herbrand path structure. By $\mathcal{TR}^{PAD}$- **quotient** of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ **modulo** $\mathbf{I}$ we mean a new specification, $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{I}}$, which is obtained from $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ by

- Replacing every literal of the form $\mathbf{not}\, b$ in $\mathbf{P} \cup \mathcal{E}$ with

$$\begin{array}{l} \mathbf{t}^{\pi} \text{ for every path } \pi \text{ such that } \mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{t} \\ \mathbf{u}^{\pi} \text{ for every path } \pi \text{ such that } \mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{u} \end{array}$$

- And afterwards removing all the remaining rules and PADs that have a literal of the form $\mathbf{not}\, b$ in the body such that $\mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{f}$ for some path π. □

3.7.10. EXAMPLE. Consider the specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ consisting in the following PADs and rules:

(1)	PAD :	$\alpha \rightarrow \alpha \otimes \mathbf{neg}\, f_1$
(2)	PAD :	$c_1 \otimes \alpha \otimes \mathbf{not}\, \mathit{inconsistent} \rightarrow \alpha \otimes c_1$
(3)	PAD :	$c_2 \otimes \alpha \otimes \mathbf{not}\, \mathit{inconsistent} \rightarrow \alpha \otimes c_2$
(4)	PAD :	$\beta \rightarrow \beta \otimes f_1$
(5)	Fluent rule:	$f_1 \leftarrow c_1 \wedge c_2$
(6)	Fluent rule:	$\mathit{inconsistent} \leftarrow f_1 \wedge \mathbf{neg}\, f_1$
(7)	Action rule:	$\gamma \leftarrow \alpha \otimes \mathbf{not}\, \mathit{inconsistent} \otimes \beta$

where *inconsistent*, f_1, c_1 and c_2 are fluents, and α, β, and γ are actions. In plain English, PAD (1) describes the effect of α on $\mathbf{neg}\, f_1$. PADs (2) and (3) encode the frame axioms for the fluents c_1 and c_2 with respect to α with further qualification that they must not cause inconsistency. PAD (4) describes the effect of β on f_1. The fluent rule (5) defines the fluent f_1 in terms of the fluents c_1 and c_2, and (6) defines the fluent *inconsistent*. Rule (7) defines the action γ. Let $\mathbf{I}$ be the Herbrand path where everything is undefined in every path. The quotient $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{I}}$ is then as follows:

(1)	$\alpha \rightarrow \alpha \otimes \mathbf{neg}\, f_1$	
(2)	$c_1 \otimes \alpha \otimes \mathbf{u}^{\pi} \rightarrow \alpha \otimes c_1$	(multiple copies for all possible paths π)
(3)	$c_2 \otimes \alpha \otimes \mathbf{u}^{\pi} \rightarrow \alpha \otimes c_2$	(again, one per path π)
(4)	$\beta \rightarrow \beta \otimes f_1$	
(5)	$f_1 \leftarrow c_1 \wedge c_2$	
(6)	$\mathit{inconsistent} \leftarrow f_1 \wedge \mathbf{neg}\, f_1$	
(7)	$\gamma \leftarrow \alpha \otimes \mathbf{u}^{\pi} \otimes \beta$	

3.7.11. DEFINITION. [Union of Path Structures] Let $\mathbf{I}_1 < \mathbf{I}_2 < \ldots$ be a (possible infinite) number of path structures. Then we define the infinite union $\mathbf{J} = \mathbf{I}_1 \cup \mathbf{I}_2 \cup \ldots$ as follows:

$$\mathbf{J}(\pi)(f) = \begin{cases} \mathbf{t} & \text{if exists a path structure } \mathbf{I}_n \text{ in the union} \\ & \text{such that } \mathbf{I}_n(\pi)(f) = \mathbf{t} \\ \mathbf{f} & \text{if exists a path structure } \mathbf{I}_n \text{ in the union} \\ & \text{such that } \mathbf{I}_n(\pi)(f) = \mathbf{f} \\ \mathbf{u} & \text{otherwise} \end{cases}$$

For every path π and literal f. □

Observe that this definition is quite intuitive, since the information ordering $\leq$ coincides (modulo the paths) with the with set-theoretic inclusion.

Next, we give a constructive definition of *well-founded models* for $\mathcal{TR}^{PAD}$ specifications in terms of a consequence operator.

3.7.12. DEFINITION. [$\mathcal{TR}^{PAD}$ Immediate Consequence Operator] The **consequence operator**, Γ, for a $\mathcal{TR}^{PAD}$ specification is defined by analogy with the classical case:

$$\Gamma(\mathbf{I}) = \text{LPM}(\frac{(\mathcal{E}, \mathbf{P}, \mathcal{S})}{\mathbf{I}})$$

Suppose $\mathbf{I}_\emptyset$ is the path structure that maps each path π to the empty Herbrand interpretation in which all atoms are undefined. That is, for every path π and literal f, we have $\mathbf{I}_\emptyset(\pi)(f) = \mathbf{u}$. The ordinal powers of the consequence operator Γ are then defined inductively as follows:

- $\Gamma^{\uparrow 0}(\mathbf{I}_\emptyset) = \mathbf{I}_\emptyset$
- $\Gamma^{\uparrow n}(\mathbf{I}_\emptyset) = \Gamma(\Gamma^{\uparrow n-1}(\mathbf{I}_\emptyset))$, if n is a successor ordinal
- $\Gamma^{\uparrow n}(\mathbf{I}_\emptyset) = \bigcup_{j \leq n} \Gamma^{\uparrow j}(\mathbf{I}_\emptyset)$, if n is a limit ordinal □

The operator Γ is monotonic with respect to the $\leq$-order when $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is fixed. Because of this, the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)\}$ has a least fixed point and is computable via transfinite induction.

3.7.13. LEMMA. *The operator Γ is monotonic with respect to the information order relation $\leq$ when $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is fixed. That is,*

$$\textit{If} \quad \mathbf{I} \leq \mathbf{I}' \quad \textit{then} \quad \Gamma(\mathbf{I}) \leq \Gamma(\mathbf{I}')$$

10. PROOF. *See Appendix E.* □

3.7.14. DEFINITION. [Well-founded Model] The **well-founded** model of a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, written $\mathrm{WFM}((\mathcal{E}, \mathbf{P}, \mathcal{S}))$, is defined as the limit of the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)\}$. □

3.7.15. EXAMPLE. Consider the specification,$(\mathcal{E}, \mathbf{P}, \mathcal{S})$, in Example 3.7.10. Suppose that we have the following set of premises in $\mathcal{S}$:

$$\begin{array}{l}\mathbf{d}_1 \rhd c_1\\ \mathbf{d}_1 \rhd c_2\\ \mathbf{d}_1 \rhd f_1\\ \mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2\\ \mathbf{d}_2 \overset{\beta}{\rightsquigarrow} \mathbf{d}_3\end{array}$$

Then, the least model $\mathcal{M}$ of $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{I}}$ has the following properties:

1. $\mathcal{M}(\langle \mathbf{d}_1\mathbf{d}_2\rangle)(\alpha) = \mathbf{t}$
2. $\mathcal{M}(\langle \mathbf{d}_2\mathbf{d}_3\rangle)(\beta) = \mathbf{t}$
3. $\mathcal{M}(\langle \mathbf{d}_1\rangle)(c_1) = \mathbf{t}$
4. $\mathcal{M}(\langle \mathbf{d}_1\rangle)(c_2) = \mathbf{t}$
5. $\mathcal{M}(\langle \mathbf{d}_1\rangle)(f_1) = \mathbf{t}$
6. $\mathcal{M}(\langle \mathbf{d}_2\rangle)(\mathbf{neg}\, f_1) = \mathbf{t}$
7. $\mathcal{M}(\langle \mathbf{d}_1\rangle)(\mathbf{neg}\, f_1) = \mathbf{f}$
8. $\mathcal{M}(\langle \mathbf{d}_0\mathbf{d}_1\rangle)(\alpha) = \mathbf{f}$
9. $\mathcal{M}(\langle \mathbf{d}_2\rangle)(c_1) = \mathbf{u}$
10. $\mathcal{M}(\langle \mathbf{d}_2\rangle)(c_2) = \mathbf{u}$
11. $\mathcal{M}(\langle \mathbf{d}_2\rangle)(f_1) = \mathbf{u}$
12. $\mathcal{M}(\langle \mathbf{d}_2\rangle)(\mathit{inconsistent}) = \mathbf{u}$
13. $\mathcal{M}(\langle \mathbf{d}_1\mathbf{d}_2\mathbf{d}_3\rangle)(\gamma) = \mathbf{u}$

Items 1,2,3, 4, and 5 hold due to the premises in $\mathcal{S}$. Item 6 holds because of the effect of α. Items 7 and 8 are false because they can be safely assumed to be false in the minimal model. Items 9 and 10 are undefined because the postcondition of α is undefined. Item 11 is undefined because c_1 and c_2 are undefined. Item 12 is undefined because f_1 is undefined. And item 13 is undefined because part of its definition is undefined. Note that it is possible for f_1 to be undefined and $\mathbf{neg}\, f_1$ to be true in a path. □

3.7.16. EXAMPLE. Consider the specification in Example 3.7.5 together with the following rule defining the fluent *inconsistent*.

$$\mathit{inconsistent} \leftarrow \mathit{healthy}(P),\ \mathbf{neg}\, \mathit{healthy}(P)$$

In the specification $\frac{\Lambda}{\mathbf{I}_\emptyset}$, the sets $\mathbf{P}$, remain the same since they all are **not**-free. In $\mathcal{E}$, only the frame axioms and the definition of *do_presc* change as follows

$$\begin{array}{l}\mathit{dna\_t}(P) \otimes \mathit{do\_presc}(T,P,D) \otimes \mathbf{u}^\pi \rightarrow \mathit{do\_presc}(T,P,D) \otimes \mathit{dna\_t}(P)\\ \mathit{consent}(A,B) \otimes \mathit{do\_presc}(T,P,D) \otimes \mathbf{u}^\pi \rightarrow \mathit{do\_presc}(T,P,D) \otimes \mathit{consent}(A,B)\\ \mathit{dr}(D) \otimes \mathit{do\_presc}(T,P,D) \otimes \mathbf{u}^\pi \rightarrow \mathit{do\_presc}(T,P,D) \otimes \mathit{presc}(D,P,T)\end{array}$$

for every π.

Since $\frac{\Lambda}{\mathbf{I}_\emptyset}$ is **not**-free, it has a minimal model $\Gamma^{\uparrow 1}(\mathbf{I}_\emptyset) = \mathcal{M}_1$. It follows from the construction of $\mathcal{M}_1$ that $\mathcal{M}_1(\langle \mathbf{d}_1 \rangle)(\mathit{inconsistent}) = \mathbf{u}$. It is not hard to see that in the WFM of Λ, *inconsistent* is also undefined in $\mathcal{M}_1(\langle \mathbf{d}_1 \rangle)$. This is because the frame axioms are preventing the inconsistency from occurring, but it is still detected. □

3.7.17. THEOREM. *$WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is the least model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$.*

11. PROOF. *See Appendix E.* □

3.8 Lifting The Interloping Restriction

In this section we remove the restriction over interloping PADs actions imposed in Section 3.4. As a result, we obtain a much more expressive language. In addition, we show how to use default negation to prevent the frame axioms from introducing inconsistencies.

Recall that two PADs are said to be interloping if they share a common primitive effect. The action theory of a $\mathcal{TR}^{PAD}$ transaction base $\mathbf{P}$, as defined in Section 3.4, was composed by the following set of axioms:

- Forward Inertia
- Forward and Backward Disablement
- Backward Inertia
- Causality
- Backward Projection

Using default negation, we can remove the restriction over interloping actions and construct a new action theory that can detect inconsistencies. In order to do this, we need to introduce a new fluent *inconsistent* defined in a similar way as in Example 3.7.16. That is, for every fluent h, the transaction base contains:

$$\mathit{inconsistent} \leftarrow h,\ \mathbf{neg}\, h$$

The *new* action theory, $\mathcal{A}(\mathcal{E})$, contains the following set of axioms $Frame(\mathcal{E})$:

Forward Inertia For each fluent literal h and each partially defined action α such that neither h nor $\mathbf{neg}\, h$ is a primitive effect of α, the following axiom is in $Frame(\mathcal{E})$:

$$(inertial(h) \wedge h) \otimes \alpha \otimes \mathbf{not}\ \mathit{inconsistent} \rightarrow \alpha \otimes h \tag{3.14}$$

Forward and Backward Disablement Let g be a literal and α a *pda*. Let $\mathbf{p}_g^1 \ldots \mathbf{p}_g^n$ be the partial action definitions of α with the primitive effect g. Let $f_g^1 \ldots f_g^n$ be the preconditions of $\mathbf{p}_g^1 \ldots \mathbf{p}_g^n$ respectively. Then the following *forward disablement* axioms are in $Frame(\mathcal{E})$:

$$\begin{aligned}(inertial(g) \wedge \mathbf{not}\, f_g^1 \wedge \cdots \wedge \mathbf{not}\, f_g^n) \otimes g \otimes \alpha \otimes \\ \mathbf{not}\ \mathit{inconsistent} \rightarrow \alpha \otimes g \\ (inertial(g) \wedge \mathbf{not}\, f_g^1 \wedge \cdots \wedge \mathbf{not}\, f_g^n) \otimes \mathbf{neg}\, g \otimes \alpha \otimes \\ \mathbf{not}\ \mathit{inconsistent} \rightarrow \alpha \otimes \mathbf{neg}\, g\end{aligned} \tag{3.15}$$

If g is a base fluent,[7] then the following *backward disablement* axioms are also in $Frame(\mathcal{E})$:

$$\begin{aligned}(inertial(g) \wedge \mathbf{not}\, f_g^1 \wedge \cdots \wedge \mathbf{not}\, f_g^n \wedge \mathbf{not}\ \mathit{inconsistent}) \otimes \alpha \otimes \\ g \rightarrow g \otimes \alpha \\ (inertial(g) \wedge \mathbf{not}\, f_g^1 \wedge \cdots \wedge \mathbf{not}\, f_g^n \wedge \mathbf{not}\ \mathit{inconsistent}) \otimes \alpha \otimes \\ \mathbf{neg}\, g \rightarrow \mathbf{neg}\, g \otimes \alpha\end{aligned} \tag{3.16}$$

In other words, if the *pda*s $\mathbf{p}_g^1 \ldots \mathbf{p}_g^n$ are disabled in some state then executing α in that state does not change the truth value of the fluents g.

Backward Inertia For each *pda* α such that f is a base fluent which is not primitive effect of α:

$$(inertial(f) \wedge \mathbf{not}\ \mathit{inconsistent}) \otimes \alpha \otimes f \rightarrow f \otimes \alpha \tag{3.17}$$

The Backward Projection and Causality axioms are removed from the action theory, since once the interloping restriction is lifted, there is not one-to-one relation between preconditions and effects.

Now we establish the relation between the action theory in $\mathcal{TR}^{PAD}$ with respect to the action theory in $\mathcal{TR}^{PAD}$ with default negation. First we need the following definition

3.8.1. Definition. [Simple Specification] We say that a specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is simple if

- It makes no use of default negation.
- It has no interloping actions.
- There is no literal f such that there exists a path π where $\mathcal{E}, \mathbf{P}, \mathcal{S}, \pi \models f$ and $\mathcal{E}, \mathbf{P}, \mathcal{S}, \pi \models \mathbf{neg}\, f$.

[7] Recall that base fluents are the fluents which are not defined by Horn-rules.

- For every database state identifier $\mathbf{d}$ and literal f, $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models f$ or $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models \mathbf{neg}\, f$. □

We say that a partial Herbrand path structure $\mathbf{I}$ is 2 valued, if and only if for every path π and **not**-free literal h, either $\mathbf{I}(\pi)(h) = \mathbf{t}$, or $\mathbf{I}(\pi)(h) = \mathbf{f}$.

3.8.2. Definition. We define the function $2val$ from simple partial Herbrand path structures, to Herbrand path structures as follows:

$$2val(\mathbf{I})(\pi) = \{f \mid \mathbf{I}(\pi)(f) = \mathbf{t}\}$$

□

Since in $\mathcal{TR}^{PAD}$ with default negation we allow interloping action, in the general case we cannot have neither Causality frame axioms nor Backward Projection. However, since now we want to compare these two different action theories for simple specifications, we add the Causality frame axioms and Backward Projection to $\mathcal{A}(\mathcal{E})$ as defined in Section 3.4.

3.8.3. Lemma. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a simple specification. Let $\mathbf{I}$ be a model of $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$. For every* **not***-free serial conjunction G,*

$$\textit{if } \mathbf{I}, \pi \models G \textit{ then } 2val(\mathbf{I}), \pi \models G$$

12. Proof. *Follows straightforwardly from Definition 3.8.2.* □

3.8.4. Theorem. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a simple specification. Let $\mathbf{I}$ be a model of $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$. Then $2val(\mathbf{I})$ is a model of $(\mathcal{A}(\mathbf{P} \cup \mathcal{E}), \mathcal{S})$.*

13. Proof. *See Appendix E.* □

3.9 Summary of the Contributions

In this chapter we extended Transaction Logic and made it suitable for reasoning about partially defined actions. We illustrated the power of the language for complex reasoning tasks involving actions and gave a sound and complete proof theory for that formalism. We also showed that, when all partially defined actions are definite, such reasoning can be done by a reduction to ordinary logic programming. This last contribution provides an easy way to implement and experiment with the formalism, although a better implementation should be using the proof theory directly, similarly to the implementation of the serial-Horn subset of $\mathcal{TR}$ in FLORA-2 [50].

This work continues the line of research started in [10], which, however, was targeting a different fragment of $\mathcal{TR}$, and it did not provide a complete proof

theory or a reduction to logic programming. It also did not consider premise statements and thus could not be used for reasoning about partially defined actions without further extensions.

In many respects, $\mathcal{TR}^{PAD}$ supports more general ways of describing actions than other related formalisms [39, 35, 8, 17, 38, 7, 16], including non-determinism, recursion, and hypothetical suppositions. Uniquely among these formalisms it supports powerful ways of action composition. Nevertheless, $\mathcal{TR}^{PAD}$ does not subsume other works on the subject, as it cannot perform certain reasoning tasks that are possible with formalisms such as [16, 7, 38].

Finally, we extended $\mathcal{TR}^{PAD}$ with default negation to allow non-monotonic reasoning. In particularly, we showed how default negation can be used to lift the restriction over interloping actions and be able to detect inconsistencies. Along the way we defined a well-founded semantics for $\mathcal{TR}^{PAD}$, which, to the best of our knowledge, has never been done before.

Chapter 4

Modeling Action Languages with $\mathcal{TR}^{PAD}$

> There is neither happiness nor misery in the world; there is only the comparison of one state with another, nothing more.
>
> Alexandre Dumas
> The Count of Monte Cristo

A number of sophisticated logical theories to reason about actions have been developed over the years, including Situation Calculus [61], Fluent Calculus and Flux [76, 62], Event Calculus [51], $\mathcal{L}_1$[8], $\mathcal{A}$[35], $\mathcal{L}_1$[8], $\mathcal{C}$[36], $\mathcal{ALM}$[46]. Unfortunately, most of such languages have their weak points along with the strong ones, and neither is sufficient as a logical foundation for agents.

$\mathcal{TR}^{PAD}$ is based on a very different logical paradigm than the aforesaid languages, and it is an interesting challenge to understand the relative expressive power of these languages.

In this chapter we identify and compare the modeling and reasoning capabilities of $\mathcal{TR}^{PAD}$ (without default negation) and $\mathcal{L}_1$. We chose $\mathcal{L}_1$ because it is a powerful language that can serve as a good representative of the family of action languages mentioned earlier. However, we also briefly discuss the relation between $\mathcal{TR}^{PAD}$ and the action languages: Situation Calculus, Event Calculus, Fluent Calculus, $\mathcal{C}$ and $\mathcal{ALM}$.

After introducing the languages, we compare them on a number of examples and then prove the equivalence between subsets of both languages. However, it is the symmetric difference of these languages that is perhaps most interesting. Throughout this chapter we investigate that difference using the following

running example based on the health insurance scenario:

4.0.1. Example. [Health Insurance] The problem is to encode the following set of health insurance regulations. (i) For vaccination to be *compliant*, doctors must require that patients obtain authorization from their health insurers *prior* to vaccination. (ii) To obtain authorization, the patient must first visit a doctor (or be a doctor). (iii) Vaccinating a *healthy* patient makes her *immune* and *healthy*. (iv) A patient who has a *flu* is not healthy, but (v) flu can be treated with *antivirals*. In addition, we know that (vi) there is a patient, John, who has a *flu*, is not *immune* and (vii) is a *doctor*. We want to find a legal sequence of actions (a plan) to make John immune and healthy. □

We show how limitations of each language can be worked around to represent the above problem. Then we venture into the domain of action planning and discuss how it is done in each language. We show that certain planning goals, those that require intermediate conditions in order to construct legal plans, cannot be easily expressed in $\mathcal{L}_1$ and that they are very natural in $\mathcal{TR}^{PAD}$.

This chapter is organized as follows. Section 4.1 presents the necessary background needed to understand $\mathcal{L}_1$. Section 4.2 illustrates similarities and differences between the two formalisms by means of non-trivial examples. Section 4.3 studies a fragment of $\mathcal{L}_1$ and reduces it to $\mathcal{TR}^{PAD}$. We show that the reduction is sound with respect to the $\mathcal{L}_1$ semantics and complete with respect to the logic programming embedding of $\mathcal{L}_1$. Section 4.4 discusses planning problems in both formalisms and Section 4.5 compares $\mathcal{TR}^{PAD}$ with other popular action languages: Situation Calculus, Event Calculus, Fluent Calculus, $\mathcal{C}$ and $\mathcal{ALM}$. Section 4.6 briefly discuss the relation between $\mathcal{TR}^{PAD}$ with default negation and the action languages mentioned above. Section 4.7 concludes the chapter.

4.1 Action Language $\mathcal{L}_1$

This section reviews the basics of the action language $\mathcal{L}_1$ [8]. The alphabet of $\mathcal{L}_1$ consists of three disjoint nonempty sets of symbols: a set of *fluent names* $\mathbf{F}$, a set of *action names* $\mathbf{A}$, and a set of *situations* $\mathbf{S}$. The set $\mathbf{S}$ contains two special situations: s_0, which is called the **initial situation**, and s_C, called the **current situation** (which is also the last one). The language $\mathcal{L}_1$ contains two kinds of propositions: *causal laws* and *facts*. In the following table, each $f, f_1 \dots f_n$ is a fluent literal, each s_i is a situation, a is an action and α is a sequence of actions.

Causal laws		
(1)	a `causes` f `if` $f_1 \dots f_n$	(causal law)

Atomic Facts		
(2)	α `occurs_at` s	(occurrence fact)
(3)	f `at` s	(fluent fact)
(4)	s_1 `precedes` s_2	(precedence fact)

The causal law *(1)* describes the effect of a on f. We will say that $f_1 \dots f_n$ is the **precondition** of the action a and f is its **effect**. Intuitively, the occurrence fact *(2)* means that the sequence α of actions occurred in situation s. The fluent fact *(3)* means that the fluent f is true in the situation s. The precedence fact *(4)* states that the situation s_2 occurred after the situation s_1. Statements of the form *(2)*, *(3)*, *(4)*, are called ***atomic facts***. A *fact* is a conjunction or disjunction of atomic facts. An $\mathcal{L}_1$ **domain description** is a set of laws and facts $\mathcal{D}$. Two causal laws of the form

$$a \texttt{ causes } f \texttt{ if } f_1 \dots f_n$$
$$a \texttt{ causes } \neg f \texttt{ if } q_1 \dots q_m$$

are **contradictory** if $\{f_1 \dots f_n\} \cap \{\neg q_1 \dots \neg q_m\} = \emptyset$.

It is worth noting that with disjunction one can express *possible states* of the world and non-determinism. For instance, we could say that in the initial state the patient is either healthy or has a flu: *healthy* `at` $s_0 \vee$ *flu* `at` s_0, this is not expressible in $\mathcal{TR}^{PAD}$. One can also state that in the initial state, either the patient is vaccinated or the doctor asks for an authorization *vaccinate* `occurs_at` $s_0 \vee$ *request_authorization* `occurs_at` s_0. With *conjunctions* of occurrence-facts we could express concurrency of action execution as in

$$\textit{vaccinate} \texttt{ occurs\_at } s_0 \wedge \textit{request\_authorization} \texttt{ occurs\_at } s_0$$

However, the semantics (cf. Definition 4.1.3) does not support concurrent execution of actions at the same state. Unfortunately, the boost of expressivity coming from the propositional combination of atomic facts cannot be exploited for reasoning. This is because the reduction of $\mathcal{L}_1$ to LP works only when such combinations are disallowed.

4.1.1. Example. [8] Suppose that we know two facts about Fred:

- Initially, Fred was alive and dry.
- When the water pistol was squirted, at a later moment, a shot was fired at Fred.

Suppose also that it is generally known that

- *squirting* makes Fred *wet* and
- *shooting* makes Fred *dead*.

The information informally introduced above can be represented by a domain description D_1 consisting of the following propositions:

$$D_1 = \begin{cases} Facts: \\ (p_1) \quad \texttt{alive at } \texttt{s}_0 \\ (p_2) \quad \texttt{dry at } \texttt{s}_0 \\ (p_3) \quad \texttt{squirt occurs\_at } \texttt{s}_0 \\ (p_4) \quad \texttt{s}_0 \texttt{ precedes } \texttt{s}_1 \\ (p_5) \quad \texttt{shoot occurs\_at } \texttt{s}_1 \\ Laws: \\ (p_6) \quad \texttt{squirt causes } \neg\texttt{dry} \\ (p_7) \quad \texttt{shoot causes } \neg\texttt{alive} \end{cases}$$

□

Intuitively, $\mathcal{L}_1$ works with the following informal assumptions

1. The values of fluents can change only by the effects of actions;
2. The only action names in **A** are those from the language of the domain description;
3. There are no effects of actions except those specified by the causal laws of the domain;
4. Only one action can be executed at the time; and
5. Actions are executed only if it is specified so in the domain description.

Semantics:

A model of a domain description $\mathcal{D}$ consists of a mapping from situations to sequences of actions and a mapping from sequences of action to states. A **state** is a set of fluent-atoms.

4.1.2. Definition. [Situation Assignment] A **situation assignment** of $\mathcal{D}$, $sit2act$, is a partial function from situations to sequences of actions such that:

- $sit2act(s_0) = [\,]$, where $[\,]$ is the empty sequence of actions.
- For every $s \in \mathbf{S}$, $sit2act(s)$ is a prefix of $sit2act(s_{\mathbf{C}})$ □

Intuitively, if $sit2act(s_k) = \alpha$, it means that executing α in s_0 leads to s_k.

4.1.3. DEFINITION. [Action Interpretation] An **action interpretation** of $\mathcal{D}$, $act2st$, is a partial function from sequences of actions to states such that:

- The empty sequence [ ] is in the domain of $act2st$ and
- For any sequence of actions α and action a, if $[\alpha, a]$ is in the domain of $act2st$, then so is α. □

By composing these two functions we can map situations to states. Given a fluent name f, and a state σ, we say that f **holds** in σ if $f \in \sigma$; $\neg f$ holds in σ if $f \notin \sigma$. The truth of a propositional combination of fluents with respect to σ is defined as usual. We say that a fluent literal f is an **immediate effect** of an action a_i in a state σ, if there is a causal law of the form a_i `causes` f `if` $f_1 \dots f_n$ in $\mathcal{D}$, whose preconditions $f_1 \dots f_n$ hold in σ. The following three sets of fluents are needed to define models in $\mathcal{L}_1$.

$$\begin{aligned} E^+_{a_i}(\sigma) &= \{f \mid f \in \mathbf{F} \text{ and } f \text{ is an immediate effect of } a_i \text{ in } \sigma\} \\ E^-_{a_i}(\sigma) &= \{f \mid f \in \mathbf{F} \text{ and } \neg f \text{ is an immediate effect of } a_i \text{ in } \sigma\} \\ Res(a_i, \sigma) &= (\sigma \cup E^+_a(\sigma)) \setminus E^-_a(\sigma) \end{aligned}$$

An action interpretation $act2st$ *satisfies* the causal laws of $\mathcal{D}$ if for *any* sequence of actions $[\alpha, a]$ from the language of $\mathcal{D}$,

$$act2st([\alpha, a]) = \begin{cases} Res(a, act2st(\alpha)) & \text{if } E^+_a(act2st(\alpha)) \cap E^-_a(act2st(\alpha)) = \emptyset \\ \text{undefined} & \text{otherwise} \end{cases}$$

4.1.4. DEFINITION. [Model] A **model** M of $\mathcal{L}_1$, is a pair $(act2st, sit2act)$, where

- $act2st$ is an action interpretation that satisfies the causal laws in $\mathcal{D}$,
- $sit2act$ is a situation assignment of $\mathbf{S}$ where $sit2act(s_\mathbf{C})$ belongs to the domain of $act2st$. □

The **actual path** of a model M for a domain description $\mathcal{D}$ is $sit2act(s_\mathbf{C})$.[1] Intuitively, it represents the unique sequence of actions defined by M and consistent with $\mathcal{D}$.

The ***query language*** associated with $\mathcal{L}_1$, denoted $\mathcal{L}^Q_1$, consists of all fluent-facts in $\mathcal{L}_1$, plus an expression of the form f `after` $[a_1 \dots a_n]$ `at` s, called a ***hypothesis***. Intuitively, it says that if the sequence $a_1 \dots a_n$ of actions *can* be executed in the situation s, then the fluent literal f must be true afterwards. Observe that by defining a pair of relationships between situations and states ($sit2act$ and $act2st$) rather than one enables $\mathcal{L}_1$ to express hypothetical queries since $act2st$ can query states which are not associated with any situation in the domain description.

[1] Recall that $s_\mathbf{C}$ is the current situation.

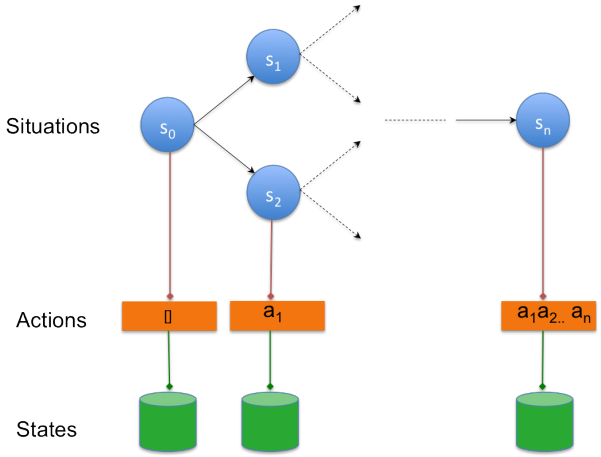

Figure 4.1: $\mathcal{L}_1$ models

4.1.5. Definition. [Satisfaction] For any model $M = (act2st, sit2act)$

1. f **at** s – is true in M if f is true in $act2st(sit2act(s))$.
2. α **occurs_at** s – is true in M. if the sequence $[sit2act(s), \alpha]$ is a prefix of the actual path $sit2act(s_{\mathbf{C}})$ of M.
3. s_1 **precedes** s_2 – is true in M if $sit2act(s_1)$ is a proper prefix of $sit2act(s_2)$.
4. f `after` $[a_1 \ldots a_n]$ `at` s is true in M if f is true in $act2st([sit2act(s), a_1 \ldots a_n])$.
5. Truth of conjunctions and disjunctions of atomic facts in M are defined as usual. □

Since fluent facts can be expressed as hypotheses, we focus on just these kind of statements.

The next definition imposes a minimality condition on the situation assignments of **S** that formalizes the informal assumption that an action is executed only if it is required by the domain description.

4.1.6. Definition. [Minimal Model] A model $M = (act2st, sit2act)$ is a ***minimal model*** of a domain description D in $\mathcal{L}_1$ if the following conditions are satisfied:

1. $act2st$ is a causal model of D
2. facts of D are satisfied in M and

3. there is no other interpretation $M' = (act2st, sit2act')$ such that M' satisfies conditions 1 and 2 above and $sit2act'(s_{\mathbf{C}})$ is a subsequence of $sit2act(s_{\mathbf{C}})$ □

Definition 4.1.7 describes the set of acceptable conclusions obtainable from a domain description $\mathcal{D}$.

4.1.7. DEFINITION. [Entailment] A domain description $\mathcal{D}$ **entails** a query q (written as $\mathcal{D} \models q$) iff q is true in all minimal models of $\mathcal{D}$. We will say that the **answer** given by $\mathcal{D}$ to a query q is *yes* if $\mathcal{D} \models q$, *no* if $\mathcal{D} \models \neg q$, *unknown* otherwise. □

Reducing $\mathcal{L}_1$ to Logic Programming

In this section we present a reduction from a fragment of $\mathcal{L}_1$, to Logic Programming. This reduction was developed in [8], and works only with *simple* domain descriptions. This notion is defined next. First we introduce the definition of *explicit paths*.

4.1.8. DEFINITION. [Explicit Path] Let D be a consistent domain description and $s_0 \ldots s_k$ be a list of situation constants occurring in statements from D. We say that D has an **explicit path** if

1. $\mathtt{s_i}$ `precedes` $\mathtt{s_{i+1}} \in D(0 \leq i < k)$
2. $D \models s_k = s_{\mathbf{C}}$
3. there is a sequence $\alpha = a_0 \ldots a_{k-1}$ of actions such that:
   - $\mathtt{a_i}$ `occurs_at` $\mathtt{s_i} \in \mathtt{D}(\mathtt{0} \leq \mathtt{i} < \mathtt{k})$
   - $D \models \alpha$ `occurs_at` $\mathtt{s_0}$ □

In other words, a domain description has an explicit path if there is complete information about the occurrences of actions and temporal relation between situations. It is easy to see that any domain description D satisfying these conditions has a unique actual path.

4.1.9. DEFINITION. [Simple Domain Description] A domain description D is **simple** if

1. D is consistent;
2. D has an explicit path;
3. All facts of D are atomic, and

4. D does not contain contradictory causal laws. □

We denote the set of LP rules that constitute the reduction, $\mathbf{P}_D$. The alphabet of $\mathbf{P}_D$ will consist of symbols for actions, fluent literals (where $\neg$ is replaced by the strong negation **neg**), and situations from the language of D, predicates $true\_at, true\_after, all\_true\_after$, $false\_after$, $one\_false\_after, causes$, and ab. We will use the following typed variables:

- $A, A_1, A_2 \ldots$ for actions
- $F, F_1, F_2 \ldots$ for fluent literals
- P for lists of fluent literals
- R for lists of actions

The corresponding lower case letter of A, F, S, etc., will denote constants of respective types. We say that the atoms of the form $true\_after(f, r, s_i)$ and $false\_after(f, r, s_i)$ are **incompatible**.

The program $\mathbf{P}_D$ in the above language is **consistent** if it has an answer set, and no answer set of $\mathbf{P}_D$ contains incompatible atoms. Let D be a simple domain description with the explicit path $a_0 \ldots a_{k-1}$. The reduction $\mathbf{P}_D$ of a domain description D consists of the following rules:

1. Domain Dependent Axioms

   (a) (AP) Description of Explicit Path

   $$\begin{array}{l} imm\_follows(s_1, s_0) \\ \ldots \\ imm\_follows(s_k, s_{k-1}) \\ occurs\_at(a_0, s_0) \\ \ldots \\ occurs\_at(a_{k-1}, s_{k-1}) \end{array}$$

   (b) (BC) Boundary Conditions

   $$true\_at(f, s_i) \in \mathbf{P}_D \text{ for each } \texttt{f at s}_\texttt{i} \in D$$

   (c) (CL) Causal Laws

   $$causes(a_i, f, p) \in \mathbf{P}_D \text{ for each } \texttt{a}_\texttt{i}\ \texttt{causes f if p} \in D$$

2. Domain Independent Axioms

(a) (EA) Effects of actions

$$
\begin{array}{lll}
e_1. & true\_after(F, [], S) & :- true\_at(F, S) \\
e_2. & true\_after(F, [A \mid R], S) & :- causes(A, F, P), \\
 & & \quad all\_true\_after(P, R, S) \\
e_3. & false\_after(F, R, S) & :- true\_after(\mathbf{neg}\, F, R, S) \\
e_4. & all\_true\_after([], R, S) & . \\
e_5. & all\_true\_after([F \mid P], R, S) & :- true\_after(F, R, S), \\
 & & all\_true\_after(P, R, S) \\
e_6. & one\_false\_after(P, R, S) & :- false\_after(F, R, S)(\text{ with } F \in P)
\end{array}
$$

(b) (FI) Inertia Axioms

$$
\begin{array}{lll}
i_1. & true\_after(F, [A \mid R], S) & :- true\_after(F, R, S), \\
 & & \mathbf{not}\, ab(F, A, R, S) \\
i_2. & ab(F, A, R, S) & :- causes(A, F, P), \\
 & & \mathbf{not}\, one\_false\_after(P, R, S)
\end{array}
$$

(c) (SI) Second Inertia Axiom

$$
\begin{array}{ll}
true\_at(F, S_2) & :- imm\_follows(S_1, S_2), \\
 & occurs\_at(A_1, S_1), true\_after(F, [A_1], S_1)
\end{array}
$$

The set of all ground instantiation of all rules of $\mathbf{P}_D$ except for the Second Inertia Axiom (SI) and the Description of the Explicit Path (AP), not containing any other situation constants except s_i is denoted by $\mathbf{H_i}$.

It is easy to see that the LP reduction $\mathbf{P}_D$ consists of the union of the sets $H_0 \ldots H_k$ together with the ground instantiations of SI and AP.

4.1.10. Definition. [$\mathbf{P}_1$] Let D be a simple domain description. Let $\mathbf{P}_D$ be the LP-reduction of D. We define $\mathbf{P}_1$ to be the program obtained from $\mathbf{P}_D$ by replacing (SI) and (AP) by

$$true\_at(F, s_i) : -true\_after(F, a_{i-1}, s_{i-1})$$

where $0 < i \leq k$ and $(a_{i-1}$ **occurs_at** $s_{i-1}) \in D$. □

Since $D \models$ `f at s` if and only if $D \models f$ `after [ ] at` s, from now on we will limit our query language to formulas of the form `f after` α `at s`. Given a query q of the form f `after` α `at` s, $\pi(q)$ will denote the LP query $true\_after(f, \alpha, s)$.

4.1.11. Proposition. *[8] Consider a simple domain description D. Then for any query q in D, $\mathbf{P}_1 \models \pi(q)$ iff $\mathbf{P}_D \models \pi(q)$*

4.1.12. PROPOSITION. *[8] Let D be a simple domain description. For any $0 \leq i \leq k$, and any collection I of formulas of the form $true\_at(f, s_i)$ such that $D \models$ `f` **at** `s_i`, the program $H_i \cup I$ has a unique answer set and is sound with respect to D.*

4.1.13. PROPOSITION. *[8] Let l_i $(0 \leq i \leq k)$ be the rule*

$$true\_at(f, s_i) : -true\_after(f, [a_{i-1}], s_{i-1})$$

For any $0 \leq m \leq k$, the program T_m:

$$T_m = H_0 \cup (l_1 \cup H_1) \cup \cdots \cup (l_m \cup H_m)$$

has a unique answer set and is sound with respect to D

Note that T_k is the same as $\mathbf{P}_D$.

4.1.14. PROPOSITION (SOUNDNESS OF $\mathbf{P}_D$). *[8] For any simple domain description D its logic programming reduction $\mathbf{P}_D$ is sound with respect to D.*

4.2 Motivating Examples

In this section we show a set of non-trivial examples to highlight the commonalities and differences between $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$. For simplicity, the examples in this chapter are all propositional. However, $\mathcal{TR}^{PAD}$ supports first order predicates and variables.

4.2.1. EXAMPLE. [Health Insurance (cont'd)] Consider the US health insurance regulations scenario of Example 4.0.1. The specification $\mathcal{T}_H = (\mathbf{P}, \mathcal{S})$ in Figure 4.2 shows a $\mathcal{TR}^{PAD}$ representation of that scenario. In the rules in Figure 4.2, *vaccinate_legally*, *request_authorization*, *vaccinate* and *take_antivir* are actions, while *healthy*, *flu*, *visited_dr*, *doctor*, *immune* and *authorized* are fluents. Note that the two PADs defining *request_authorization* are interloping, which was discussed before. For clarity, each statement is numbered with the corresponding regulation number from Example 4.0.1.

The corresponding domain description $\mathcal{D}_H$ for the language $\mathcal{L}_1$ is shown in Figure 4.3. Since fluent rules are not allowed in $\mathcal{L}_1$, we manually encoded the consequence of regulation *(iv)* in the initial state. □

4.2.2. EXAMPLE. [Health Insurance (cont'd)] Consider Example 4.2.1 extended with the following additional information.

(viii) executing *take_antivir* in the initial state leads to state 1.

$$\mathbf{P} = \begin{cases} (i) & \textit{vaccinate\_legally} \leftarrow \textit{request\_authorization} \otimes \textit{authorized} \otimes \textit{vaccinate} \\ (ii) & \textit{doctor} \otimes \textit{request\_authorization} \rightarrow \textit{request\_authorization} \otimes \textit{authorized} \\ (ii) & \textit{visited\_dr} \otimes \textit{request\_authorization} \rightarrow \textit{request\_authorization} \otimes \textit{authorized} \\ (iii) & \textit{healthy} \otimes \textit{vaccinate} \rightarrow \textit{vaccinate} \otimes \textit{immune} \otimes \textit{healthy} \\ (iv) & \mathbf{neg}\ \textit{healthy} \leftarrow \textit{flu} \\ (v) & \textit{take\_antivir} \rightarrow \textit{take\_antivir} \otimes \textit{healthy} \otimes \mathbf{neg}\ \textit{flu} \end{cases}$$

$$\mathcal{S} = \begin{cases} (vi) & \mathbf{d}_1 \rhd \textit{flu} \\ (vi) & \mathbf{d}_1 \rhd \mathbf{neg}\ \textit{immune} \\ (vii) & \mathbf{d}_1 \rhd \textit{doctor} \end{cases}$$

Figure 4.2: $\mathcal{TR}^{PAD}$ formalization of the health care scenario

$$\mathcal{D}_H = \begin{cases} (i) & \textit{vaccinate\_legally}\ \texttt{causes}\ \textit{immune} \wedge \textit{healthy}\ \texttt{if}\ \textit{healthy} \wedge \textit{authorized} \\ (ii) & \textit{request\_authorization}\ \texttt{causes}\ \textit{authorized}\ \texttt{if}\ \textit{doctor} \\ (ii) & \textit{request\_authorization}\ \texttt{causes}\ \textit{authorized}\ \texttt{if}\ \textit{visited\_dr} \\ (iii) & \textit{vaccinate}\ \texttt{causes}\ \textit{immune}\ \texttt{if}\ \textit{healthy} \\ (iv) & \neg\textit{healthy}\ \texttt{at}\ s_0 \\ (v) & \textit{take\_antivir}\ \texttt{causes}\ \textit{healthy}\ \texttt{if}\ \textit{flu} \\ (vi) & \textit{flu}\ \texttt{at}\ s_0 \\ (vii) & \textit{doctor}\ \texttt{at}\ s_0 \end{cases}$$

Figure 4.3: $\mathcal{L}_1$ formalization of the health care scenario

(ix) executing *vaccinate* in state 1 leads to state 2.

This additional information is shown in Figure 4.4 for both languages. In $\mathcal{TR}^{PAD}$, we can use the inference system to derive $\mathbf{P}, \mathcal{S}, \mathbf{d}_0\texttt{---} \models \textit{take\_antivir} \otimes \textit{vaccinate} \otimes \textit{healthy}$, meaning that the patient becomes healthy as a result. In $\mathcal{L}_1$, the reduction of $\mathcal{D}_H$ to logic programming, $\mathbf{P}_{\mathcal{D}_H}$, can be used to establish the same thing (albeit in different notation): $\mathcal{D}_H \models \textit{healthy}\ \texttt{after}\ [\textit{take\_antivir}, \textit{vaccinate}]\ \texttt{at}\ s_0$. □

Discussion: The formalizations of Example 4.0.1 in $\mathcal{L}_1$ and $\mathcal{TR}^{PAD}$ are not equivalent, however. For example, $\mathcal{L}_1$ does not allow compound actions, making the representation of the problem a little harder and less modular; and it does not allow fluent rules which makes it impossible to express dependencies between fluents. On the other hand, $\mathcal{TR}^{PAD}$ (recall that we are working without default negation) does not support interloping actions in the definition of *request_authorization*. This constraint can be circumvented using the transformation in Figure 4.5, but this comes at the expense of readability.

$$\mathcal{T}_H = \begin{cases} (viii) & - & \mathbf{d}_0 \overset{take\_antivir}{\rightsquigarrow} \mathbf{d}_1 \\ (ix) & - & \mathbf{d}_1 \overset{vaccinate}{\rightsquigarrow} \mathbf{d}_2 \end{cases}$$

$$\mathcal{D}_H = \begin{cases} (viii) & - & take\_antivir \texttt{ occurs\_at } s_0 \\ (viii) & - & s_0 \texttt{ precedes } s_1 \\ (ix) & - & vaccinate \texttt{ occurs\_at } s_0 \\ (ix) & - & s_1 \texttt{ precedes } s_2 \end{cases}$$

Figure 4.4: Describing states and executions in $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$.

$$\mathcal{D}_H = \begin{cases} (i) & - & vaccinate\_legally \leftarrow request\_authorization_1 \otimes authorized \otimes vaccinate \\ (i) & - & vaccinate\_legally \leftarrow request\_authorization_2 \otimes authorized \otimes vaccinate \\ (ii) & - & doctor \otimes request\_authorization_1 \rightarrow request\_authorization_1 \otimes authorized \\ (ii) & - & visited\_dr \otimes request\_authorization_2 \rightarrow request\_authorization_2 \otimes authorized \end{cases}$$

Figure 4.5: Replacing Interloping actions in $\mathcal{TR}^{PAD}$

In sum, the above examples show certain similarities in the modeling capabilities of $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$: elementary actions (PADs vs. causal laws), states (*state*-premises vs. fluent facts), execution of actions (*state*-premises vs. occurrence facts), hypothetical queries (hypothetical transactions vs. hypotheses). However, the semantics of these languages are completely different and so are some of the capabilities (compound actions and fluent rules vs. interloping actions). From the reasoning perspective, $\mathcal{TR}^{PAD}$ has a sound and complete proof system, whereas $\mathcal{L}_1$'s reasoning depends on a sound, but *incomplete* translation to logic programing.

4.3 Representing $\mathcal{L}_1^{\mathbf{A}}$ in $\mathcal{TR}^{PAD}$

In this section we define a reduction for the *accurate* fragment of $\mathcal{L}_1$, denoted $\mathcal{L}_1^A$ to $\mathcal{TR}_D^{PAD}$ (without default negation), and provide a soundness proof. The *accurate* fragment of $\mathcal{L}_1$ is defined as $\mathcal{L}_1^{\mathrm{A}}$ except that it allows only *simple* domain descriptions and disallows interloping causal laws. Although the reduction presented here is not complete with respect to $\mathcal{L}_1^A$, we show that it is complete with respect to the LP reduction of $\mathcal{L}_1$ developed in [8]. Let $\mathcal{D}$ be a simple $\mathcal{L}_1^A$ domain description. Given an alphabet $\mathcal{L}_\mathcal{D}$ of $\mathcal{D}$, the corresponding language $\mathcal{L}_{\mathcal{TR}}$ of the target $\mathcal{TR}^{PAD}$ formulation will consist of symbols for actions and fluents literals from $\mathcal{L}_\mathcal{D}$, except that the symbol $\neg$ in $\mathcal{L}_1^{\mathrm{A}}$, is replaced with **neg** in $\mathcal{L}_{\mathcal{TR}}$. In the remainder of the section, let $\mathcal{D}$ be a simple domain description

in $\mathcal{L}_1^{\mathrm{A}}$. Since simple domain descriptions have an explicit linear order over the situations and actions, we can disregard the `precedes` facts from the reduction, as they become redundant. Thus, we only translate the remaining laws and facts. The reduction $\Lambda(\mathcal{D}) = (\mathbf{P}, \mathcal{S})$ of $\mathcal{D}$ is defined in Figure 4.6. For the reduction, we map each situation s_i in $\mathcal{D}$ to the database state $\mathbf{d}_i$. In addition,

Fact or Law	$\mathcal{L}_1^{\mathrm{A}}$	$\mathcal{TR}^{PAD}$
Causal Law	a `causes` f `if` $b_1 \in \mathcal{D}$	$b_1 \otimes a \rightarrow a \otimes f \in \mathbf{P}$
Fluent Fact	f `at` $s_i \in \mathcal{D}$	$\mathbf{d}_i \rhd f \in \mathcal{S}$
Occurrence Fact	a `occurs_at` $s_i \in \mathcal{D}$	$\mathbf{d}_i \overset{a}{\rightsquigarrow} \mathbf{d}_{i+1} \in \mathcal{S}$

Figure 4.6: Reduction for the accurate fragment of $\mathcal{L}_1$ to $\mathcal{TR}_D^{PAD}$

we postulate that every fluent in $\Lambda(\mathcal{D})$ is inertial in every database state. That is, for every fluent f and database state $\mathbf{d}$,

$$\mathbf{P}, \mathcal{S}, \mathbf{d} \models \mathit{inertial}(f)$$

Thus, since $\mathit{inertial}(f)$ is "always true" for every fluent and every state, to avoid tedious repetition we remove the *inertial* predicate from every rule in the action theory without harming the semantics of the rule. In addition, we include in the reduction the action theory of $\mathbf{P}$. Recall that the action theory $\mathcal{A}(\mathbf{P})$ of a transaction base $\mathbf{P}$ consists of $\mathbf{P}$ and the frame axioms. Note that the translation of occurrence-facts of $\mathcal{L}_1$ takes care of both the facts of the form a_i `occurs_at` s_i and s_i `precedes` a_{i+1} since, by definition, these two types of formulas are encoded in the order of actions in an explicit actual path. Recall that the query language in $\mathcal{L}_1$ consist of hypothesis of the form $q = f$ `after` $[a_1, a_2, \ldots, a_n]$ `at` s_i. Therefore, to check soundness we restrict our query language to statements of the form: $a_1 \otimes a_2 \otimes \cdots \otimes a_n \otimes f$.

4.3.1. Theorem. ***(Soundness)*** *Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}^{PAD}$ reduction of D. Suppose that $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models \alpha \otimes f$. Then $D \models$ `f after` α `at` $\mathtt{s_i}$.*

14. Proof. *See Appendix.* □

As illustrated in the previous example, $\Lambda(D)$ is incomplete with respect to D. However, $\Lambda(D)$ is complete with respect to $\mathbf{P}(D)$ as stated in the following theorem.

4.3.2. Theorem (Completeness with Respect to $\mathbf{P}_D$). *Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}_D^{PAD}$ reduction of D. Suppose $\mathbf{P}_D \models \mathit{true\_after}(f, r, s_i)$. Then $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \models r \otimes f$.*

15. Proof. *See Appendix.* □

4.4 Planning: $\mathcal{L}_1$ vs $\mathcal{TR}^{PAD}$

We now turn to the problem of planning [83] agents' actions and compare the capabilities of $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$. Planning has been used in the areas of robotics, computer-aided design, manufacturing, computer graphics, aerospace applications, drug design, etc [83]. In planning, one starts with an initial state and action specifications and seeks to find a sequence of actions that lead to a state with certain desirable properties. We will show how the two languages approach the corresponding modeling and reasoning tasks.

Planning in $\mathcal{L}_1$. In [8], the initial state and the goal are represented by fluent facts, and the action descriptions by causal laws. However, $\mathcal{L}_1$ is not expressive enough to encode a *planning strategy*, since it expresses recursion which is needed to search over the space of all possible plans. To cope with this problem, $\mathcal{L}_1$ is embedded in LP and planning strategies are then expressed in that larger setting.

Let $\mathcal{D}$ be an $\mathcal{L}_1$ domain description, s_0 the initial state, and *goal* the planning condition (i.e., we are looking for a sequence of actions that leads to a state satisfying *goal*). Given a sequence of actions α_i, let $\mathcal{D}^{\alpha_i}$ denote the following domain description:

$$\mathcal{D} \cup \{s_j \texttt{ precedes } s_{j+1}, a_j \texttt{ occurs\_at } s_j \mid j = 0 \ldots i\}$$

The planner for $\mathcal{D}$ can be described by the following loop, where initially $N = 0$.

1. Generate all possible sequences of actions $\alpha_1 \ldots \alpha_m$ of length N,
2. For every domain $\mathcal{D}^{\alpha_i}$, $i = 1 \ldots m$, check: $\mathcal{D}^{\alpha_i} \models$ *goal* `after` α_i.
   If true, the goal has been reached and the plan α_i is returned.
3. Increase N, and go to Step 1.

This program generates all the plans that satisfy the goal, and guarantees that the shortest plan will be found first.

Planning in $\mathcal{TR}^{PAD}$. In [11], it was shown that planning strategies can be represented directly as $\mathcal{TR}$ rules and transactions, and plans could then be found by simply executing suitable transactions. Here we extend that formulation to model *conditional* planning. We also make use of premise statements and PADs to support more complex planning problems, including planning in the presence of incomplete information. The purpose here is to demonstrate that planning is possible, not to find the shortest plan. Additional techniques found in [11], including "script-based planning" and "locking," can also be ported to $\mathcal{TR}^{PAD}$. Suppose we have a planning problem consisting of *(i)* an initial state $\mathbf{d}_0$, *(ii)* a set of actions $\mathbf{A}$ consisting of n elementary actions defined as PADs and m compound actions, a set of PADs $\mathbf{P}_{PAD}$ and rules $\mathbf{P}_{rule}$ for the actions in $\mathbf{A}$. and *(iii)* a planning goal *goal*. Then, the $\mathcal{TR}^{PAD}$ representation of a planner consists of:

1. A set of *state*-premises that model the knowledge about the initial state $\mathbf{d}_0$,

2. A new set of actions $\mathbf{A}' = \mathbf{A} \cup \mathbf{A}^s$, where $\mathbf{A}^s = \{a^s \mid a \text{ is a } pda \text{ in } \mathbf{A}\}$. The new actions a^s are *compound* (not *pdas*, although they are created out of *pdas*).
   The new set of PADs, $\mathbf{P}'_{PAD}$, has a PAD of the form $b_1 \otimes a \otimes b_2 \rightarrow b_3 \otimes a \otimes b_4 \otimes \mathit{succeeded}_a$ for each PAD of the form $b_1 \otimes a \otimes b_2 \rightarrow b_3 \otimes a \otimes b_4$ in $\mathbf{P}_{PAD}$. Intuitively, $\mathit{succeeded}_a$ is used to test that the *pda* a was executed successfully.
   The set of rules $\mathbf{P}'_{rule}$ contains a rule r' for every rule r in $\mathbf{P}_{rule}$, where r' is an exact copy of r except that each *pda* $a \in \mathbf{A}$ that occurs in the body of r is replaced in r' with the corresponding compound action $a^s \in \mathbf{A}^s$. Also, for each pda $a \in \mathbf{A}$ and the corresponding new compound action $a^s \in \mathbf{A}^s$, $\mathbf{P}'_{rule}$ has a new rule of the form $a^s \leftarrow a \otimes \mathit{succeeded}_a$.

3. $\mathbf{A}'$ has two additional compound actions: *plan* and *act*. Intuitively, *act* is a generic action and *plan* represents sequences of such—generic—actions. $\mathbf{P}'_{rule}$ includes additional rules that define *plan* and *act*, as shown below.

4. A set of *run*-premises that encodes the possible executions of the *pdas*, described below.

5. A transaction of the form $\mathit{plan} \otimes \mathit{goal}$, whose execution is expected to produce the requisite plan.

The encoding of Planning in $\mathcal{TR}^{PAD}$ is composed by the following premises, PADs, and serial-Horn rules:

Initial State For each fluent $f \in \mathbf{d}_0$

$$\mathbf{d}_0 \rhd f$$

Actions For each PAD in $\mathcal{A}$ of the form $b_1 \otimes a \otimes b_2 \rightarrow b_3 \otimes a \otimes b_4$

$$b_1 \otimes a \otimes b_2 \rightarrow b_3 \otimes a \otimes b_4 \otimes \mathit{succeeded}_a$$
$$a^s \leftarrow a \otimes \mathit{succeeded}_a$$

Auxiliary Rules Let $c_1 \ldots c_m$ be the compound actions in $\mathcal{A}$ where each pda a is replaced by a^s; and let *skip* be an action that does not cause a state

change. Then

$$\begin{array}{l} \mathit{plan} \leftarrow \mathit{act} \otimes \mathit{plan} \\ \mathit{act} \leftarrow a_1^s \\ \quad \cdots \\ \mathit{act} \leftarrow a_n^s \\ \mathit{act} \leftarrow c_1 \\ \quad \cdots \\ \mathit{act} \leftarrow c_m \\ \mathit{act} \leftarrow \mathit{skip} \end{array}$$

Goal Let *goal* be the planing goal. Then such goal is represented by the following transaction:

$$?- \mathit{plan} \otimes \mathit{goal}$$

Executions for each pda $a \in \mathcal{A}$ and a sequence r consisting of the indices $\{1 \ldots n\}$

$$\mathbf{d}_r \overset{a}{\rightsquigarrow} \mathbf{d}_{r'}$$

The key features of $\mathcal{TR}^{PAD}$ that enable this sort of general representation of planning are: *(i)* Premises and PADs are used to describe the content of the initial state, the frame axioms, and the effects of the actions; *(ii)* Compound Actions, which allow combining simpler actions into complex ones in a modular way; *(iii)* Recursion allows the inference system to use the generate-and-test method for plans; *(iv)* Non-determinism allows actions to be executed in different ways and, together with recursion, supports exploration of the search space of all possible plans. Finally, the last two features also enable one to define rules that produce various heuristic-directed searches available in various advanced planning strategies.

Observe that neither of the above solutions guarantees termination, but both can be restricted by putting an upper limit on the maximum length of the plans.

4.4.1. Example. [Planning] Consider the specification $\mathcal{T}_H$ and the domain description $\mathcal{D}_H$ for the health care scenario of Example 4.2.1. Recall that our goal is to find a *legal* plan that makes the patient John (who is a flu-afflicted doctor and one who lacks immunity) into an immune and healthy person. That is, our planning goal is $g_1 = \mathit{immune} \wedge \mathit{healthy}$. We will examine the behavior of the planners in both formalisms.

The case of $\mathcal{L}_1$: The LP program would start checking if $\mathcal{D}_H \models g_1$ `after` [ ]. Since the goal is not satisfied in $\mathcal{D}_H$ it would increase N and try sequences of length 1. The planner will find the plan [*take_antivir*, *vaccinate*]. when $N = 2$. However, this plan is not compliant with the law as required. This "illegal" plan was found because the goal does not represent the meaning of *legal* plans.

(1) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathbf{neg}\ \mathit{immune}$
(2) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \mathit{take\_antivir}^s \otimes \mathit{healthy}$
(3) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2\mathbf{d}_3 \models \mathit{request\_authorization}^s \otimes \mathit{authorized}$
(4) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_3 \models \mathit{plan} \otimes \mathbf{neg}\ \mathit{immune} \wedge \mathit{authorized}$
(5) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3\mathbf{d}_4 \models \mathit{vaccinate}^s \otimes \mathit{immune} \wedge \mathit{healthy}$
(6) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\text{---}\mathbf{d}_4 \models \mathit{plan} \otimes \mathbf{neg}\ \mathit{immune} \wedge \mathit{authorized} \otimes \mathit{plan} \otimes \mathit{immune} \wedge \mathit{healthy}$

Figure 4.7: Planner execution in $\mathcal{TR}^{PAD}$

Unfortunately, the query language of $\mathcal{L}_1$ is not expressive enough to deal with the requirement that the patient must obtain an authorization *before* vaccination. One could try the goal $g_1 \wedge \mathit{authorized}$, but it is easy to see that this goal can lead to illegal plans as well. A solution could be to remove *vaccinate* from the action description to avoid the bad plans but this weakens the domain description and might block other desired inferences.

The case of $\mathcal{TR}^{PAD}$: As described earlier, we need to transform the specification into the following (we show only the main parts):

$\mathit{healthy} \otimes \mathit{vaccinate} \rightarrow \mathit{vaccinate} \otimes \mathit{immune} \otimes \mathit{healthy} \otimes \mathit{succeed}_{\text{vaccinate}}$
$\mathit{vaccinate}^s \leftarrow \mathit{vaccinate} \otimes \mathit{succeed}_{\text{vaccinate}}$
$\mathit{vaccinate\_legally} \leftarrow \mathit{request\_authorization}^s \otimes \mathit{authorized} \otimes \mathit{vaccinate}^s$
$\mathit{plan} \leftarrow \mathit{act} \otimes \mathit{plan}$
$\mathit{act} \leftarrow \mathit{vaccinate}^s$
$\mathbf{d}_{[\,]} \overset{vaccinate}{\rightsquigarrow} \mathbf{d}_{[1]}$

Since we saw that the goal g_1 may lead to bad plans, we will modify the goal to specify that the patient can be vaccinated only after getting an authorization. This can be expressed as follows: $g_2 = \mathit{plan} \otimes (\mathbf{neg}\ \mathit{immune} \wedge \mathit{authorized}) \otimes \mathit{plan} \otimes (\mathit{immune} \wedge \mathit{healthy})$. The goal states that the planner must first try to obtain an authorization while the patient is still not immunized. Having achieved that, the planner will go on and plan for immunizing the patient and making her healthy. Note the ability of $\mathcal{TR}^{PAD}$ to specify intermediate conditions that the planner must achieve, not just the final goal. This is not possible in $\mathcal{L}_1$ without complicated encoding. The $\mathcal{TR}^{PAD}$ planner will construct a desired plan while proving the goal g_2 from the above specification at the initial state $\mathbf{d}_0$. Figure 4.7 illustrates how this works. The resulting plan will be $\mathit{take\_antivir} \otimes \mathit{request\_authorization} \otimes \mathit{vaccinate}$, which is equivalent to $\mathit{take\_antivir} \otimes \mathit{vaccinate\_legally}$. □

4.5 Relationship with Other Action Languages

We will now briefly compare $\mathcal{TR}^{PAD}$ with several well-known action languages, which provide interesting features not present in $\mathcal{L}_1$.

The $\mathcal{ALM}$ language [46]. This action language introduces the following features that $\mathcal{L}_1$ lacks: defined fluents, modular definition of actions, sorts, executability conditions and a form of concurrency. Although in $\mathcal{ALM}$ one can describe the effects and hierarchies of actions, and define fluents based on other fluents, one cannot (i) express the execution of actions like occurrence facts in $\mathcal{L}_1$ and *run*-premises in $\mathcal{TR}^{PAD}$ do, or (ii) assert information about the states, like fluent facts in $\mathcal{L}_1$ and *state*-premises in $\mathcal{TR}^{PAD}$ do. Recursion is disallowed for actions, but it is allowed for fluents.

$\mathcal{TR}^{PAD}$ can express most of these new features easily: defined fluents are expressed with fluent rules, modular definition of actions is done using compound actions, sorts can be emulated by predicates, and executability conditions can be represented as in our planning example (the a^s type of compound actions). However, $\mathcal{TR}^{PAD}$ does not yet handle concurrency.

The $\mathcal{C}$ language [36]. This language is based on the theory of causal explanation. That is, everything that is true in a state must be caused. This implies that the frame axioms are not part of the semantics but are expressed as axioms. In that sense, $\mathcal{TR}^{PAD}$ is closer to $\mathcal{C}$ than to $\mathcal{L}_1$. The language $\mathcal{C}$ is the simplest among the formalisms mentioned so far. It only allows causal laws and fluent definitions of the form: **caused** F **if** G and **causes** F **if** G **after** H, where F, G, H are propositional formulas, and only H can contain actions. Note that H may contain more than one action, which leads to concurrency in causal laws. Although causal laws can contain disjunctions in the rule conditions and effects, which is disallowed in PADs, in the propositional case disjunction can be modeled in $\mathcal{TR}^{PAD}$ by splitting rules. In this way, $\mathcal{TR}^{PAD}$ can model non-concurrent $\mathcal{C}$ domain description. In addition, [36] also shows how to encode forward-reasoning frame axioms, but $\mathcal{C}$ is not expressive enough to solve problems that involve backward reasoning, which is easily done in $\mathcal{TR}^{PAD}$.

Situation Calculus and Golog [48, 71, 56]. In the Situation Calculus (SC) a domain description is composed of the following axioms:

- An axiom for each action in the language specifying the action preconditions. Action preconditions are conjunction of fluent literals.
- For each fluent, the *successor state axioms* which describe the effect of the different actions on that fluent. These axioms also take care of encoding the inertia laws.
- Axioms describing defined fluents.
- The *foundational axioms* of the situation calculus. The description of these axioms are beyond the scope of this work, further details can be found in [48], [71].

There are two important features in $\mathcal{TR}^{PAD}$ that are lacking in SC: hypothetical formulas (which may include actions), and a direct connection between the precondition of the action and its effect. In SC the preconditions of actions are specified separately from their effect, so it is rather difficult to specify different effects for an action that depend on different preconditions. The lack of hypothetical formulas is a more delicate issue since, for instance, these kind of statements are necessary to model production systems augmented with ontologies (c.f. Chapter 5). SC also lacks the flexibility to specify which actions are subject to the inertia laws in which state.

Some features, like recursion, sequence of actions, and complex actions, were absent in the earlier versions of SC but were incorporated later on, when Golog was defined. However, Golog is not a logic, but an imperative programming language based on SC. It inherits from SC the limitations regarding hypothetical reasoning and the ability to easily define effects based on different preconditions.

To summarize, $\mathcal{TR}^{PAD}$ offers a more modular, succinct, and clear way of specifying action preconditions and effects in the form of PADs. It gracefully supports hypothetical tests, including hypothetical actions, that are very useful in many scenarios, such as preventing undesired executions. It is worth noting that $\mathcal{TR}^{PAD}$ does not require foundational axioms of SC and states do not occur as arguments of actions or of fluents, unlike situations in SC. All these features coexist within a single logical language with a single unifying model and proof theory; $\mathcal{TR}^{PAD}$ does not resort to an external imperative language to provide action composition and other basic features.

Fluent Calculus and Flux [76, 62]. The Fluent Calculus (FC) deviates from SC by introducing states instead of situations and by specifying the effects of actions using action-based state update axioms (as opposed to SC's successor state axioms). These axioms also take care of the inertia laws. As in SC, FC theories need a set of foundational axioms. FC also has Flux, a high-level programming language. Like Golog, it is not a logical language, but an imperative language that operates with logical statements. FC allows nondeterministic actions, looping actions, and defined fluents. In that sense FC is closer to $\mathcal{TR}^{PAD}$ than SC. However, FC (and Flux) offer concurrent actions which is lacking in $\mathcal{TR}^{PAD}$.

On the other hand, $\mathcal{TR}^{PAD}$ allows complex hypothetical tests including hypothetical actions, complex actions, a simple and modular way of expressing the laws of inertia, and it does not require a complex set of foundational axioms or an external non-logical language.

Event Calculus [51]. Event calculus (EC) is a methodology for specifying actions in logic programming. It includes predicates for describing the initial situation, the effects of actions, and for specifying which fluents hold at what

times. The event calculus solves the frame problem in a way that is similar to the successor state axiom but relies on non-monotonic aspects of logic programming (unlike SC, which is completely first-order). It is capable of representing a variety of features present in $\mathcal{TR}^{PAD}$, like defined fluents, actions with non-deterministic effects, compound actions. It also has some features that are not present in $\mathcal{TR}^{PAD}$, like parallel actions. However, EC does not support hypothetical actions and recursion. As in SC, fluents have a time point as an argument, this together with the successor state axioms make it is harder the combination of this formalism with Ontologies

In summary, $\mathcal{TR}^{PAD}$ offers a powerful combination of features for action representation most of which are not present in any one of the other systems. These include recursion, non-determinism, compound and partially defined actions, hypothetical reasoning, forward and backward reasoning in time, and sound and complete proof theory. Nevertheless, $\mathcal{TR}^{PAD}$ does not completely subsume any of the other systems discussed in this chapter, for it does not support concurrency and interloping partial action definitions.

4.6 Considering $\mathcal{TR}^{PAD}$ with default negation

One of the main limitation that $\mathcal{TR}^{PAD}$ has with respect to these action languages is that $\mathcal{TR}^{PAD}$ does not allow interloping action definitions. However, as we have seen in Section 3.8, when we extend $\mathcal{TR}^{PAD}$ with default negation we can lift this restriction and strongly increase the expressive power of $\mathcal{TR}^{PAD}$. The exact comparison of the expressive power of $\mathcal{TR}^{PAD}$ with default negation and Situation Calculus, Event Calculus, Fluent Calculus, $\mathcal{C}$ and $\mathcal{ALM}$ is a topic for future research.

4.7 Summary of the Contributions

In this chapter we explored and compared two expressive formalisms for reasoning about actions: $\mathcal{TR}^{PAD}$ (without default negation) and $\mathcal{L}_1$. We have shown that these formalisms have different capabilities and neither subsumes the other. Nevertheless, we established that a large subset of $\mathcal{L}_1$ [8] can be soundly represented in $\mathcal{TR}^{PAD}$ and that the LP reduction of that subset is logically equivalent to the corresponding subset of $\mathcal{TR}^{PAD}$. We also compared the two logics in the domain of action planning and showed that $\mathcal{TR}^{PAD}$ has a significant advantage when it comes to planning under constraints. We sketched as well the relation between $\mathcal{TR}^{PAD}$ with other action languages: $\mathcal{C}$ [36], and $\mathcal{ALM}$ [46], Situation Calculus and Golog [48, 71, 56], Fluent Calculus and Flux [76, 62], and Event Calculus [51].

Another interesting aspect of $\mathcal{TR}^{PAD}$ (without default negation) is that it is based on a *monotonic* logic—unlike all the above approaches.

Finally, we briefly discussed how $\mathcal{TR}^{PAD}$ with default negation is related to the action languages mentioned above. In particular, we pointed out that with default negation we can remove one of the main limitations that $\mathcal{TR}^{PAD}$ has with respect to the other action languages: interloping actions.

Chapter 5

Modeling Production Systems

> Now it happens that turtles are great speed enthusiasts, which is natural. The *esperanzas* know that and don't bother themselves about it. The *famas* know it, and make fun of it. The *cronopios* know it, and each time they meet a turtle, they haul out the box of colored chalks, and on the rounded blackboard of the turtle's shell they draw a swallow.
>
> Julio Cortazar
> Historias de Cronopios y de Famas

In this chapter we explore how to combine production systems (PSs) and Description Logics ontologies and how to give a declarative semantics to the combination of production systems and *rule-based* ontologies.

Production systems are one of the oldest knowledge representation paradigms that are still popular today. Production systems are widely used in biomedical information systems, to enforce constraints on databases, to model business processes, accounting, etc.

Such systems consist of a set of *production rules* that rely on forward chaining reasoning to update the underlying database, called *working memory*. Traditionally, PSs have had only operational semantics, where satisfaction of rule conditions is checked using pattern matching, and rule actions produce assertion and deletions of facts from the working memory.

Informally, given a working memory, the rule interpreter applies rules in three steps: (1) *pattern matching*, (2) *conflict resolution*, and (3) *rule execution*.

In the first step, the interpreter decides—typically using the RETE algorithm [32]—for each rule r and for each variable substitution $\mathcal{S}$ whether r can be applied in the working memory using $\mathcal{S}$. This step returns all pairs $(r, \mathcal{S})$ such that r can be applied using $\mathcal{S}$; this set is called the *conflict resolution set*. In step (2), the interpreter chooses zero or one pair from the conflict resolution set; in case the set is empty or no pair is chosen, the system terminates. In the last step, the working memory is updated following the additions and removals in the action part of the selected rule. The interpreter then starts again with step (1).

PSs syntax and semantics have been standardized as W3C's Production Rule Dialect of the Rule Interchange Format (RIF-PRD) [80]. The RIF-PRD specification has a number of limitations, however. First, it omits certain important primitives that are found in many commercial production systems such as IBM's *JRules*[49]. The *FOR*-loop and the *while*-loop constructs are examples of such an omission. Second, RIF-PRD still does not integrate with ontologies [6, 42]. Here, by ontology we mean a formal representation of a domain of interest, expressed in terms of concepts and roles, which denote classes of objects and binary relations between classes of objects, respectively.

5.0.1. Example. To illustrate the need for ontology integration, consider a set of PSs that keeps a number of clinical databases that are compliant with the health insurance regulations. The clinical record of each patient together with other data must be accessible by all the clinics in the network. This needs a shared vocabulary that, in this case, is defined in a shared DL ontology. However, each PS can have extra concepts outside the ontology, which are meant for local use only. The following production rules state that *(i)* if a doctor D requests a DNA test T to be performed for patient P, then the system records that P is taking the test T; and *(ii)* patients getting a DNA test must not be considered unhealthy.

r_1: `Forall` D, P, T: `if` $\mathit{requested}(D, P, T) \wedge \mathit{dnaT}(T)$ `then` $\mathit{Assert}(\mathit{takesT}(P, T))$
r_2: `For` P, T: $\mathit{takesT}(P, T) \wedge \mathit{dnaT}(T)$ `do` $\mathit{Retract}(\mathbf{neg}\ \mathit{healthy}(P))$

The DL ontology that defines the shared concepts and implements different constraints is as follows

$$\mathit{flu} \sqsubseteq \mathbf{neg}\ \mathit{healthy} \qquad \mathit{dnaT} \sqsubseteq \mathbf{neg}\ \mathit{virusT} \qquad \exists \mathit{takesT}.\mathbf{neg}\ \mathit{virusT} \sqsubseteq \mathit{healthy}$$

The DL axioms say that a patient with a flu is not a healthy patient, that DNA tests do not search for viruses, and that if a person is taking a test not related with any virus disease, then we can conclude that she is healthy. □

The `Forall` construct in r_1 should not be confused with the `For` construct in r_2. The former is just a way RIF-PRD declares variables used in the body

of a rule. The latter is a FOR-loop extension found in commercial systems, but not in RIF-PRD.

Note that in the ontologies one can have both **neg**- and ¬-literals, while $\mathcal{TR}^{PAD}$ uses **neg**- and **not**-literals instead. This is because logic programming rules cannot use classical negation, while ontologies do not use default negation.

The complexity of the regulations in our example makes it difficult to determine whether executing a production rule leaves the database in a compliant state. Suppose we have the following initial database

$$\mathsf{WM}_0 = \{\mathit{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathit{flu}(\text{Laura}), \mathit{dnaT}(\text{pcr})\}$$

This example raises several question. Three of the most relevant ones are:

- *How do we check if a rule condition holds in* WM_0*?* In traditional PS semantics, WM_0 is viewed as a unique model in which we can check the satisfaction of formulas. However, under DL semantics, WM_0 would be seen as theory (ABox) that together with the TBox has a possibly infinite set of models, and thus entailment is needed.

- *How do we interpret the retraction of an atom as stated in rules* r_1 *and* r_2*?* In traditional PS semantics, to retract an atom is equivalent to changing the truth value of a fact from true to false since there is a unique model. It is a simple operation that is achieved by removing the fact from the working memory. In DL to retract a fact that is entailed by the knowledge base or to enforce the knowledge base to entail a fact to be false, is a complex problem that cannot always be solved [58, 26]. In particular, how do we interpret the retraction executed by r_2, where P is instantiated with *Laura*, given that **neg** *healthy*(Laura) is inferred by the ontology?

- *What do we do when an inconsistency arise?* In traditional PSs working memories contain only positive atoms and since rules can only assert and retract facts, there is no room for inconsistencies. Clearly, this does not hold anymore once we allow negative facts and ontologies. In particular, how do we treat the inconsistency that results after execution of rule r_1 in WM_0? (Observe that in the state resulting from execution of r_1 in WM_0 we can infer *healthy*(Laura) and **neg** *healthy*(Laura)).

To answer these questions we need to define a precise semantics (both model-theoretic and computational) to the combination of rules, ontologies, and production systems.

Our contribution in this chapter is two-fold: *(i)* a new semantics for production systems augmented with *DL* ontologies that includes looping-rules, and can handle inconsistency; *(ii)* a sound embedding of the combination of PS and

rule-based ontologies into $\mathcal{TR}^{PAD}$ which provides a model-theoretic semantics to the combination;

Our formalization is significantly more general than RIF-PRD or other existing formalizations of production rules in that it supports wider ontology integration and covers important extensions that exist in commercial systems such as the aforesaid *FOR*-loop.

This work continues the line of research started in [25, 72], but here we target a more complex and standard definition of production systems.

This chapter is organized as follows. Section 5.1 presents the necessary background on Description Logic needed to understand this chapter. Section 5.2 briefly surveys previous results on the combination of PS and ontologies, and on the reduction of PSs to formalisms with model-theoretic semantics. Section 5.3 introduces an operational semantics for production systems augmented with *DL* ontologies. Section 5.4 provides a reduction from the semantics proposed here to $\mathcal{TR}^{PAD}$ and presents soundness results for this reduction. Section 5.5 concludes the chapter.

5.1 Background on Description Logic

In this Section we briefly review the basic notions from Description Logic (DL) that we will use in this chapter. Details can be found in [6].

Description Logic is a family of knowledge representation formalisms that provide a syntax and a model-theoretic semantics for a compact representation of information.

In DL, the domain of interest is modeled by means of *concepts*, that represents sets of objects, and *roles*, that are binary relations between objects. Complex concepts and roles can be obtained from atomic ones using suitable constructs.

A DL knowledge base (KB) has two parts: the TBox, with terminological knowledge, which consists of a number of class definitions, and the ABox, which consists of assertions about actual individuals.

Concept axioms in the *TBox* are of the form $C \sqsubseteq D$ (meaning the extension of C is a subset of the extension of D; D is more general than C) or $C \equiv D$ (where $C \equiv D$ is interpreted as $C \sqsubseteq D$ and $D \sqsubseteq C$) with C and D (possibly complex) descriptions. Given a TBox axiom of the form $C \sqsubseteq D$, the concept C is called the *body*, and D is called the *head* of the axiom.

Descriptions and TBox axioms can be understood as formulas of first-order logic with one free variable and closed universal formulas. For example, the description $A \sqcap \neg B \sqcap \exists R.C$ corresponds to the formula $A(x) \wedge \neg B(x) \wedge \exists y.(R(x,y) \wedge C(y))$. Therefore, the semantics of DL can be given by its translation to FOL. Details can be found in [6].

In this chapter we will use DLs that can be embedded into Logic Programming (LP). In recent years, the relationship between DLs and LP has attracted much interest and several LP-expressible DLs have been proposed [45, 63, 64, 28, 53, 41]. In particular, [41] defines a class of DLs called *Datalog-rewritable DLs*. This class is interesting in our setting because reasoning with DLs in such a class can be reduced to reasoning with logic programming, more precisely, with Datalog programs [2]. See Section 2.2 for further details on LP.

5.1.1. DEFINITION. [Datalog-rewritable] A DL $\mathcal{D}$ is ***Datalog-rewritable*** if there is a transformation dtg from $\mathcal{D}$ to Datalog programs such that for any knowledge base $(\mathcal{T}, \mathcal{A})$ in $\mathcal{D}$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, the following holds: For any concept or role name Q, and an individuals $\vec{i}$ in $(\mathcal{T}, \mathcal{A})$,

$$(\mathcal{T}, \mathcal{A}) \models Q(\vec{i}) \text{ iff } \mathsf{dtg}((\mathcal{T}, \mathcal{A})) \models Q(\vec{i})$$

We say that dtg is ***modular*** if

$$\mathsf{dtg}((\mathcal{T}, \mathcal{A})) = \mathsf{dtg}(\mathcal{T}) \cup \mathsf{dtg}(\mathcal{A}) \qquad \square$$

One Datalog-rewritable DL is $\mathcal{LDL}^+$ [41]. For concreteness, in Section 5.3 we will work with this DL, but our results do not depend on a particular choice of a Datalog-rewritable DL.

$\mathcal{LDL}^+$ is defined by restricting the shape of the axioms in the TBox, as shown in Figure 5.1. Further details and a reduction to Datalog can be found in [41].

An $\mathcal{LDL}^+$ KB is a pair $(\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a finite TBox and $\mathcal{A}$ is a finite ABox such that

- $\mathcal{T}$ is a set of *terminological axioms* of the form $C \sqsubseteq D$, where C is a body concept and D is a head concept; and *role axioms* $E \sqsubseteq F$, where E is a basic role and F is a head role.
- $\mathcal{A}$ is a set of assertions of the form $D(o)$ and $F(o_1, o_2)$ where D is a head concept and F is a head role.

5.1.2. EXAMPLE. [41] Suppose we have the following knowledge base $(\mathcal{T}, \mathcal{A})$:

$$\mathcal{T} = \left\{ \begin{array}{l} \geq 2 PapersToRev \sqsubseteq OverL \\ OverL \sqsubseteq \forall Superv^+.OverL \end{array} \right. \quad \mathcal{A} = \left\{ \begin{array}{l} Superv(a, b) \\ Superv(b, c) \end{array} \right.$$

The first two axioms say that someone with more than two papers to review is overloaded and that an overloaded person causes all the supervised persons to be overloaded as well. The symbol + over the role *Superv* stands for the transitive closure of the role. The statements in the ABox defines the supervision hierarchy. $\square$

Body Roles			
E_1, E_2	$\rightarrow$	P	(Role name)
		E_1^-	(inverse)
		$E_1 \circ E_2$	(Sequence)
		E_1^+	(Transitive Closure)
		$E_1 \sqcap E_1$	(Conjunction)
		$E_1 \sqcup E_2$	(Disjunction)
		$\top^2$	(Top)
		$\{o, o\}$	(Nominals)
Head Roles			
F_1, F_2	$\rightarrow$	P	(Role name)
		F_1^-	(inverse)
		$F_1 \sqcap F_2$	(Conjunction)
		$\top^2$	(Top)
Body Concepts			
C_1, C_2	$\rightarrow$	A	(Concept name)
		$\exists E_1.C_1$	(Existential Restriction)
		$\leq nE_1.C_1$	(AtLeast Restriction)
		$C_1 \sqcup C_2$	(Disjunction)
		$C_1 \sqcap C_2$	(Conjunction)
		$\{o\}$	(Nominals)
		$\top$	(Top)
Head concepts			
D	$\rightarrow$	C_1	(Basic Concept)
		$\forall E_1.C_1$	(Universal Restriction)

Figure 5.1: $\mathcal{LDL}^+$ Syntax

We conclude this brief introduction of $\mathcal{LDL}^+$ with a review of its relationship to OWL 2 fragments [18].

- **OWL 2 EL**. This fragment correspond to the DL $\mathcal{EL}^{++}$[5]. This fragment and $\mathcal{LDL}^+$ do not subsume each other. Not all $\mathcal{EL}^{++}$ axioms can be expressed in $\mathcal{LDL}^+$ but those that do not contain $\bot$, concrete domains, and existential quantification in axioms' right side. On the other hand, $\mathcal{LDL}^+$ has many constructs that $\mathcal{EL}^{++}$ does not allow: number restrictions, inverse, general sequence of roles, role conjunction, role disjunction, etc.

- **OWL 2 QL**. This fragment correspond to the DL-*Lite* family. Again, this fragment neither subsumes nor is subsumed by $\mathcal{LDL}^+$. DL-Lite axioms that have no negation and existential quantification on the right side are also axioms in $\mathcal{LDL}^+$, but $\mathcal{LDL}^+$ has constructs that are not expressible in DL-*Lite*, such as role sequence.

- **OWL 2 RL**. This fragment is a strict subset of $\mathcal{LDL}^+$.

5.2 Related Work

In this section we compare our approach with other literature on the declarative semantics for production systems and on the operational and declarative semantics for the combination of PS and ontologies. The work described in [72, 25] provides an operational and model-theoretic semantics to the combination of PS and ontologies. The model-theoretic semantics is given by an embedding of PSs into fix-point logic. However, they cannot handle looping rules, their semantics cannot handle inconsistencies, their interpretation of retraction of DL facts is not intuitive since a fact can remain true after being deleted, and their reduction to a declarative formalism is considerably more complex than the one presented here. In [55, 85], the goal is to devise languages for unifying some aspects of active rules, logic rules, and production systems. They do not deal with considerably more complex standard languages such as production systems augmented with ontologies and looping rules. In particular, [55, 85] do not show how to embed production systems into those languages, although they provide some examples showing how typical production rules can be expressed in their language. In [70] the authors only allow a very restricted type of production systems: stratified PS. Such PS are much weaker that the ones formalized here, and again, they do not consider ontologies. In addition, they do not tackle the problem of the integration with ontologies. In [22, 9], the authors reduce the semantics of PS to logic programming (LP). Their reduction is considerably more complex and less compact than ours—it results in an infinite number of

rules. In addition, they use stable models semantics which has much higher computational complexity than the well founded semantics used here. Given the complexity of such a reduction, the proposed integration with LP ontologies is not ideal, since the ontology needs to be transformed with state arguments and auxiliary predicates. In addition, neither of them allow looping rules. Finally, [52] presents a new formalism that combines some aspects of logic rules and production rules. However, negation in rule conditions[1] and looping rules are disallowed. Furthermore, their embedding into Horn Logic is less clear and compact than our embedding in $\mathcal{TR}^{PAD}$.

5.3 Combining Production Systems and Ontologies

In this section we propose a new semantics for the combination of production systems and *arbitrary* DL ontologies. This approach follows the outline of [72], but includes looping rules, it can handle inconsistencies produced by the system, and it gives a more intuitive semantics to the retraction of DL facts.

Syntax

The alphabet of a language $\mathcal{L}_{\mathsf{PS}}$ for a production system is defined the same way as in the case of $\mathcal{TR}$ except that now the set of all predicates $\mathcal{P}$ is partitioned into two countably infinite subsets, $\mathcal{P}_{\mathrm{PS}}$ and $\mathcal{P}_{\mathrm{DL}}$. The latter will be used to represent predicates occurring only in the ontology. A **term** is either a variable or a constant symbol and, to avoid unnecessary distractions, unnecessary distractions, we will leave out the various additional forms allowed in RIF, such as frames and the RIF membership and subclass relations ($o\#t$, $t\#\#s$). However, they can easily be added without increasing the complexity of the problem.

5.3.1. Definition. [Atomic Formulas] Let $p \in \mathcal{P}$ be a predicate and $t_1, \ldots, t_n$ be terms. An **atomic formula** is a statement of the form $p(t_1 \ldots t_n)$. □

A **literal** is either an atom, a formula of the form **neg** f where f is a $\mathcal{P}_{\mathrm{DL}}$-atom, or a formula of the form $\neg f$ where f is a $\mathcal{P}_{\mathrm{PS}}$-atom.

5.3.2. Definition. [Condition Formula] A **condition formula** ϕ has one of the following forms:

- a literal l,
- $\phi_1 \wedge \phi_2$, where ϕ_1 and ϕ_2 are condition formulas.

[1] The authors informally claim that negation could be added, but they do not provide formal details.

- $\phi_1 \vee \phi_2$, where ϕ_1 and ϕ_2 are condition formulas. □

Observe that all the rule conditions in Example 5.0.1 are condition formulas.

An essential feature of production systems is the ability to perform actions such as insertion and deletion of atoms. We now define the concrete actions for accomplishing that.

5.3.3. DEFINITION. [Atomic Action] Let $p(\vec{c})$ be a literal. An **atomic action** is a statement that has one the following forms:

- ***assert***$(p(\vec{c}))$: Adds the literal $p(\vec{c})$ to the working memory
- ***retract***$(p(\vec{c})) : \begin{cases} \text{if } p \in \mathcal{P}_{\text{PS}} & \text{Removes the } atom^2 p(\vec{c}) \text{ from the working memory} \\ \text{if } p \in \mathcal{P}_{\text{DL}} & \text{Enforces the literal } p(\vec{c}) \text{ to be false in the working memory} \end{cases}$ □

Beside these elementary actions, RIF also provides actions to change or delete objects and properties. Such actions can be treated similarly to `FOR`-rules below or as sequences of simpler actions, so we leave them out as well.

5.3.4. DEFINITION. [Production System] A *Production System*, PS, is a tuple

$$\mathsf{PS} = (\mathcal{T}, \mathsf{L}, R)$$

such that

- $\mathcal{T}$ is a DL ontology (TBox) whose predicates belong to $\mathcal{P}_{\text{DL}}$;
- L is a set of rule labels, and
- R is a set of rules, which are statements of one of the following forms[3]

$$\texttt{IF-THEN Rule:} \quad r: \quad \texttt{Forall } \vec{x} \texttt{: if } \phi_r(\vec{x}) \texttt{ then } \psi_r(\vec{x}) \tag{5.1}$$

$$\texttt{FOR Rule:} \quad r: \quad \texttt{For } \vec{x} \texttt{: } \phi_r(\vec{x}) \texttt{ do } \psi_r(\vec{x}) \tag{5.2}$$

where

- $r \in \mathsf{L}$ is the above rule's label,
- ϕ_r is a condition formula in $\mathcal{L}$ with free variables $\vec{x}$,
- $\psi_r(\vec{x})$ is a sequence of atomic actions with free variables contained in $\vec{x}$.□

[2]Negative literals with predicate symbols in $\mathcal{P}_{\text{PS}}$ cannot occur in the working memories. See Definition 5.3.5.

[3] To avoid a misunderstanding, recall that the `Forall` construct is just a RIF-PRD syntax for declaring variables; it does not indicate a loop. In contrast, the `For-do` construct specifies a loop; it is found only in commercial PS, like JRules.

Operational Semantics

We now turn to the operational semantics of the combination of PS with ontologies. In a PS, two different constants represent two different domain elements (*unique name assumption*). In addition, production systems assume that each constant symbol is also a symbol in the domain of discourse, i.e., they deal with Herbrand domains.

It is also worth noting that the semantics presented in this section does not depend on the specifics of the DL associated with production systems. For concreteness, one can think of $\mathcal{LDL}^+$ [41].

5.3.5. Definition. [Working Memory] A **working memory**, WM, for a PS language $\mathcal{L}$ is a disjoint union

$$\mathsf{WM} = \mathsf{WM}_{\mathrm{PS}} \uplus \mathsf{WM}_{\mathrm{DL}}$$

where $\mathsf{WM}_{\mathrm{PS}}$ is a set of ground atoms that use predicate symbols from $\mathcal{P}_{\mathrm{PS}}$ and $\mathsf{WM}_{\mathrm{DL}}$ is a set of ground literals that use predicate symbols from $\mathcal{P}_{\mathrm{DL}}$. □

5.3.6. Definition. [$\mathcal{T}$-structure] Let $\mathcal{T}$ be a DL TBox. A ***$\mathcal{T}$-structure***, $\mathcal{I}$, for a PS language $\mathcal{L}$ has the form

$$\mathcal{M} = (\mathsf{WM}_{\mathrm{PS}} \uplus \mathsf{WM}_{\mathrm{DL}} \uplus \mathbf{E}_{\mathrm{DL}}, \sigma)$$

where $\mathsf{WM} = \mathsf{WM}_{\mathrm{PS}} \uplus \mathsf{WM}_{DL}$ is a working memory, $\mathbf{E}_{DL}$ is a set of $\mathcal{P}_{DL}$-literals, σ is a variable assignment, and $(\mathsf{WM}_{DL} \uplus \mathbf{E}_{DL}, \sigma)$ is a model of $\mathcal{T}$. □

We say that (WM, σ), where WM is a working memory, is a ***prestructure***.

5.3.7. Example. Consider Example 5.0.1. The two disjoint sets composing the initial working memory WM_0 are as follows:

$$\mathsf{WM}_{0\mathrm{PS}} = \{\mathit{requested}(\mathrm{Smith}, \mathrm{Laura}, \mathrm{pcr})\}$$
$$\mathsf{WM}_{0\mathrm{DL}} = \{\mathit{flu}(\mathrm{Laura}), \mathit{dna}\mathcal{T}(\mathrm{pcr})\}$$

In addition, we can build up a $\mathcal{T}$-structure, $\mathcal{M}$, by pairing any arbitrary assignment σ with WM_0 together with $\{\mathbf{neg}\ \mathit{healthy}(\mathrm{Laura})\}$. That is, $\mathcal{M} = (\mathsf{WM}_0 \uplus \{\mathbf{neg}\ \mathit{healthy}(\mathrm{Laura})\}, \sigma)$. □

5.3.8. Definition. [Satisfaction] A $\mathcal{T}$-structure $\mathcal{M} = (\mathsf{WM}_{\mathrm{PS}} \uplus \mathsf{WM}_{\mathrm{DL}} \uplus \mathbf{E}_{\mathrm{DL}}, \sigma)$ *satisfies* a literal l, denoted $\mathcal{M} \models l$, iff

- if l is a $\mathcal{P}_{\mathsf{PS}}$-atom then $l^{\mathcal{I}} \in \mathsf{WM}_{\mathrm{PS}}$
- if l is a $\mathcal{P}_{\mathrm{DL}}$-literal then $\mathsf{WM}_{\mathrm{DL}} \uplus \mathbf{E}_{\mathrm{DL}} \models l^{\mathcal{I}}$

If ϕ is a formula of the form $\neg\phi_1,\ \phi_1 \wedge \phi_2,\ \phi_1 \vee \phi_2$ then we define $\mathcal{M} \models \phi$ as usual in FOL. A formula ϕ **holds** in a prestructure (WM, σ) relative to an ontology $\mathcal{T}$, denoted $\mathcal{T}, (\mathsf{WM}, \sigma) \models \phi$, iff $\mathcal{M} \models \phi$ for every $\mathcal{T}$-structure of the form

$$\mathcal{M} = (\mathsf{WM} \uplus \mathbf{E}_{\mathrm{DL}}, \sigma)$$

(That is, WM and σ are fixed but the $\mathbf{E}_{\mathrm{DL}}$ varies.) □

A prestructure is ***$\mathcal{T}$-consistent*** if there is a $\mathcal{T}$-structure with the same working memory and variable assignment, i.e., $(\mathsf{WM} \uplus \mathbf{E}_{DL}, \sigma)$ that does not entail f and $\mathbf{neg}\, f$ for any atom f. Note that in such a $\mathcal{T}$-structure, if $\mathbf{neg}\, f$ is true then $\neg f$ is also. A working memory is ***$\mathcal{T}$-consistent*** if it is part of a $\mathcal{T}$-consistent prestructure.

5.3.9. Definition. [Atomic Transition] Let (WM, σ) be a prestructure, t_1, t_2 be terms, and α be an action. We say that there is an ***α-transition*** from the prestructure (WM, σ) to the prestructure (WM', σ), denoted $(\mathsf{WM}, \sigma) \overset{\alpha}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$, iff

- if $\alpha = \textbf{\textit{assert}}(p(t))$ then $\mathsf{WM}' = (\mathsf{WM} \cup \{p(t^\sigma)\}) \setminus \{\mathbf{neg}\, p(t^\sigma)\}$
- if $\alpha = \textbf{\textit{retract}}(p(t))])$ then $\begin{cases} \text{if } p \in \mathcal{P}_{\mathrm{PS}} & \mathsf{WM}' = \mathsf{WM} \setminus \{p(t^\sigma)\} \\ \text{if } p \in \mathcal{P}_{\mathrm{DL}} & \mathsf{WM}' = (\mathsf{WM} \cup \\ & \{\mathbf{neg}\, p(t^\sigma)\}) \setminus \{p(t^\sigma)\} \end{cases}$

where t^σ is $\sigma(t)$ if t is a variable and it is t if t is a constant. □

5.3.10. Definition. [Compound Transition] Let (WM_0, σ) be a prestructure and $\alpha_1 \cdots \alpha_n$ be atomic actions. We write

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

iff there are prestructures $(\mathsf{WM}_1, \sigma) \ldots (\mathsf{WM}_{n-1}, \sigma)$ such that

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma) \overset{\alpha_2}{\twoheadrightarrow} (\mathsf{WM}_2, \sigma) \overset{\alpha_3}{\twoheadrightarrow} \ldots \overset{\alpha_{n-1}}{\twoheadrightarrow} (\mathsf{WM}_{n-1}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma) \qquad \square$$

If, for some σ and $n \geq 1$, there is a transition $(\mathsf{WM}_0, \sigma) \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$ between prestructures then we will also write $\mathsf{WM}_0 \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} \mathsf{WM}'$.

Since actions may introduce inconsistencies with respect to the ontology, we need to define the notion of a consistent result of applying an action. Let α be an action and suppose that

$$\mathsf{WM} \overset{\alpha}{\twoheadrightarrow} \mathsf{WM}'$$

is a transition among prestructures such that WM is $\mathcal{T}$-consistent and WM' is not. One way to define a consistent result of applying an action to WM is to take a maximal subset of WM' that belongs to some $\mathcal{T}$-consistent prestructure. However, a maximal subset might not be unique. A workaround here is to take the intersection of all the possible consistent results. This approach is called *When in Doubt Throw it Out* (WIDTIO) [81]. This form of belief revision is in line with traditional ontologies and it has been also used in the context of evolution of DL knowledge bases [19]. It is worth noticing that here we are using a simple notion of belief revision for ontologies, and we are *not* attempting to make original contributions in this area

5.3.11. Definition. [Consistent Result] Let WM and WM_n be working memories, such that WM is $\mathcal{T}$-consistent, and α be an action. Suppose that there is a transition of the form

$$\mathsf{WM} \overset{\alpha}{\twoheadrightarrow} \hat{\mathsf{WM}}$$

and $\hat{\mathsf{WM}}$ is not $\mathcal{T}$-consistent. Let M be the set of all maximal subsets of $\hat{\mathsf{WM}}$ that contain $\hat{\mathsf{WM}} \setminus \mathsf{WM}$ and are $\mathcal{T}$-consistent. We define the $\mathcal{T}$-**consistent result** of applying α to WM as

$$\hat{\mathsf{WM}}^{cons} = \bigcap_{\mathsf{WM}^{max} \in M} \mathsf{WM}^{max}$$

□

5.3.12. Example. Suppose we execute r_1 in WM_0. We obtain the inconsistent working memory

$$\mathsf{WM}_1 = \{\mathit{takeT}(\text{Laura}, \text{pcr}), \mathit{flu}(\text{Laura}), \mathit{dnaT}(\text{pcr}), \mathit{requested}(\text{Smith}, \text{Laura}, \text{pcr})\}$$

We have two maximal consistent subsets of WM_1

- $\mathsf{WM}'_1 = \{\mathit{takeT}(\text{Laura}, \text{pcr}), \mathit{dnaT}(\text{pcr}), \mathit{requested}(\text{Smith}, \text{Laura}, \text{pcr})\}$
- $\mathsf{WM}'_1 = \{\mathit{takeT}(\text{Laura}, \text{pcr}), \mathit{flu}(\text{Laura}), \mathit{requested}(\text{Smith}, \text{Laura}, \text{pcr})\}$

Thus, the consistent result is:

$$\mathsf{WM}_1^{cons} = \{\mathit{takeT}(\text{Laura}, \text{pcr}), \mathit{requested}(\text{Smith}, \text{Laura}, \text{pcr})\}$$ □

5.3.13. Definition. [Consistent Transition] Let (WM_0, σ) be a $\mathcal{T}$-consistent prestructure and $\alpha_1 \dots \alpha_n$ be a sequence of actions. Suppose that we have the following transitions:

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma) \overset{\alpha_2}{\twoheadrightarrow}, \dots (\mathsf{WM}_{n-1}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

Let WM_i^{cons} $(i = 1 \ldots n)$ be the $\mathcal{T}$-consistent results of applying α_i to $(\mathsf{WM}_{i-1}^{cons}, \sigma)$ (where $\mathsf{WM}_0^{cons} = \mathsf{WM}_0$). We define the corresponding **$\mathcal{T}$-consistent transition** as

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1^{cons}, \sigma) \overset{\alpha_2}{\twoheadrightarrow}, \ldots (\mathsf{WM}_{n-1}^{cons}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n^{cons}, \sigma) \qquad \square$$

5.3.14. Definition. [Intermediate and Cyclic WM] Let (WM_0, σ) be a $\mathcal{T}$-consistent prestructure and suppose r is a rule with actions $\alpha_1 \cdots \alpha_n$ in the head. Suppose there are working memories $\mathsf{WM}_1 \ldots \mathsf{WM}_n$ such that

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma) \overset{\alpha_2}{\twoheadrightarrow} \ldots \overset{\alpha_{n-1}}{\twoheadrightarrow} (\mathsf{WM}_{n-1}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

In this case we say that WM_0 and WM_n are **cycle** working memories, whereas $\mathsf{WM}_1 \ldots \mathsf{WM}_{n-1}$ are **intermediate** working memories.[4] $\square$

Intuitively, cycle working memories are the initial and the final (resulting) working memories, whereas intermediate working memories are the intermediate states produced by the execution of the rule actions.

We say that a transition of the form

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma) \overset{\alpha_2}{\twoheadrightarrow} \ldots \overset{\alpha_{n-1}}{\twoheadrightarrow} (\mathsf{WM}_{n-1}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

is **non-trivial** if $\mathsf{WM}_0 \neq \mathsf{WM}_n$.

The following two definitions formalize the *conflict resolution strategy* for a given rule r. We say that a rule r is eligible for execution in a working memory WM if r's condition holds in WM, and the working memory resulting from applying r is consistent with the ontology. In addition, if r is an `IF-THEN` rule we require that r's action changes WM, and if r is a `FOR` rule we require that r's action is not instantiated twice with the same assignment.

5.3.15. Definition. [Fireable IF-THEN Rule] Let r be a rule of the form:

$$r\texttt{: Forall } \vec{x}\texttt{: if } \phi_r(\vec{x}) \texttt{ then } \psi_r(\vec{x}) \tag{5.3}$$

We say that r is **fireable** in a prestructure (WM_0, σ) iff

1. (WM_0, σ) is $\mathcal{T}$-consistent.
2. $\mathcal{T}, (\mathsf{WM}_0, \sigma) \models \phi_r$
3. There is a non-trivial $\mathcal{T}$-consistent transition of the form

$$(\mathsf{WM}_0, \sigma) \overset{\psi_r(\sigma(\vec{x}))}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

where $\overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow}$ is defined in Definition 5.3.10.

[4] RIF-PRD calls these cycle states because these are the states where cycles of rule applications begin.

In this case we say that r ***causes transition*** from WM_0 to WM_n and denote it as $\mathsf{WM}_0 \overset{r}{\hookrightarrow} \mathsf{WM}_n$. □

5.3.16. Definition. [Fireable FOR Rule] Let r be a rule of the form:

$$r\colon \texttt{For } \vec{x}\colon \phi_r(\vec{x}) \quad \texttt{do } \psi_r(\vec{x}) \tag{5.4}$$

We say that r is **fireable** in a working memory WM_0 iff

- WM_0 is $\mathcal{T}$-consistent.
- there are prestructures $(\mathsf{WM}_0, \sigma_0), (\mathsf{WM}_1, \sigma_0), (\mathsf{WM}_1, \sigma_1) \ldots (\mathsf{WM}_n, \sigma_{n-1})$ such that there are $\mathcal{T}$-consistent transitions of the form

$$\begin{array}{c} (\mathsf{WM}_0, \sigma_0) \overset{\psi_r(\sigma_0(\vec{x})}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma_0) \\ (\mathsf{WM}_1, \sigma_1) \overset{\psi_r(\sigma_1(\vec{x})}{\twoheadrightarrow} (\mathsf{WM}_2, \sigma_1) \\ \vdots \\ (\mathsf{WM}_{n-1}, \sigma_{n-1}) \overset{\psi_r(\sigma_{n-1}(\vec{x})}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma_{n-1}) \end{array} \tag{5.5}$$

  where the following conditions hold:

  1. *Looping*: $\mathcal{T}, (\mathsf{WM}_i, \sigma_i) \models \phi_r$ for each cycle working memory, WM_i $(0 \leq i \leq n-1)$
  2. *No repetitions*: For each pair of prestructures $(\mathsf{WM}_i, \sigma_i), (\mathsf{WM}_j, \sigma_j)$ $(0 \leq i < j \leq n-1)$, we have that $\sigma_i \neq \sigma_j$
  3. *Termination*: There is no assignment σ such that it produces a $\mathcal{T}$-consistent transition from WM_n, and (WM_n, σ) satisfies r's condition.

In this case we say that r ***causes transition*** from WM_0 to WM_n and denote it as $\mathsf{WM}_0 \overset{r}{\hookrightarrow} \mathsf{WM}_n$. □

Condition 3 above says that rules are no longer fired once the system reaches a fixpoint. This guarantees that a PS does not have trivial infinite runs.

Recall that a PS applies rules in three steps: (1) pattern matching, (2) conflict resolution, (3) rule execution, and then it loops back to (1). So far we have described only the steps (1) and (3). The next series of definitions describes Step (2) and show how looping is modeled in the semantics. This semantics does not depend on any particular conflict resolution strategy so, for concreteness, in Step (2) we will simply randomly choose a fireable rule from the conflict resolution set.[5] Some other works [9, 25] use the same strategy.

[5]Recall that the conflict resolution set contains all the rules that can be fired on a given working memory.

5.3.17. DEFINITION. [Consistent Transition Graph] The ***transition graph***, $\mathcal{T}_{\mathsf{PS}}$, of a production system is a directed labeled graph, whose set of nodes is the set of all working memories. There is an edge between two nodes WM and WM', labeled with α, σ for some action α and variable assignment σ, iff $(\mathsf{WM}, \sigma) \overset{\alpha}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$. We will use $\mathcal{P}_{\mathsf{WM}}$ to denote the set of all paths (sequences of WMs) in the graph $\mathcal{T}_{\mathsf{PS}}$ starting at WM. □

5.3.18. DEFINITION. [Split] Let $\pi = \mathsf{WM}_0 \dots \mathsf{WM}_n$ be a path in $\mathcal{P}_{\mathsf{WM}_0}$. A *split* of π is a pair of subpaths, π_1 and π_2, such that $\pi_1 = \mathsf{WM}_0 \dots \mathsf{WM}_i$ and $\pi_2 = \mathsf{WM}_i \dots \mathsf{WM}_n$ for some i $(1 \leq i \leq n)$. In this case, we write $\pi = \pi_1 \circ \pi_2$. □

5.3.19. DEFINITION. [Run] A path π in $\mathcal{P}_{\mathsf{WM}_0}$ is a ***run*** $\mathcal{R}$ for a production system PS iff there are splits $\pi = \pi_1 \circ \dots \circ \pi_n$ and rules $r_1 \dots r_n$ such that for each $i = 1 \dots n$, $WM_{i,start} \overset{r_i}{\hookrightarrow} WM_{i,end}$, where $WM_{i,start}$ is the first element in π_i and $WM_{i,end}$ is its last. Note that this implies that every π_i is a $\mathcal{T}$-consistent transition (see Definition 5.3.13). □

We will refer to the ith **cycle** working memory in a run as WM_i.

5.4 Production Systems in $\mathcal{TR}^{PAD}$

In this section we present the reduction of production systems augmented with Datalog-rewritable ontologies to $\mathcal{TR}^{PAD}$. Given an alphabet $\mathcal{L}_{\mathsf{PS}}$ for a production system PS, the corresponding language $\mathcal{L}_{\mathcal{TR}}$ of the target $\mathcal{TR}^{PAD}$ formulation will consist of symbols for rule labels, constants, and predicates. In addition, $\mathcal{L}_{\mathcal{TR}}$ has the following symbols:

- the *pdas* *add_used* and *clean_used*, and for every predicate $p \in \mathcal{L}_{\mathsf{PS}}$, *Ins_p*, and *del_p*;
- the compound action *act*;
- the defined fluent *inconsistent*, and for every rule label r the defined fluent *fireable_r*;
- the fluents *inertial* and *used*.

Intuitively, the *pdas* *Ins_p*, and *del_p* above represent the actions ***assert*** and ***retract***. The *pdas* *add_used* and *clean_used*, and the fluent *used*, are used to keep track of the assignments that have already been used to instantiate a `FOR`-production rule. The compound action *act* represents a generic production rule. The defined fluent *fireable_r* is true if the condition of the rule r holds and the action produces no inconsistencies. The defined fluent *inconsistent* is true, if there

is an inconsistency in the state. The fluent *inertial* is used to distinguish inertial from non-inertial fluents.

Let $\psi = \alpha_1 \dots \alpha_n$ be a sequence of atomic actions. We use $\widehat{\psi}$ to denote the $\mathcal{TR}$ -serial conjunction $\widehat{\psi} = \widehat{\alpha_1} \otimes \dots \otimes \widehat{\alpha_n}$ where

$$\widehat{\alpha_i} = \begin{cases} \mathit{Ins\_}p(t_1 \dots t_n) & \text{if } \alpha_i = \boldsymbol{assert}(p(t_1 \dots t_n)) \\ \mathit{del\_}p(t_1 \dots t_n) & \text{if } \alpha_i = \boldsymbol{retract}(p(t_1 \dots t_n)) \end{cases}$$

Let $\phi = f_1 \wedge \dots \wedge f_n \wedge l_1 \dots l_m$ be a conjunction of atoms (f_i) and negative literals (l_j). Then let $\widehat{\phi}$ denote the $\mathcal{TR}$ -serial conjunction $\widehat{\phi} = f_1 \wedge \dots \wedge f_m \wedge \sim l_1 \wedge \dots \wedge \sim l_m$, where

$$\sim l_j = \begin{cases} \mathbf{not}\, f(\vec{c}) & \text{if } l_j = \neg f(\vec{c}) \wedge f \in \mathcal{P}_{\mathrm{PS}} \\ \mathbf{neg}\, f(\vec{c}) & \text{if } l_j = \mathbf{neg}\, f(\vec{c}) \wedge f \in \mathcal{P}_{\mathrm{DL}} \end{cases}$$

In the following, let $\mathsf{PS} = (\mathcal{T}, \mathsf{L}, R)$ be a production system. For simplicity we assume that conditions in production rules are conjunction of fluent literals. In addition, we assume we have an ***initial working memory***, WM_0, that represents the knowledge we have about the initial state of the system. A production system coupled with a working memory is called a ***configuration***.

The reduction, Λ_{PS}, of a configuration (PS, WM_0) to $\mathcal{TR}^{PAD}$ is a $\mathcal{TR}^{PAD}$ knowledge-base $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ composed of the following PADs ($\mathcal{E}$), rules for defined fluents ($\mathbf{P}$) and premises ($\mathcal{S}$). From now on we assume that the ontology $\mathcal{T}$ is *Datalog-expressible* (e.g., $\mathcal{DL}^+$—see Section 5.1).

1. **State identifiers**: There is an *initial* state identifier $\mathbf{d}_0$. The rest of the state identifiers are indexed by sequences of actions that need to be applied to $\mathbf{d}_0$ in order to reach those other states. That is, they have the form $\mathbf{d}_{0,\alpha_1,\dots,\alpha_n}$, where $n \geq 0$ and each α_i is a ground instance of a $\mathcal{TR}^{PAD}$'s partially defined action *add_used*, *clean_used*, *Ins_p*, or *del_p*, for some p.

2. **Ontology $\mathcal{T}$: $\mathbf{P}$** contains all the rules from the Datalog rendering of $\mathcal{T}$.

3. **Initial Database:** The premises below characterize the content of the initial working memory WM_0.

   - For each atomic literal $p(t_1, \dots, t_n)$ in WM_0

     $\mathbf{d}_0 \rhd p(t_1, \dots, t_n) \in \mathcal{S}$
     $\mathbf{d}_0 \rhd \mathit{inertial}(p(t_1, \dots, t_n)) \in \mathcal{S}$[6]

4. **Frame Axioms:** The following frame axioms encode the laws of inertia. In addition, they take care of the actual "removal" of $\mathcal{L}_{\mathsf{PS}}$ atoms from the working memory, and the cleaning of the used assignments.

[6] We could have written this as $\mathit{inertial}(p, t_1, \dots, t_n)$ to avoid the appearance of being second order or that the use of function symbols here is essential.

- For each action α_q involves assertion or retraction of an atom with predicate symbol q, where $p \neq q$:

$$\left\{ \begin{array}{r} (\mathit{inertial}(p(\vec{X})) \wedge p(\vec{X})) \otimes \alpha_q(\vec{Y}) \otimes \mathbf{not}\ \mathit{inconsistent} \rightarrow \\ \alpha(\vec{Y}) \otimes (p(\vec{X}) \wedge \mathit{inertial}(p(\vec{X}))) \end{array} \right\} \in \mathcal{E}$$

- For each action α_p involves assertion or retraction of an atom with predicate symbol p:

$$\left\{ \begin{array}{r} (\mathit{inertial}(p(\vec{X})) \wedge p(\vec{X}) \wedge \vec{X} \neq \vec{Y}) \otimes \alpha_p(\vec{Y}) \otimes \mathbf{not}\ \mathit{inconsistent} \rightarrow \\ \alpha(\vec{Y}) \otimes (p(\vec{X}) \wedge \mathit{inertial}(p(\vec{X}))) \end{array} \right\} \in \mathcal{E}$$

- For each predicate p

$$\left\{ \begin{array}{r} (\mathit{inertial}(p(\vec{X})) \wedge p(\vec{X})) \otimes \mathit{add\_used}(\vec{Y}) \rightarrow \\ \mathit{add\_used}(\vec{Y}) \otimes (p(\vec{X}) \wedge \mathit{inertial}(p(\vec{X}))) \\ (\mathit{inertial}(p(\vec{X})) \wedge p(\vec{X}) \wedge p \neq \mathit{used}) \otimes \mathit{clean\_used} \rightarrow \\ \mathit{clean\_used} \otimes (p(\vec{X}) \wedge \mathit{inertial}(p(\vec{X}))) \end{array} \right\} \in \mathcal{E}$$

5. **Actions:** The following rules encode the actions ***assert*** and ***retract*** in $\mathcal{TR}^{PAD}$:

   - *Insert:* For each predicate $p \in \mathcal{L}_{\mathrm{PS}}$ (whether in $\mathcal{P}_{\mathrm{DL}}$ or $\mathcal{P}_{\mathrm{PS}}$):

$$\left\{ \begin{array}{r} \mathit{Ins\_}p(t_1, \ldots, t_n) \rightarrow \mathit{Ins\_}p(t_1, \ldots, t_n) \otimes \\ (p(t_1, \ldots, t_n) \wedge \mathit{inertial}(p(t_1, \ldots, t_n))) \end{array} \right\} \in \mathcal{E}$$

   - *Retract:* For each predicate $p \in \mathcal{P}_{\mathrm{DL}}$,

$$\left\{ \begin{array}{r} \mathit{del\_}p(t_1, \ldots, t_n) \rightarrow \mathit{del\_}p(t_1, \ldots, t_n) \otimes \\ (\mathbf{neg}\, p(t_1 \ldots t_n) \wedge\ \mathit{inertial}(\mathbf{neg}\, p(t_1 \ldots t_n))) \end{array} \right\} \in \mathcal{E}$$

     Recall that the effect of the *pda* $\mathit{del\_}p$ for PS atoms is given by the interaction with the frame axioms. For instance, if applying $\mathit{del}_{dnaT}(pcr)$ in $\mathbf{d}_1$ results in a state $\mathbf{d}_2$, it holds that $\mathbf{d}_2$ is equal to $\mathbf{d}_1$ except for $\mathit{dnaT}(pcr)$, which is not carried to $\mathbf{d}_2$ by the frame axioms. This is equivalent to remove $\mathit{dnaT}(pcr)$ from $\mathbf{d}_2$.

6. **Production rules:** The following rules encode the production rules.

   - For each `IF-THEN`-rule of the form "r: `Forall` $\vec{x}$: `if` $\phi_r(\vec{x})$ `then` $\psi_r(\vec{x})$"

$$r \leftarrow \mathit{fireable\_}r(\vec{X}) \otimes \widehat{\psi}_r(\vec{X}) \ \in \mathbf{P}$$

- For each FOR-rule of the form "r: For $\vec{x}$: $\phi_r(\vec{x})$ do $\psi_r(\vec{x})$"

$$\left.\begin{array}{l} r \leftarrow \mathit{fireable\_r}(\vec{X}) \otimes \widehat{\psi}_i(\vec{X}) \otimes \mathit{add\_used}(\vec{X}) \otimes loop\_r \\ loop\_r \leftarrow r \\ loop\_r \leftarrow (\mathbf{not}\, \exists \vec{X} : \mathit{fireable\_r}(\vec{X})) \otimes \mathit{clean\_used} \end{array}\right\} \in \mathbf{P}$$

where $\mathbf{not}\, \exists \vec{X} : \mathit{fireable\_r}(\vec{X})$ above is a shorthand for $\mathbf{not}\, p'$ such that p' is a new predicate defined as $p' \leftarrow \mathit{fireable\_r}(\vec{X})$.

7. **Auxiliary Actions and Premises:**

- *Run-Premises*: For each pda α and a sequence ξ of *Ins*/*del*/*add_used*/*clean_used* actions, the set of premises $\mathcal{S}$ contains the following run-premise:

$$\mathbf{d}_\xi \overset{a}{\rightsquigarrow} \mathbf{d}_{\xi,a}$$

For example, $\mathbf{d}_{0,\mathit{Ins\_p(c)}} \overset{\mathit{Ins}_q(d)}{\rightsquigarrow} \mathbf{d}_{0,\mathit{Ins\_p(c)},\mathit{Ins}_q(d)}$.

- *Inconsistency*: For each predicate $p \in \mathcal{L}_{\mathrm{PS}}$, $\mathbf{P}$ contains a rule of the form:

$$\mathit{inconsistent} \leftarrow p(\vec{X}),\ \mathbf{neg}\, p(\vec{X})$$

- *Adding used assignments:*

$$\left\{\ \mathit{add\_used}(\vec{Y}) \rightarrow \mathit{add\_used}(\vec{Y}) \otimes \mathit{used}(\vec{X})\ \right\} \in \mathcal{E}$$

- *Fireability*:
  - If r is an IF-THEN rule

$$\left\{\begin{array}{l} \mathit{fireable\_r}(\vec{X}) \leftarrow \widehat{\phi}_r(\vec{X}) \wedge \bigvee_{p \in \mathcal{P}} inertial(p(Y)) \wedge \\ (\Diamond\widehat{\psi}_r(\vec{X}) \otimes \mathbf{not}\ \mathit{inconsistent} \wedge \mathbf{not}\, inertial(p(Y))) \end{array}\right\} \in \mathbf{P}$$

  - If r is an FOR rule

$$\left\{\begin{array}{l} \mathit{fireable\_r}(\vec{X}) \leftarrow \widehat{\phi}_r(\vec{X}) \wedge \mathbf{not}\ \mathit{used}(\vec{X})) \wedge (\Diamond\widehat{\psi}_r(\vec{X}) \otimes \\ \mathbf{not}\ \mathit{inconsistent}) \end{array}\right\} \in \mathbf{P}$$

Observe that in the definition of *fireable_r* for IF-THEN rules, we also require that the effect of the action must produce a change, in particular in the rule above we require that it must retract some inertial fact. We omit the analogous definition requiring that the change consists of inserting a new inertial fact.

- *Random choice of action*: Suppose $\{r_1 \dots r_n\} = \mathsf{L}$

$$\left.\begin{array}{l} \mathit{act} \leftarrow r_1 \\ \vdots \\ \mathit{act} \leftarrow r_n \end{array}\right\} \in \mathbf{P}$$

To run k rules of the production system we use the transaction:

$$?-_{(\mathbf{d}_0)} \underbrace{\mathit{act} \otimes \cdots \otimes \mathit{act}}_{k}$$

5.4.1. THEOREM (SOUNDNESS). *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}^{PAD}$ embedding of a PS configuration. Suppose*

$$\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d}_0 \dots \mathbf{d}_k \models \underbrace{\mathit{act} \otimes \cdots \otimes \mathit{act}}_{m}$$

Then there are working memories $\mathsf{WM}_1 \dots \mathsf{WM}_m$, and rules $r_1 \dots r_m$ such that

$$\begin{array}{c} \mathsf{WM}_0 \overset{r_1}{\hookrightarrow} \mathsf{WM}_1 \\ \vdots \\ \mathsf{WM}_{m-1} \overset{r_m}{\hookrightarrow} \mathsf{WM}_m \end{array}$$

16. PROOF. *See appendix G.* □

5.5 Summary of the Contributions

In this chapter we proposed a new semantics for the combination of production systems with *arbitrary* DL ontologies. Unlike previous approaches [72, 25, 22, 9, 55, 85], the semantics presented here supports extensions, like the *FOR*-loops or *while*-loops, that are not included in RIF-PRD, but are found in commercial production systems such as IBM's *JRules*[49]. In addition, our approach can handle inconsistencies produced by the interaction of rule actions and the ontology.

We also defined a sound embedding of such semantics, restricted to rule-based DL ontologies, into $\mathcal{TR}^{PAD}$. This reduction gives a declarative semantics to the combination, and is considerably simpler and compact that other approaches, including [72, 55, 85, 22, 52].

Chapter 6

Conclusions

Dad: What are you watching, Mafalda?
Mafalda: The fight.
Dad: But... it's a soap opera! What fight are you talking about?
Mafalda: The writer's fight. It's fascinating seeing the writer's struggle to escape the clutches of intelligence.

Joaquín Salvador Lavado (Quino)
Mafalda

In this thesis we extended Transaction Logic and made it suitable for reasoning about partially defined actions. We illustrated the power of the language for complex reasoning tasks involving actions and gave a sound and complete proof theory for that formalism. We showed that $\mathcal{TR}^{PAD}$ has a great deal of sophistication in action composition, enabling hypothetical, recursive, and nondeterministic actions. In particular, compared with other actions languages like [39, 35, 8, 17, 38, 16], $\mathcal{TR}^{PAD}$ supports more general ways of describing actions and can be more selective in when and whether the fluents are subject to the laws of inertia. Moreover, we proved that $\mathcal{TR}^{PAD}$ generalizes Horn-$\mathcal{TR}^{-}$. This implies that the frame axioms proposed in this thesis *behave as expected* in the relational case. That is, they correctly model the inertia laws.

We also showed that, when all partially defined actions are definite, reasoning in $\mathcal{TR}^{PAD}$ can be done by a reduction to ordinary logic programming. This last contribution provides an easy way to implement and experiment with the formalism, although a better implementation should be using the proof theory directly, similarly to the implementation of the serial-Horn subset of $\mathcal{TR}$ in

FLORA-2 [50].

This work continues the line of research started in [10], which, however, was targeting a different fragment of $\mathcal{TR}$. It did not provide a complete proof theory or a reduction to logic programming. It also did not consider premise statements and thus could not be used for reasoning about partially defined actions without further extensions.

To boost the non-monotonic capabilities of $\mathcal{TR}^{PAD}$, we extended $\mathcal{TR}^{PAD}$ with *default* negation (a.k.a. negation as failure). Default negation allows a logic system to conclude the negation of any atom that the system unsuccessfully finishes exploring all possible proofs. That is, if we fail to prove that a fact is true, by *default*, we assume it is false. Default negation has become a central ingredient in the design of logic programming languages, databases [77, 37], truth maintenance system [47], etc. We showed how default negation can be used to simplify the axioms of inertia, and lift a restriction we have to impose on $\mathcal{TR}^{PAD}$ specifications without default negation.

We also explored and compared two expressive formalisms for reasoning about actions: $\mathcal{TR}^{PAD}$ (without default negation) and $\mathcal{L}_1$. We showed that these formalisms have different capabilities and neither subsumes the other. Nevertheless, we established that a large subset of $\mathcal{L}_1$ can be soundly represented in $\mathcal{TR}^{PAD}$ and that the LP reduction of that subset is logically equivalent to the corresponding subset of $\mathcal{TR}^{PAD}$. We also compared the two logics in the domain of action planning and showed that $\mathcal{TR}^{PAD}$ has a significant advantage when it comes to planning under constraints. Finally, we briefly compared $\mathcal{TR}^{PAD}$ with other action languages, Situation Calculus and Golog, Fluent Calculus and Flux, and Event Calculus, $\mathcal{ALM}$ and $\mathcal{C}$. Uniquely among these formalisms $\mathcal{TR}^{PAD}$ supports powerful ways of action composition.

As an application, we studied how to use $\mathcal{TR}^{PAD}$ to combine *production systems* extended with lopping rules and ontologies. Traditional PSs consist of a set of *production rules* that rely on forward chaining reasoning to update the underlying database, called *working memory*. Traditionally, PSs have had only operational semantics, where satisfaction of rule conditions is checked using pattern matching, and rule actions produce assertion and deletions of facts from the working memory. Since there is no semantics available for the combination of PS with looping rules and ontologies, first we proposed a new—operational—semantics such combination. This semantics continues the line of research of [72], which, however, was targeting a more restricted and not standard definition of production systems. For instance, [72] did not consider looping-rules and it could not handle inconsistencies produced by the system. Finally, we presented a sound embedding of such semantics into $\mathcal{TR}^{PAD}$, providing in this a declarative semantics to PSs augmented with *rule-based* ontologies.

Finding decidable fragments of $\mathcal{TR}^{PAD}$ for which termination of reasoning is guaranteed will be the focus of our future work.

The contributions provided in this thesis have a strong impact in the subfield of Artificial Intelligence dedicated to reasoning about actions. We expect these results to find applications in intelligent agent systems, semantic web services, question answering systems, and other areas.

Appendix A

Soundness and Completeness Proofs of the Inference System $\mathcal{F}$

In this Appendix we prove soundness and completeness of the inference system $\mathcal{F}$ developed in Section 3.3. For simplicity we present a ground version of the inference system. Lifting to the non-ground case is done in a standard way (cf. [11]).

Soundness of $\mathcal{F}$

This appendix contains proofs of soundness of the inference system $\mathcal{F}$ developed in Section 3.3. We assume that all transactions are serial goals, that the transaction base is a set of serial Horn rules and PADs, and that the set of premises are *state*- and *run*-premises defined in Definition 3.1.1. For convenient reference we reproduce the axioms and inference rules of system $\mathcal{F}$ below.

A.0.1. Definition. [Inference System $\mathcal{F}$] Let $\mathbf{P}$ be a transaction base and $\mathcal{S}$ a set of premises. The inference system $\mathcal{F}$ consists of the following axioms and inference rules, where $\mathbf{d}$, $\mathbf{d}_1$, $\mathbf{d}_2$, and so on, denote database states.
Axioms:

1. *No-op:* $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash ()$

Inference rules: In the rules below, a, and α are literals, and ϕ, ψ, and b_i $(i = 1, 2, 3, 4)$ are serial goals.

1. *A subset of Horn inference rules from [11, 12]:*

(a) *Applying transaction definitions:*

$$\frac{a \leftarrow \phi \in \mathbf{P} \qquad \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi \otimes \psi}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash a \otimes \psi}$$

(b) *Hypothetical operations:*

$$\frac{\mathbf{P}, \mathcal{S}, \mathbf{d}, \mathbf{d}'_1, ..., \mathbf{d}'_n \vdash \beta \qquad \mathbf{P}, \mathcal{S}, \mathbf{d}, \mathbf{d}_1, ..., \mathbf{d}_m \vdash \gamma}{\mathbf{P}, \mathbf{d}, \mathbf{d}_1, ..., \mathbf{d}_m \vdash \Diamond\beta \otimes \gamma}$$

2. *Premise rules:* Suppose $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ or $\mathbf{d} \rhd f$ is a premise in $\mathcal{S}$. Then

$$\frac{\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha} \qquad \frac{\mathbf{d} \rhd f \in \mathcal{S}}{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash f}$$

3. *Forward Projection:* Suppose α is a partially defined ground action term. Then

$$\frac{\begin{array}{c} b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \in \mathbf{P} \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_2 \vdash b_2 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha \end{array}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_3 \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_2 \vdash b_4}$$

4. *Sequencing:*

$$\frac{\begin{array}{c} \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \vdash \psi \\ \text{where } 1 \leq i \leq n \end{array}}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi \otimes \psi}$$

5. *Decomposition:* Suppose ϕ and ψ are serial conjunctions of literals and hypotheticals. Then

$$\frac{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \phi \otimes \psi}{\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \phi \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \psi}$$

A.0.2. Theorem (Soundness of $\mathcal{F}$). *If* $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$ *then* $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$

To prove Theorem A.0.2, it is enough to show that every axiom and inference rule of system $\mathcal{F}$ is sound. Soundness of the axioms and of the Horn inference rules in $\mathcal{F}$ follows from Theorem A.2 in [11] after simple adjustments for the existence of PADs (instead of elementary updates defined by transition oracles) in $\mathbf{P}$. Lemma A.0.3 establishes the soundness of the remaining inference rules.

A.0.3. LEMMA (INFERENCE RULES). *1. Suppose that the premise* $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ *is in* $\mathcal{S}$. *Then* $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \alpha$.

2. Suppose that $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ *is a PAD in* $\mathbf{P}$. *If*

$$\begin{array}{c} \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_1 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models b_2 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \alpha \end{array}$$

then $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_3$ *and* $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models b_4$.

3. Let ϕ *and* ψ *be serial conjunctions of atoms. If, for some* $1 \leq i \leq n$,

$$\begin{array}{c} \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \models \phi \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models \psi \end{array}$$

then $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi \otimes \psi$.

4. Let ϕ *and* ψ *be serial conjunctions of fluents.*

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \phi \otimes \psi$$

then $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \phi$ *and* $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \psi$.

Proof. Claim 1 follows immediately, since for every model $\mathbf{M}$ of $(\mathbf{P}, \mathcal{S})$ and premise $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ in $\mathcal{S}$, it follows by Definition 3.1.3 that $\mathbf{M}, \langle \mathbf{d}_1\mathbf{d}_2 \rangle \models \alpha$. Therefore, by definition of entailment in $\mathcal{TR}^{PAD}$, we have $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \phi$.

To prove Claim 2, suppose that the *PAD* $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ is in $\mathbf{P}$ and the entailments in the statement of the claim hold. Let $\mathbf{M}$ be a model of $(\mathbf{P}, \mathcal{S})$ such that

- $\mathbf{M}, \langle \mathbf{d}_1 \rangle \models b_1$
- $\mathbf{M}, \langle \mathbf{d}_2 \rangle \models b_2$
- $\mathbf{M}, \langle \mathbf{d}_1\mathbf{d}_2 \rangle \models \alpha$

Since $\mathbf{M}$ is also a model of $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ and the premise of that implication holds in $\mathbf{M}$, by assumption, it follows that

$$\mathbf{M}, \langle \mathbf{d}_1\mathbf{d}_2 \rangle \models b_3 \otimes \alpha \otimes b_4$$

must hold. Since b_3 and b_4 are conjunctions of fluents and $\mathbf{M}, \langle \mathbf{d}_1\mathbf{d}_2 \rangle \models \alpha$, the definition of satisfaction in TR implies

$$\begin{array}{c} \mathbf{M}, \langle \mathbf{d}_1 \rangle \models b_3 \\ \mathbf{M}, \langle \mathbf{d}_2 \rangle \models b_4 \end{array}$$

Since $\mathbf{M}$ is an arbitrary model of $(\mathbf{P}, \mathcal{S})$, we obtain $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_3$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models b_4$. Claims 3 and 4 follows directly from the definition of serial conjunction $\otimes$. □

Completeness of $\mathcal{F}$

To prove completeness of the inference system $\mathcal{F}$ of Section 3.3, we construct a canonical Herbrand model of the $\mathcal{TR}^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. As before, $\mathcal{U}$ will be denoting the Herbrand universe of the logic language and $\mathcal{B}$ its Herbrand domain. A *classical* Herbrand structure is a subset of $\mathcal{B}$. Recall that we assume that all transactions are serial goals, that the transaction base is a set of serial Horn rules and PADs, and that premise statements are as in Definitions 3.1.1.

A.0.4. Definition. [Canonical Model] The canonical model of a transaction base $\mathbf{P}$ and a set of premises $\mathcal{S}$ is a Herbrand path structure $\mathbf{M}_{\mathbf{P},\mathbf{S}}$, such that

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle) = \{b \in \mathcal{B} \mid \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash b\}$$

for any sequence of states $\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle$.

To justify its name, we need to show that canonical models are indeed models. The next lemma shows that $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ is a path structure. That it is a model follows from Theorem A.0.8, below. Recall that in $\mathcal{TR}^{PAD}$ we use *pda*s instead of elementary updates, so elementary updates and transition oracles of TR play no role in our construction.

A.0.5. Lemma. *Let* $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ *be the canonical model of* $\mathbf{P}, \mathcal{S}$*. Then,*

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d} \rangle) \models^c l \quad \textit{for every literal} \quad l \in \mathbf{d}$$

Proof. If $l \in \mathbf{d}$ then, by the No-op axiom and inference rule 4, $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash l$. By construction of $\mathbf{M}_{\mathbf{P},\mathbf{S}}$, $l \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d} \rangle)$, which implies $\mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d} \rangle) \models^c l$. □

The following lemma is a key property of canonical models.

A.0.6. Lemma. *If* b *is a ground atomic formula, then*

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models b \quad \textit{iff} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash b$$

Proof. By the definition of satisfaction in path structures,
$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models b$ if and only if $b \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle)$. By Definition A.0.4, $b \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle)$ if and only if $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash b$. □

We now generalize the above result to serial conjunctions.

A.0.7. Theorem. *Let* ϕ *be a ground serial conjunction. Then*

$$\textit{if} \quad \mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \phi \quad \textit{then} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$$

Proof. Let ϕ have the form $b_1 \otimes \cdots \otimes b_k$, where $k \geq 0$ and each b_i is a ground atomic formula. Our proof is by induction on k. In the base case, $k = 0$ and ϕ is the empty clause (i.e., ()). If the expression $\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models ()$ is true, then $n = 1$, since the empty clause is true only on paths of length one. But the sequent $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models ()$ is an axiom, and the claim follows.

For the inductive case, assume the claim is true for all k such that $0 \leq k < m$. We show that it is true for $k = m$. Below we use ϕ_m to denote $b_1 \otimes \cdots \otimes b_{m-1}$.

$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \phi_m \otimes b_m$	given
$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_i \rangle \models \phi_m$ and $\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_i \ldots \mathbf{d}_n \rangle \models b_m$	for some i, by Definition 3.1.3
$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models b_m$	by I. Hypothesis
$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \vdash b_m$	by Lemma A.0.6
$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m \otimes b_m$	by Inferece Rule 4 $\square$

A.0.8. THEOREM. $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ *is a model of* $\mathbf{P}$ *and* $\mathcal{S}$

Proof. Since the proof that $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ satisfies the Horn rules in $\mathbf{P}$ is very similar to the proof of Theorem B.9 in [11], we only show that $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ satisfies the PADs in $\mathbf{P}$ and the premises in $\mathcal{S}$.

Let $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ be a *PAD* in $\mathbf{P}$ and $\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle$ a path. If

$$\begin{aligned} &\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \alpha \\ &\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_1, \\ &\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_2 \end{aligned}$$

Then by Theorem A.0.7

$$\begin{aligned} &\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \alpha \\ &\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1 \\ &\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash b_2 \end{aligned} \tag{A.1}$$

and by inference rule 3 we get

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_3 \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash b_4 \tag{A.2}$$

Since b_1 and b_2 are classical conjunctions of fluents, by inference rule 5 we conclude $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1'$ for every atomic conjunct b_1' of b_1 and $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash b_2'$ for every atomic conjunct b_2' of b_2. From this and Lemma A.0.6 it now follows that

$$\begin{aligned} &\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_1' \\ &\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_2' \end{aligned}$$

and thus, by the definition of $\wedge$, that

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_3$$
$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_4$$

Finally, we observe that $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ is a model of $\mathcal{S}$ because every premise in $\mathcal{S}$ gives rise to a sequent in $\mathcal{F}$, by inference rule 2. Hence, for every premise $\mathbf{d} \rhd f$ or $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2$ in $\mathcal{S}$ we derive the corresponding sequent $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash f$ or $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha$. Therefore $f \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d} \rangle)$ and $\alpha \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1\mathbf{d}_2 \rangle)$. □

A.0.9. Corollary (Completeness of $\mathcal{F}$). *Let ϕ be a ground serial conjunction. Then*

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi \quad \textit{implies} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$$

Proof. Suppose that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$. Then:

$\mathbf{M}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \phi$	for every model, $\mathbf{M}$, of $\mathbf{P}$ and $\mathcal{S}$
$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \phi$	since $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ is a model of $(\mathbf{P}, \mathcal{S})$, by Theorem A.0.8
$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$	by Theorem A.0.7 □

Appendix B

Proof of the Reduction of Horn-$\mathcal{TR}^-$ to $\mathcal{TR}^{PAD}$

This appendix proves that $\mathcal{TR}^{PAD}$ generalizes Horn-$\mathcal{TR}^-$. This implies, as a corollary, that the frame axioms in the action theory behaves as expected in the relational case. That is, they can model the inertia laws lying behind the relational transition oracles. For convenient reference we will repeat the main definitions.

B.0.10. Definition. [Relational specifications for serial-Horn programs] A $\mathcal{TR}^{PAD}$ specification $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ is a ***relational specification*** of a serial-Horn program $(\mathbf{P}, \mathbf{D})$ if and only if:

Initial State for every ground base `fluent`-literal f such that $f \in \mathbf{D}$, $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ has these premise formulas:

$$\mathbf{d}_0 \rhd f$$
$$\mathbf{d}_0 \rhd \mathit{inertial}(f)$$

Transaction Base

$$\begin{aligned}\mathbf{Q} = \ & \mathbf{P} \\ & \cup \{\mathit{insert}(f) \rightarrow \mathit{insert}(f) \otimes f \mid \\ & \qquad \text{for every ground base } \texttt{fluent}\text{-literal } f\} \\ & \cup \{\mathit{delete}(f) \rightarrow \mathit{delete}(f) \otimes \mathbf{neg}\, f \mid \\ & \qquad \text{for every ground base } \texttt{fluent}\text{-literal } f\}\end{aligned}$$

Plus the *action theory* of $\mathbf{Q}$.

Transitions for every elementary action α, and sequence r of elementary action, $\mathcal{S}$ contains run-premises of the form:

$$\mathbf{d}_{0,r} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_{0,r,\alpha}$$

In addition, we assume that every ground base fluent is inertial in every state.

B.0.11. Definition. [Correspondence of states] Let $(\mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Given a state identifier $\mathbf{d}$ in $\mathcal{L}_{\mathcal{TR}^{PAD}}$, let $\mathbf{D}(\mathbf{d})$ denote the following set of database fluents in the language $\mathcal{L}_{\mathcal{TR}^-}$ of Transaction Logic:

$$\mathbf{D}(\mathbf{d}) \;=\; \{f \mid f \text{ is a ground } \textit{base } \texttt{fluent}\text{-term such that} \mathbf{P}, \mathcal{S}, \mathbf{d} \models f\}$$

□

B.0.12. Proposition (State consistency and completeness). *Let* $(\mathbf{P}, \mathbf{D})$ *be a Horn-*$\mathcal{TR}^-$ *program and* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}}$ *be a relational specification for* $(\mathbf{P}, \mathbf{D})$ *(see Definition 3.4.6). Let* α *be an action, and* $\mathbf{d}_1, \mathbf{d}_2$ *be state identifiers such that* $\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \alpha$. *If* $\mathbf{D}(\mathbf{d}_1)$ *is consistent then so is* $\mathbf{D}(\mathbf{d}_n)$. *If, in addition,* $\mathbf{D}(\mathbf{d}_1)$ *is complete then so is* $\mathbf{D}(\mathbf{d}_n)$.

Proof. The proof relies on the fact that $\mathcal{TR}^{PAD}$ has a sound and complete proof theory and proceeds by induction on the number N of steps needed to derive

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \alpha \tag{B.1}$$

Observe that since $insert(f)$ and $delete(f)$ have neither a precondition nor a post-condition, we can disregard the following frame axioms

- Forward and Backward Disablement
- Backward Projection
- Causality

Moreover, since the only state-premises we have use $\mathbf{d}_0$, we can also disregard the Backward Inertia frame axiom.

Base case: $N = 1$. In that case, (B.1) can be derived only by the run-premise inference rule. Therefore (B.1) must have the form

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha \tag{B.2}$$

From the previous facts we know that α is either of the form $insert(f)$ or of the form $delete(f)$. For concreteness assume that $\alpha = insert(f)$. From the

definition of $\mathcal{S}$ we know that for every elementary action α, and sequence r of elementary actions, $\mathcal{S}$ contains run-premises of the form:

$$\mathbf{d}_{0,r} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_{0,r,\alpha}$$

Thus, it follows that $\mathbf{d}_2 = \mathbf{d}_{1,insert(f)}$, where the subindex 1 is a sequence of elementary actions. From definition of satisfaction in path structures we know that for every model $\mathcal{M}$ of $\mathbf{Q}, \mathcal{S}$:

$$\mathcal{M}, \langle \mathbf{d}_1 \mathbf{d}_2 \rangle \vdash insert(f)$$

Since there are no state premises of the form $\mathbf{d}_2 \rhd g$ for any fluent g, it follows that the base fluent facts that hold in $\mathcal{M}(\langle \mathbf{d}_2 \rangle)$ are induced by the PADs that exist in the transaction base. Thus, since $\mathcal{M}(\langle \mathbf{d}_0 \rangle)$ is complete and consistent, from the Forward Inertia frame axioms we can conclude that if $\mathcal{M}, \langle \mathbf{d}_1 \rangle \models g$ then $\mathcal{M}, \langle \mathbf{d}_2 \rangle \models g$ for every base fluent g other than f or $\mathbf{neg}\, f$. And from the definition of $insert(f)$, we know that $\mathcal{M}, \langle \mathbf{d}_2 \rangle \models f$. It follows that $\mathbf{D}(\mathbf{d}_2)$ is also consistent and complete.

Induction step: $N = k$, and assume that whenever (B.1) can be derived by the proof theory in less than k steps, then consistency of $\mathbf{D}(\mathbf{d}_1)$ entails consistency of $\mathbf{D}(\mathbf{d}_n)$. If, in addition, $\mathbf{D}(\mathbf{d}_1)$ is complete then so is $\mathbf{D}(\mathbf{d}_n)$. Observe that (B.1) can possibly be derived only via one of the following rules: Applying transaction definition Rule or Sequencing Rule. We will consider each possibility in turn. Since α is an action, (B.1) cannot be derived neither by the Forward Projection Rule, nor by the Decomposition Rule.

- Applying transaction definition Rule: Suppose that α is a composed action, and there is a rule in $\mathbf{P}$ of the form

  $$\alpha \leftarrow \beta$$

  and (B.1) was derived via the Applying transaction definition Rule. Then we know that

  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \beta \tag{B.3}$$

  was derived in less than k steps. By Inductive Hypothesis, if $\mathbf{D}(\mathbf{d}_1)$ is consistent and complete, then so is $\mathbf{D}(\mathbf{d}_n)$.

- Sequencing Rule: Suppose that $\alpha = \beta \otimes \gamma$, and (B.1) was derived via the Sequencing Rule. Then we know that

  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_k \vdash \beta \quad \text{and} \quad \mathbf{Q}, \mathcal{S}, \mathbf{d}_k \ldots \mathbf{d}_n \vdash \gamma \tag{B.4}$$

  were derived in less than k steps. By Inductive Hypothesis, if $\mathbf{D}(\mathbf{d}_1)$ is consistent and complete, then so is $\mathbf{D}(\mathbf{d}_k)$, and by inductive hypothesis again, so is $\mathbf{D}(\mathbf{d}_n)$. □

B.0.13. THEOREM (SOUNDNESS). *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$*. Suppose that* $\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \dots \mathbf{d}_n \models h$*. Then there exist relational database states* $\mathbf{D}_1$*, ...,* $\mathbf{D}_{n-1}$ *(in* $\mathcal{L}_{\mathcal{TR}}$*) such that*

$$\mathbf{P}, \mathbf{D}, \mathbf{D}_1 \dots \mathbf{D}_{n-1}, \mathbf{D}(\mathbf{d}_n) \models h$$

where $\mathbf{D}(\mathbf{d}_n)$ *is as in Definition 3.4.7.*

Proof. The proof relies on the fact that $\mathcal{TR}^{PAD}$ has a sound and complete proof theory. We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \dots \mathbf{d}_n \vdash h \tag{B.5}$$

Base case: $N = 1$. In that case, (B.5) can only be derived by the (run or state) premise inference rule or by the axiom in $\mathcal{F}$. We consider each case in turn:

- Suppose that (B.5) was derived by the axiom in $\mathcal{F}$, then it follows that (B.5) has the form: $\mathbf{P}, \mathbf{d} \vdash ()$. The claim follows by the axioms in the Horn-$\mathcal{TR}^-$ proof system.

- Suppose that and (B.5) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_0 \vdash h$ for some fluent literal h, and it was derived by a state premise inference rule. By the definition of $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$, it follows that $\mathbf{d}_0 \rhd h \in \mathcal{S}$ if and only if $h \in \mathbf{D}$. Thus, $\mathbf{D}(\mathbf{d}_0) = \mathbf{D}$ and by definition of satisfaction in Horn-$\mathcal{TR}^-$, we can conclude that $\mathbf{P}, \mathbf{D} \models h$.

- Suppose that (B.5) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \mathbf{d}_{n+1} \vdash \alpha$ for some partially defined action α, and it was derived by a run premise inference rule. The cases where α is an *insert* or a *delete* are completely symmetrical. For concreteness assume that $\alpha = insert(f)$. By definition of $\mathcal{S}$ we know that $\mathbf{d}_{n+1} = \mathbf{d}_{n,\alpha}$ and there is a run premises in $\mathcal{S}$ of the form

  $$\mathbf{d}_n \overset{\alpha}{\rightsquigarrow} \mathbf{d}_{n+1}$$

  From the Forward Inertia frame axioms, it follows that for every model $\mathcal{M}$ of $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$, it holds that:

  1. For every base fluent g

     $$\begin{aligned} &\{g \mid \mathcal{M}, \langle \mathbf{d}_n \rangle \models g \quad \text{and} \quad g \neq f \text{ and } g \neq \mathbf{neg}\, f\} \\ &= \\ &\{g \mid \mathcal{M}, \langle \mathbf{d}_{n,\alpha} \rangle \models g \quad \text{and} \quad g \neq f \text{ and } g \neq \mathbf{neg}\, f\} \end{aligned}$$

2. From the definition of $insert(f)$, we know that

$$\mathcal{M}, \langle \mathbf{d}_{n,\alpha} \rangle \models f$$

Therefore it follows that

$$\mathbf{D}(\mathbf{d}_{n,\alpha}) = \mathbf{D}(\mathbf{d}_n) \cup \{f\} \setminus \{\mathbf{neg}\, f\}$$

this implies that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}_n)\mathbf{D}(\mathbf{d}_{n,\alpha}) \vdash \alpha$$

Induction step: $N = k > 1$ and assume that whenever (B.5) can be derived by the proof theory in less than k steps, then there are relational states $\mathbf{D}_1 \ldots \mathbf{D}_n$ such that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}), \mathbf{D}_1 \ldots \mathbf{D}_{n-1}, \mathbf{D}(\mathbf{d}_n) \vdash h$$

- ***Forward Projection:*** Suppose that (B.5) was derived via the Forward Projection rule. This means that (B.5) was derived using a PAD $\mathbf{p} \in \mathbf{Q}$ that belongs to one of the following types of rules:
  - A Frame Axiom
  - The encoding of an elementary action.

  We consider each of these cases in turn:

  - Suppose (B.5) was derived via the Forward Projection rule and $\mathbf{p}$ is a Forward Inertia *pda*. This implies that:
    1. $\mathbf{p}$ has the form: $inertial(f) \wedge f \otimes \alpha \rightarrow \alpha \otimes f$
    2. (B.5) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f$
    3. neither f nor $\mathbf{neg}\, f$ is a primitive effect of α, and,
    4. the following statements were derived in less than k steps:
       (a) $\mathbf{Q}, \mathcal{S}, \mathbf{d}_{n-1} \vdash f$
       (b) $\mathbf{Q}, \mathcal{S}, \mathbf{d}_{n-1}, \mathbf{d}_n \vdash \alpha$

    by inductive hypothesis:

$$\mathbf{P}, \mathcal{S}, \mathbf{D}(\mathbf{d}_{n-1}) \vdash f$$
$$\mathbf{P}, \mathcal{S}, \mathbf{D}(\mathbf{d}_{n-1})\mathbf{D}(\mathbf{d}_n) \vdash \alpha$$

    The cases where α is an *insert* or a *delete* are completely symmetrical. For concreteness assume that $\alpha = insert(g)$ where $g \neq f$ and $g \neq \mathbf{neg}\, f$. Therefore, it follows by definition of the built-in operation $insert(g)$ in $\mathcal{TR}^-$ that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}_n) \models f$$

– Suppose (B.5) was derived via the Forward Projection rule and $\mathbf{p}$ is the definition of *insert* or a *delete*. The cases where $\mathbf{p}$ defines an *insert* or a *delete* are completely symmetrical. For concreteness assume that $\mathbf{p}$ defines $insert(f)$. This implies that:

1. $\mathbf{p}$ has the form: $\alpha \rightarrow \alpha \otimes f$
2. (B.5) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f$
3. the following statement were derived in less than k steps: $\mathbf{Q}, \mathcal{S}, \mathbf{d}_{n-1}, \mathbf{d}_n \vdash \alpha$

by inductive hypothesis:

$$\mathbf{Q}, \mathcal{S}, \mathbf{D}(\mathbf{d}_{n-1})\mathbf{D}(\mathbf{d}_n) \vdash \alpha$$

It follows that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}_n) \models f$$

- ***Decomposition Rule***: Suppose (B.5) was derived via the Decomposition Rule rule. This implies that the following statements was derived in less than k steps:

$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \vdash \psi \otimes \phi$$

where ϕ and ψ are serial conjunction of fluent literals. The inductive claim trivially follows from the inductive hypothesis.

- ***Sequencing Rule***: Suppose (B.5) was derived via the Sequencing Rule rule. This implies that the following statements were derived in less than k steps:

$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_n \vdash \phi$$
$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_n \vdash \psi$$

Then, by the inductive hypothesis, there are relational states $\mathbf{D}_1 \ldots \mathbf{D}_{n-1}$ such that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}), \mathbf{D}_1 \ldots \mathbf{D}_{k-1}, \mathbf{D}(\mathbf{d}_k) \models \phi$$
$$\mathbf{P}, \mathbf{D}(\mathbf{d}_k), \mathbf{D}_1 \ldots \mathbf{D}_{n-1}, \mathbf{D}(\mathbf{d}_n) \models \psi$$

It follows that

$$\mathbf{P}, \mathbf{D}(\mathbf{d}), \mathbf{D}_1 \ldots \mathbf{D}_{n-1}\mathbf{D}(\mathbf{d}_n) \models \phi \otimes \psi$$

- Applying transaction definition Rule: Suppose that h is a composed action, and there is a rule in $\mathbf{P}$ of the form

$$h \leftarrow \beta$$

and (B.1) was derived via the Applying transaction definition Rule. Then we know that

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \beta \tag{B.6}$$

was derived in less than k steps. The claim follows from the Inductive Hypothesis. This concludes the soundness proof. □

B.0.14. THEOREM (COMPLETENESS). *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$*. Suppose that*

$$\mathbf{P}, \mathbf{D}, \ldots \mathbf{D}_n \models h$$

then there are state identifiers $\mathbf{d}_1 \ldots \mathbf{d}_n$ *such that*

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_n \models h$$

Proof. The proof relies on the fact that serial-Horn Transaction Logic has a sound and complete proof theory [11]. We reproduce a ground version of that theory below. This ground version suffices for the purpose of our proof, since the problem can be reduced to the case where $\mathbf{P}$ is ground.

TR_0 **(axiom):** $\mathbf{P}, \mathbf{D} \vdash ()$, where $()$ is an empty serial conjunction of actions, which we will view as a special fluent that is true in every state.

TR_1 **(folding):** Suppose $\alpha \leftarrow \beta \in \mathbf{P}$. Then, for any sequence of database states $\mathbf{D}_1, ..., \mathbf{D}_n$, from $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \beta \otimes \gamma$ derive $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \alpha \otimes \gamma$.

TR_2 **(hypothetical):** From $\mathbf{P}, \mathbf{D}, \mathbf{D}'_1, ..., \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, ..., \mathbf{D}_m \vdash \gamma$ derive $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, ..., \mathbf{D}_m \vdash \Diamond\beta \otimes \gamma$.

TR_3 **(query):** Suppose f is a fluent such that $f \in \mathbf{D}_1$. Then from $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash f \otimes \beta$.

TR_4 **(update):** Suppose u is an elementary transition ($insert(f)$ or $delete(f)$) such that $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 \models u$. Then from $\mathbf{P}, \mathbf{D}_2 ... \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 ... \mathbf{D}_n \vdash u \otimes \beta$.

Observe that we only need to consider states which are "reachable" from $\mathbf{D}$. We say that $\mathbf{D}_1$ is reachable from $\mathbf{D}$, if there a serial conjunction of elementary actions ϕ such that

$$\mathbf{P}, \mathbf{D}, \ldots, \mathbf{D}_1 \vdash \phi \tag{B.7}$$

Let $\mathbf{D}_1$ be reachable from $\mathbf{D}$. We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash h \tag{B.8}$$

using the above inference rules and the axiom.
Base case: $N = 1$. In that case, (B.8) must have been derived by the axiom TR_0 and thus must have the form $\mathbf{P}, \mathbf{D} \vdash ()$. The proof of this case follows from Axiom 1 in $\mathcal{F}$.
Inductive case: $N = k > 1$. Suppose that whenever (B.8) can be derived by the above proof theory in less than k steps then there are state identifiers $\mathbf{d}_1 \ldots \mathbf{d}_n$ such that

$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_n \models h$$

To prove that the same holds also when (B.8) is derived using k steps, note that the last step in the derivation must be an application of one of the rules TR_1, ..., TR_4. The cases where the last step in the derivation of (B.8) was either TR_1, or TR_2, follow straightforwardly from inductive hypothesis and rules 1a and 1b respectively. We consider each of the remaining two cases in turn.

- TR_4: (B.8) was derived because $h = u \otimes \beta$, where u is an elementary transition such that $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models u$, and $\mathbf{P}, \mathbf{D}_2 \ldots \mathbf{D}_n \vdash \beta$.

  Since $\mathbf{D}_1$ is reachable from $\mathbf{D}$ with a finite number of *insert* and *delete* operations, we know that there is serial conjunction of elementary actions ϕ s.t. $\mathbf{P}, \mathbf{D} \ldots \mathbf{D}_1 \vdash \phi$, can be derived by the above proof theory.

  Thus, by construction of $\mathcal{S}$ we know that there is a database state identifier $\mathbf{d}_\phi$ such that

  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_\phi \models \phi$$

  Suppose for concreteness that $u = insert(g)$. From the definition of $\mathcal{S}$ we know that

  $$\mathbf{d}_\phi \overset{insert(g)}{\rightsquigarrow} \mathbf{d}_{\phi,insert(g)} \in \mathcal{S}$$

  Therefore

  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_\phi, \mathbf{d}_{\phi,insert(g)} \vdash u$$

  The claim now follows from 4 in $\mathcal{F}$. □

- TR_3: (C.1) was derived because $\alpha = f \otimes \beta$, $f \in \mathbf{D}_1$, and $\mathbf{P}, \mathbf{D}_1 \ldots \mathbf{D}_n \vdash \beta$ was derived previously. Since $\mathbf{D}_1$ is reachable from $\mathbf{D}$ with a finite number of *insert* and *delete* operations ϕ, we know that either

  1. $f \in \mathbf{D} =$ and f was not removed by action ϕ, or
  2. it was inserted by some *insert* action in ϕ.

  In the first case, by definition, we know that

  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \models f$$
  $$\mathbf{Q}, \mathcal{S}, \mathbf{d}_0 \models inertial(f)$$

thus the claim follows from rules 2 and 4 in $\mathcal{F}$.

The second case follows straightforwardly using the premises and following a similar reasoning as above. □

Appendix C

Proofs of the reduction of Serial Horn-$\mathcal{TR}^-$ to Logic Programming

This appendix contains proofs of soundness and completeness of the reduction of serial Horn-$\mathcal{TR}^-$ to LP developed in Section 3.5. We assume that all transactions are serial goals, and that the transaction base is a set of serial Horn rules. For convenient reference we reproduce some of the definitions below.

C.0.15. Definition. [Consistency and completeness of `state`-terms] Let $\Gamma(\mathbf{P}, \mathbf{D})$ be the LP reduction of a serial-Horn TR program $(\mathbf{P}, \mathbf{D})$ and let s be a ground `state`-term. We say that s is **complete** if and only if for any ground *base* `fluent`-term f

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, s) \quad \text{or} \quad \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(\mathbf{neg}\, f, s)$$

We will say that s is **consistent** if and only if there is no ground *base* `fluent`-term f such that both of the following hold:

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, s) \quad \text{and} \quad \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(\mathbf{neg}\, f, s)$$

□

We will now establish a number of properties of the LP-reduction.

C.0.16. Proposition (State consistency and completeness). *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a relational serial-Horn Transaction Logic program* $(\mathbf{P}, \mathbf{D})$. *Let* s, $\hat{s}$ *be ground* `state`*-terms such that* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, s)$ *holds, where* α *is a ground* `action`*-term. If* $\hat{s}$ *is consistent then so is* s. *If, in addition,* $\hat{s}$ *is complete then* s *is also complete.*

Proof. Recall that $\Gamma(\mathbf{P}, \mathbf{D})$ has a unique least Herbrand model, $\mathbf{M}$, and, therefore, $\Gamma(\mathbf{P}, \mathbf{D}) \models \mathit{Execute}(\alpha, \hat{s}, s)$ if and only if $\mathit{Execute}(\alpha, \hat{s}, s) \in \mathbf{M}$. This model is computed via a sequence of bottom-up derivation steps, which apply the rules of $\Gamma(\mathbf{P}, \mathbf{D})$ to the facts in $\Gamma(\mathbf{P}, \mathbf{D})$ and then repeatedly to the newly derived facts. Our proof will proceed by induction on the number N of such steps. We will prove only the second claim, namely, that consistency and completeness of $\hat{s}$ implies these properties for s. A proof of the fact that consistency alone (without completeness) of $\hat{s}$ implies consistency for s can be obtained by disregarding the completeness considerations in the proof below.

Base case: $N = 1$. This means that $\mathit{Execute}(\alpha, \hat{s}, s)$ is a fact in $\Gamma(\mathbf{P}, \mathbf{D})$ and thus it can be derived by the rule Execution only. Therefore, α is an elementary action and $s = \mathit{Result}(\alpha, \hat{s})$. Since insert and delete actions are symmetric in $\Gamma(\mathbf{P}, \mathbf{D})$, let us assume for concreteness that $\alpha = \mathit{insert}(f)$ for some fluent f. By the rule Inertial, all base fluents except f and $\mathbf{neg}\, f$ are inertial with respect to α. By Frame Axiom, this means that, for every base ground `fluent`-term h other than f and $\mathbf{neg}\, f$,[1] $\mathit{Holds}(h, s) \in \mathbf{M}$ if $\mathit{Holds}(h, \hat{s}) \in \mathbf{M}$. Since $\hat{s}$ is a complete and consistent state, it follows that for every fluent other than f or $\mathbf{neg}\, f$, the fluent or its negation holds in s. For the remaining fluents f and $\mathbf{neg}\, f$, the rule Effect+ yields $\mathit{Holds}(f, s) \in \mathbf{M}$ while $\mathit{Holds}(\mathbf{neg}\, f, s)$ can be derived neither by Frame Axiom nor by the Effect axioms—the only rules that can possibly derive *Holds*-facts for states other than s_0. This establishes the base case of the induction.

Induction step: $N = k$, where $k > 1$. Assume that the claim holds for all facts of the form $\mathit{Execute}(\alpha, \hat{s}, s)$ that were derived via $k-1$ or fewer derivation steps. $\mathit{Execute}(\alpha, \hat{s}, s)$ can possibly be derived only via one of the following rules: Unfolding, Sequencing, Hypothetical, or Query. We will consider each possibility in turn.

Unfolding: Suppose $\mathit{Execute}(\alpha, \hat{s}, s)$ was derived via a ground instance

$$\mathit{Execute}(\alpha, \hat{s}, s) \leftarrow \mathit{Execute}(\beta, \hat{s}, s)$$

of the rule Unfolding. This means that $\mathit{Execute}(\beta, \hat{s}, s) \in \mathbf{M}$, and it has been derived before $\mathit{Execute}(\alpha, \hat{s}, s)$, i.e., using $< k$ steps. Hence, s is consistent and complete, by the inductive hypothesis.

Sequencing: Suppose that $\alpha = \beta \otimes \gamma$, and $\mathit{Execute}(\alpha, \hat{s}, s)$ was derived via a ground instance

$$\mathit{Execute}(\beta \otimes \gamma, \hat{s}, s) \leftarrow \mathit{Execute}(\beta, \hat{s}, s'), \mathit{Execute}(\gamma, s', s)$$

of the rule Sequencing. This means that $\mathit{Execute}(\beta, \hat{s}, s')$ and $\mathit{Execute}(\gamma, s', s)$ have already been derived in less than k steps. By the inductive hypothesis,

[1] Recall that, by convention, double-negation cancels out.

since $\hat{s}$ is consistent and complete, so is s'. Applying the inductive hypothesis again to $Execute(\gamma, s', s)$, we conclude that s is also consistent and complete.

Hypothetical: Suppose $\alpha = \Diamond\beta$, and $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\Diamond\beta, \hat{s}, \hat{s}) \leftarrow Execute(\beta, \hat{s}, s')$$

of the rule Hypothetical. Since here $s = \hat{s}$, the claim follows trivially.

Query: Suppose $Execute(\alpha, \hat{s}, s)$ was derived by the the rule Query. The argument here is the same as in the case of the rule Hypothetical: $s = \hat{s}$ and therefore s both consistent and complete. □

C.0.17. DEFINITION. [Correspondence between states in $\mathcal{L}_{LP}$ and $\mathcal{L}_{TR}$] Given a ground `state`-term t in $\mathcal{L}_{LP}$, let $\mathbf{D}(t)$ denote the following set of database fluents in the language $\mathcal{L}_{TR}$ of Transaction Logic:

$$\mathbf{D}(t) \;=\; \{f \mid f \text{ is a ground } \textit{base } \texttt{fluent}\text{-term such that } \Gamma(\mathbf{P}, \mathbf{D}) \models Holds(f, t)\} \quad \square$$

C.0.18. THEOREM (SOUNDNESS). *Let $\Gamma(\mathbf{P}, \mathbf{D})$ be an LP-reduction of a relational serial-Horn TR program $(\mathbf{P}, \mathbf{D})$ and suppose that $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, s)$, where $\hat{s}$ and s are ground* `state`*-terms and $\hat{s}$ is consistent. Then there exist relational database states $\mathbf{D}_1, \ldots, \mathbf{D}_n$ (in $\mathcal{L}_{TR}$) such that*

$$\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n\mathbf{D}(s) \;\models\; \alpha$$

where $\mathbf{D}(\hat{s})$ and $\mathbf{D}(s)$ are as in Definition 3.5.3.

Proof. The proof is by induction on the number N of derivation steps needed to conclude $Execute(\alpha, \hat{s}, s) \in \mathbf{M}$, where $\mathbf{M}$ is the unique least model of $\Gamma(\mathbf{P}, \mathbf{D})$. Observe that since $\hat{s}$ is consistent, so is s, by Proposition 3.5.2,

Base case: $N = 1$, i.e., $Execute(\alpha, \hat{s}, s) \in \mathbf{M}$ was derived in just one derivation step. This could be done only via the rule Execution, and in this case α is a an elementary action $insert(f)$ or $delete(f)$ and $s = Result(\alpha, \hat{s})$. Recall that the treatment of insert and delete actions in $\Gamma(\mathbf{P}, \mathbf{D})$ is completely symmetric. For concreteness, we assume that $\alpha = delete(f)$.

Similarly to the proof of Proposition 3.5.2, it is easy to show by direct inspection of the rules in $\Gamma(\mathbf{P}, \mathbf{D})$ that if g is unrelated to f and both f and g are ground base fluents then $Holds(g, \hat{s}) \in \mathbf{M}$ iff $Holds(g, s) \in \mathbf{M}$.

Concerning f, we know from the rule Effect- that $Holds(\mathbf{neg}\, f, s) \in \mathbf{M}$ and that (by consistency and completeness of $\hat{s}$) either $Holds(f, \hat{s})$ or $Holds(\mathbf{neg}\, f, \hat{s})$ holds, but not both. Therefore $\mathbf{D}(s) = \mathbf{D}(\hat{s}) - \{f\} + \{\mathbf{neg}\, f\}$. By the definition of the relational deletion operations in Transaction Logic, it follows that $\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}(s) \models \alpha$.

Inductive hypothesis: $N = k > 1$ and assume that the claim holds for all statements $Execute(\alpha, \hat{s}, s)$ that are derivable via less than k derivation steps using the rules in $\Gamma(\mathbf{P}, \mathbf{D})$. As in earlier proofs, $Execute(\alpha, \hat{s}, s)$ can possibly be derived only via one of the following rules: Unfolding, Sequencing, Hypothetical, or Query. So we will consider each possibility in turn.

Unfolding: Suppose $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\alpha, \hat{s}, s) \leftarrow Execute(\beta, \hat{s}, s)$$

of rule Unfolding. This implies the following:

- $\alpha \leftarrow \beta$ is a ground instance of an implication in $\mathbf{P}$
- $Execute(\beta, \hat{s}, s) \in \mathbf{M}$, and it has been derived before $Execute(\alpha, \hat{s}, s)$, i.e., using $< k$ steps.

By the inductive hypothesis, $\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n\mathbf{D}(s) \models \beta$ for some intermediate database states $\mathbf{D}_1$, ..., $\mathbf{D}_n$, and $\mathbf{D}(s)$ is consistent. This and the fact that $\alpha \leftarrow \beta$ is an instance of an implication in $\mathbf{P}$ implies $\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n\mathbf{D}(s) \models \alpha$, by the definition of implication in TR.

Sequencing: Suppose that $\alpha = \beta \otimes \gamma$, and $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\beta \otimes \gamma, \hat{s}, s) \leftarrow Execute(\beta, \hat{s}, s'), Execute(\gamma, s', s)$$

of rule Sequencing. This means that $Execute(\beta, \hat{s}, s')$ and $Execute(\gamma, s', s)$ have already been derived in less than k steps. Since $\hat{s}$ is consistent and complete, Proposition 3.5.2 ensures that so are s' and s. By the inductive hypothesis, we conclude that

$$\mathbf{P}, \mathbf{D}(\hat{s})\mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_m\mathbf{D}(s') \models \beta$$
$$\mathbf{P}, \mathbf{D}(s')\mathbf{D}_{m+1}\mathbf{D}_{m+2} \ldots \mathbf{D}_n\mathbf{D}(s) \models \gamma$$

for some intermediate states $\mathbf{D}_1$, ..., $\mathbf{D}_n$, and that $\mathbf{D}(s')$, $\mathbf{D}(s)$ are consistent. The claim now follows from the definition of serial conjunction $\otimes$ in TR.

Hypothetical: Suppose $\alpha = \Diamond\beta$, and $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\Diamond\beta, \hat{s}, \hat{s}) \leftarrow Execute(\beta, \hat{s}, s')$$

of the rule Hypothetical. By the inductive hypothesis, $\mathbf{P}, \mathbf{D}(\hat{s}) \ldots \mathbf{D}(s') \models \beta$. Therefore, the definition of the hypothetical operator in TR yields $\mathbf{P}, \mathbf{D}(\hat{s}) \models \Diamond\beta$.

Query: Suppose $Execute(\alpha, \hat{s}, s)$ was derived by the rule Query. This means that $Execute(\alpha, \hat{s}, s)$ was derived via a rule of the form $Execute(f, \hat{s}, \hat{s}) \leftarrow Holds(f, \hat{s})$ and $Holds(f, \hat{s}) \in \mathbf{M}$, where f is a ground base fluent. In particular, $\alpha = f$ and $s = \hat{s}$. By Definition 3.5.3, $f \in \mathbf{D}(\hat{s})$ and, by the definition of executional entailment for fluents in TR, $\mathbf{P}, \mathbf{D}(\hat{s}) \models f$. □

C.0.19. THEOREM (COMPLETENESS). *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a relational serial-Horn TR program* $(\mathbf{P}, \mathbf{D})$. *Suppose* $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_n\bar{\mathbf{D}} \models \alpha$, *where* $\hat{\mathbf{D}} = \mathbf{D}(\hat{s})$ *for some consistent ground* `state`*-term* $\hat{s}$. *Then there is a consistent ground* `state`*-term* $\bar{s}$ *such that* $\bar{\mathbf{D}} = \mathbf{D}(\bar{s})$ *and* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.[2]

Proof. The proof relies on the fact that serial-Horn Transaction Logic has a sound and complete proof theory [11]. We reproduce a ground version of that theory below. This ground version suffices for the purpose of our proof, since the problem can be reduced to the case where $\mathbf{P}$ is ground.

TR_0 **(axiom):** $\mathbf{P}, \mathbf{D} \vdash ()$, where $()$ is an empty serial conjunction of actions, which we will view as a special fluent that is true in every state.

TR_1 **(folding):** Suppose $\alpha \leftarrow \beta \in \mathbf{P}$. Then, for any sequence of database states $\mathbf{D}_1, ..., \mathbf{D}_n$, from $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \beta \otimes \gamma$ derive $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \alpha \otimes \gamma$.

TR_2 **(hypothetical):** From $\mathbf{P}, \mathbf{D}, \mathbf{D}'_1, ..., \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, ..., \mathbf{D}_m \vdash \gamma$ derive $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, ..., \mathbf{D}_m \vdash \Diamond\beta \otimes \gamma$.

TR_3 **(query):** Suppose f is a fluent such that $f \in \mathbf{D}_1$. Then from $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash \beta$
derive $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \vdash f \otimes \beta$.

TR_4 **(update):** Suppose u is an elementary transition ($insert(f)$ or $delete(f)$) such that $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 \models u$. Then from $\mathbf{P}, \mathbf{D}_2 ... \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 ... \mathbf{D}_n \vdash u \otimes \beta$.

We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \hat{\mathbf{D}}, \mathbf{D}_1, \ldots, \mathbf{D}_n, \bar{\mathbf{D}} \vdash \alpha \tag{C.1}$$

using the above inference rules and the axiom.

Base case: $N = 1$. In that case, (C.1) must have been derived by the axiom TR_0 and thus must have the form $\mathbf{P}, \hat{\mathbf{D}} \vdash ()$, where $\hat{\mathbf{D}} = \bar{\mathbf{D}}$ (and thus $\hat{s} = \bar{s}$). Since $()$ is treated as a fluent that is true in every state, the rule Query of $\Gamma(\mathbf{P}, \mathbf{D})$ ensures that $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute((), \hat{s}, \bar{s})$, since $\hat{s} = \bar{s}$.

Inductive case: $N = k > 1$. Suppose that whenever (C.1) can be derived by the above proof theory in less than k steps then $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$. To prove that the same holds also when (C.1) is derived using k steps, note that the last step in the derivation must be an application of one of the rules TR_1, ..., TR_4. We consider each of these cases in turn.

TR_1: (C.1) was derived because $\alpha \leftarrow \beta \in \mathbf{P}$ and $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 ... \mathbf{D}_n\bar{\mathbf{D}} \vdash \beta \otimes \gamma$ was derived previously, in less than k steps. By the inductive assumption,

[2] $\mathbf{D}(\hat{s})$ and $\mathbf{D}(\bar{s})$ were introduced in Definition 3.5.3.

$\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\beta \otimes \gamma, \hat{s}, \bar{s})$ must hold. But then, by the rule Unfolding of $\Gamma(\mathbf{P},\mathbf{D})$ we can derive $\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.

TR_2: (C.1) was derived because $\alpha = \Diamond\beta \otimes \gamma$ and both $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}'_1 ... \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 ... \mathbf{D}_m \bar{\mathbf{D}} \vdash \gamma$ were derived previously via less than k steps. By the inductive assumption, $\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\beta, \hat{s}, s'_n)$, for some consistent state s'_n, and $\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\gamma, \hat{s}, \bar{s})$. But then rules Hypothetical and Sequencing yield $\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.

TR_3: (C.1) was derived because $\alpha = f \otimes \beta$, $f \in \hat{\mathbf{D}}$, and $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_n \bar{\mathbf{D}} \vdash \beta$ was derived previously. Since $f \in \hat{\mathbf{D}}$, Definition 3.5.3 implies $\Gamma(\mathbf{P},\mathbf{D}) \models Holds(f, \hat{s})$. The inductive assumption also gives us $\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\beta, \hat{s}, \bar{s})$. The inductive claim now follows from these two facts and the rules Query and Sequencing.

TR_4: (C.1) was derived because $\alpha = u \otimes \beta$, where u is an elementary transition such that $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \models u$, and $\mathbf{P}, \mathbf{D}_1 \ldots \mathbf{D}_n \bar{\mathbf{D}} \vdash \beta$ was derived earlier. By the rule Execution, we have

$$\Gamma(\mathbf{P},\mathbf{D}) \models Execute(u, \hat{s}, Result(u, \hat{s})) \tag{C.2}$$

Moreover, it is easy to show from the definitions of $insert(f)$, $delete(f)$ and the rules Effect+, Effect-, and Frame Axiom that $\mathbf{D}_1 = \mathbf{D}(Result(u, \hat{s}))$. This and the inductive assumption lets us conclude

$$\Gamma(\mathbf{P},\mathbf{D}) \models Execute(\beta, Result(u, \hat{s}), \bar{s}) \tag{C.3}$$

The inductive claim now follows from (C.2), (C.3), and the rule Sequencing. This concludes the proof. □

Appendix D

Proofs for the Reduction of $\mathcal{TR}_{D}^{PAD}$ to Logic Programming

In this chapter we prove soundness and completeness of the reduction of $\mathcal{TR}_{D}^{PAD}$ to sorted Horn logic programming developed in Section 3.6.

D.0.20. Lemma. *Let* $\Gamma(\mathbf{P}, \mathcal{S})$ *be an LP-reduction of a* $\mathcal{TR}_{D}^{PAD}$ *specification* $(\mathbf{P}, \mathcal{S})$. *Suppose* $s = db2st_{\mathcal{S}}(\mathbf{d})$, *then* $\mathbf{d} = st2db(s)$.

Proof. The proof is by induction on the structure of the `state`-term $db2st_{\mathcal{S}}(\mathbf{d}) = s$.

Base case: s is a `state`-constant. The claim follows directly from Definition 3.6.2.

Inductive step: Suppose $db2st_{\mathcal{S}}(\mathbf{d}) = s$, and $s = Result(\alpha, s_0)$ for some *pda* α and a `state`-term s_0. By definition, there must be a premise $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$ such that $s_0 = db2st_{\mathcal{S}}(\mathbf{d}_0)$. By the inductive hypothesis, $\mathbf{d}_0 = st2db(s_0)$. From this and the definition of $st2db$ it follows that $\mathbf{d} = st2db(s)$. □

D.0.21. Lemma. *Let* $\Gamma(\mathbf{P}, \mathcal{S})$ *be an LP-reduction of a* $\mathcal{TR}^{PAD}$ *specification* $(\mathbf{P}, \mathcal{S})$. *Let* s *be a ground* `state`*-term. Suppose* $Holds(f, s)$ *is a fact in* $\Gamma(\mathbf{P}, \mathcal{S})$. *Then*

1. $st2db(s)$ *is defined and*
2. $\mathbf{P}, \mathcal{S}\ st2db(s) \models f$.

As a special case, we get $\mathbf{P}, \mathcal{S}, st2db(s) \models ()$.

Proof. Suppose $Holds(f, s)$ is a fact in $\Gamma(\mathbf{P}, \mathcal{S})$. By construction of $\Gamma(\mathbf{P}, \mathcal{S})$, $Holds(f, s)$ must have gotten into $\Gamma(\mathbf{P}, \mathcal{S})$ due to the Premises part of the

construction because $\mathcal{S}$ has a *state*-premise of the form $\mathbf{d} \rhd f$ such that $s = db2st_{\mathcal{S}}(\mathbf{d})$. (If $f = ()$, then the same conclusion follows from the No-op rule.) By Lemma D.0.20, we conclude that $\mathbf{d} = st2db(s)$ and, therefore, $st2db(s)$ is defined. Also, by the Premise inference rule, $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash f$ and, by the soundness of the inference system, $\mathbf{P}, \mathcal{S}, \mathbf{d} \models f$. □

The following technical results are used in the proof of soundness of the reduction from $\mathcal{TR}_D^{PAD}$ to logic programming.

D.0.22. LEMMA. *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose that, for some ground* `state`*-terms s_1 and s_2 and a pda α*

- $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$
- $st2db(s_1)$, $st2db(s_2)$ *are defined, and*
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \alpha$, *where* $\mathbf{d}_1 = st2db(s_1)$ *and* $\mathbf{d}_2 = st2db(s_2)$

Then

- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathbf{d}(s_1)$
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models \mathbf{d}(s_2)$

Proof. Recall that $\Gamma(\mathbf{P}, \mathcal{S})$ has a unique least Herbrand model, $\mathbf{M}$, and, therefore, for any ground predicate p, $\Gamma(\mathbf{P}, \mathcal{S}) \models p$ if and only if $p \in \mathbf{M}$. This model is computed via a sequence of bottom-up derivation steps, which apply the rules of $\Gamma(\mathbf{P}, \mathcal{S})$ to the facts in $\Gamma(\mathbf{P}, \mathcal{S})$ and then repeatedly to the newly derived facts.

The proof is by induction on the number N of derivation steps needed to conclude $Holds(f, s_1) \in \mathbf{M}$ (or $Holds(f, s_2) \in \mathbf{M}$). Since the proofs for $Holds(f, s_1)$ and $Holds(f, s_2)$ are almost identical, we only give a proof for $Holds(f, s_1)$.

Base case: $N = 1$. Observe that $Holds(f, s_1)$ can be derived in one step only if $Holds(f, s_1)$ is a fact in $\Gamma(\mathbf{P}, \mathcal{S})$. The claim now follows from Lemma D.0.21.

Inductive step: $N = k > 1$. Suppose that the claim is true for all `state`-terms s_1, s_2 such that $Holds(f, s_1)$ was derived in less than k steps using the rules in $\Gamma(\mathbf{P}, \mathcal{S})$.

Note that, after the first iteration, the *Holds* facts can be derived only via the Forward Projection rules in $\Gamma(\mathbf{P}, \mathcal{S})$, so examine this case.

Forward Projection: Suppose $Holds(f, s_1)$ was derived by a ground instance of one of the following the rules

$$\begin{aligned} Holds(b_3, S) \leftarrow \quad & Execute(\alpha, S, Result(\alpha, S)), \\ & Execute(b_1, S, S), Execute(b_2, Result(\alpha, S), Result(\alpha, S)) \\ Holds(b_4, Result(\alpha, S)) \leftarrow \quad & Execute(\alpha, S, Result(\alpha, S)), \\ & Execute(b_1, S, S), Execute(b_2, Result(\alpha, S), Result(\alpha, S))) \end{aligned}$$

This implies that

- there is a PAD of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ in $\mathbf{P}$.
- $Execute(\alpha, s_1, Result(\alpha, s_1))$, $Execute(b_1, S, S)$ and $Execute(b_2, Result(\alpha, S), Result(\alpha, S))$ were added to $\mathbf{M}$ in less than k derivation steps.

The cases where f is derived by either of the two forward projection rules above are analogous, so, for concreteness, we assume that f is derived by the first rule and that f is a conjunct in b_3.

Recall that, by assumption, $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \alpha$, where $\mathbf{d}_1 = st2db(s_1)$ and $\mathbf{d}_2 = st2db(Result(\alpha, s_1))$, and that by the inductive hypothesis, we have

- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_1$
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models b_2$.

Since the antecedent of the aforesaid PAD $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ is satisfied, the definition of satisfaction in TR implies that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_3$. Since f is a conjunct in b_3, we conclude that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models f$. □

D.0.23. PROPOSITION. *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}^{PAD}_{D}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$ where s_1 and s_2 are ground* `state`*-terms. Then*

- *$st2db(s_1)$ and $st2db(s_2)$ are defined and*
- *$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \phi$, where $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d}_2 = st2db(s_2)$*

Proof. The proof is by induction on the number N of derivation steps needed to conclude $Execute(\phi, s_1, s_2) \in \mathbf{M}$, where $\mathbf{M}$ is the unique least model of $\Gamma(\mathbf{P}, \mathcal{S})$.

Base case. $N = 1$: Let s_1 and s_2 be ground `state`-terms. Suppose $Execute(\phi, s_1, s_2)$ was derived in just one derivation step. This means that $Execute(\phi, s_1, s_2) \in \Gamma(\mathbf{P}, \mathcal{S})$ and it got into $\Gamma(\mathbf{P}, \mathcal{S})$ due to the premise $\mathbf{d}_1 \overset{\phi}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$, where $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_2)$. This and Lemma D.0.20 yields $\mathbf{d}_1 = st2db(s_1)$ and $\mathbf{d}_2 = st2db(s_2)$. By the soundness of the Premise inference rule, $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \models \phi$.

Inductive step: $N = k > 1$. Suppose that the claim of the proposition holds for all `state`-terms s_1, s_2 such that the statement $Execute(\alpha, s_1, s_2)$ is derivable in less than k derivation steps using the axioms in $\Gamma(\mathbf{P}, \mathcal{S})$.

The statement $Execute(\alpha, s_1, s_2)$ can be derived only via one of the following axioms: Unfolding, Premises, Sequencing, Query and Hypothetical. We consider each case in turn.

***Unfolding*:** Suppose $Execute(\phi, s_1, s_2)$ was derived via a ground instance of the Unfolding rule in $\Gamma(\mathbf{P}, \mathcal{S})$:

$$Execute(\phi, s_1, s_2) \leftarrow Execute(\psi, s_1, s_2)$$

This implies the following:

- $\phi \leftarrow \psi$ is a ground instance of an implication in $\mathbf{P}$
- $Execute(\psi, s_1, s_2) \in \mathbf{M}$, and it has been derived before $Execute(\phi, s_1, s_2)$, i.e., using less than k steps.

By the inductive hypothesis,

- $st2db(s_1)$ and $st2db(s_2)$ are defined and
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \psi$, where $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d}_2 = st2db(s_2)$

From this and the definition of implication in TR, it follows that

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \phi$$

Sequencing: Suppose that $\phi = \beta \otimes \gamma$, and $Execute(\phi, s_1, s_2)$ was derived via a ground instance of the sequencing rule in $\Gamma(\mathbf{P}, \mathcal{S})$:

$$Execute(\beta \otimes \gamma, s_1, s_2) \leftarrow Execute(\beta, s_1, s), Execute(\gamma, s, s_2)$$

This means that $Execute(\beta, s_1, s)$ and $Execute(\gamma, s, s_2)$ have already been derived in a number of steps smaller than k.

By the inductive hypothesis, it follows that $st2db(s_1)$, $st2db(s)$ and $st2db(s_2)$ are defined and

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d} \models \beta$$
$$\mathbf{P}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_2 \models \gamma$$

where $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d} = st2db(s)$ and $\mathbf{d}_2 = st2db(s_2)$. This and the definition of the serial conjunction $\otimes$ in TR imply that

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \phi$$

Decomposition: Suppose that ϕ is a fluent or an hypothetical, and $Execute(\phi, s_1, s_2)$ was derived via a ground instance of the decomposition rule in $\Gamma(\mathbf{P}, \mathcal{S})$:

$$Execute(\phi, s_1, s_1) \leftarrow Execute(g, s_1, s_1)$$

where $s_1 = s_2$. This implies the following:

- g is a conjunction of fluents and hypotheticals.

- $Execute(g, s_1, s_1) \in \mathbf{M}$, and it has been derived before $Execute(\phi, s_1, s_2)$, i.e., using less than k steps.

By the inductive hypothesis,

- $st2db(s_1)$ is defined and
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models g$, where $\mathbf{d}_1 = st2db(s_1)$.

From this and the definition of serial conjunction in TR, it follows that

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \phi$$

***Hypothetical*:** Suppose $\phi = \Diamond\psi$, and $Execute(\phi, s_1, s_2)$ was derived via a ground instance of the hypothetical rule in $\Gamma(\mathbf{P}, \mathcal{S})$:

$$Execute(\Diamond\psi, s_1, s_1) \leftarrow Execute(\psi, s_1, s)$$

where $s_1 = s_2$. By the inductive hypothesis,

- $st2db(s_1)$ and $st2db(s)$ are defined and
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\, \mathbf{d} \models \psi$, where $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d} = st2db(s)$

Therefore, the definition of the hypothetical operator in TR yields

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \phi$$

***Query*:** Suppose $Execute(\phi, s_1, s_2)$ was derived via the Query rule of $\Gamma(\mathbf{P}, \mathcal{S})$:

$$Execute(\phi, s_1, s_1) \;\leftarrow\; Holds(\phi, s_1)$$

where $s_1 = s_2$, and $Holds(f, s_1)$ was derived in less than k steps. This implies that ϕ is a fluent. Observe that $Holds(\phi, s_1)$ can be derived only by the rules Premises and Forward Projection in $\Gamma(\mathbf{P}, \mathcal{S})$. If $Holds(\phi, s_1)$ was derived via the rule Premises, it means that $Holds(\phi, s_1)$ is a fact in $\Gamma(\mathbf{P}, \mathcal{S})$ and the claim follows from Lemma D.0.21.

Suppose that $Holds(\phi, s_1)$ was derived via the forward projection rule. This means that ϕ is a primitive pre-effect or a primitive effect of some PAD α. The proofs for pre-effects and effects are similar, so we assume that ϕ is a primitive pre-effect of α. It follows that there is a PAD of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ in $\mathbf{P}$, ϕ is a conjunct in b_3, and $Holds(f, s_1)$ was derived by the forward projection rule in $\Gamma(\mathbf{P}, \mathcal{S})$ associated with this PAD:

$$\begin{aligned} Holds(b_3, s_1) \leftarrow \quad & Execute(\alpha, s_1, Result(\alpha, s_1)), \\ & Holds(b_1, s_1), Holds(b_2, Result(\alpha, s_1)) \end{aligned}$$

Recall from Section 3.6 that the above rules where fluents like b_3 can be complex formulas are just shortcuts for sets of rules that are obtained by distributing $\wedge$ through $Holds$ and eliminating conjunction in rule heads and disjunctions in rule bodies. Since ϕ is a conjunct in b_3, one of the rules obtained in this way will be

$$\begin{aligned} Holds(\phi, s_1) \leftarrow \quad & Execute(\alpha, s_1, Result(\alpha, s_1)), \\ & Holds(b_1, s_1), Holds(b_2, Result(\alpha, s_1)) \end{aligned}$$

That is, $Holds(\phi, s_1)$ was derived in less than k steps, Since $Execute(\alpha, s_1, Result(\alpha, s_1))$ is in the body of the above rule, it was also derived in less than k steps. By the inductive hypothesis,

- $st2db(s_1)$ and $st2db(Result(s_1))$ are defined and
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d} \models \alpha$, where $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d} = st2db(Result(s_1))$

The claim now follows from Lemma D.0.22. □

D.0.24. Theorem (Soundness). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$ where s_1 and s_2 are ground* `state`*-terms. Then there exist relational database states $\mathbf{d}_1, \ldots, \mathbf{d}_2$ (in $\mathcal{L}_{TR}$) such that the following holds:*

1. $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \models \alpha$
2. $\mathbf{d}_1 = st2db(s_1)$, $\mathbf{d}_2 = st2db(s_2)$
3. $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathbf{d}(s_1)$ *and* $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models \mathbf{d}(s_2)$

Proof. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$. Claims 1 and 2 follow directly from Proposition D.0.23. Claim 3 follows from Claim 2 and Lemma D.0.22. □

D.0.25. Theorem (Completeness). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$. Then the following holds:*

- *If $n = 1$, and there is a* `state`*-term s_1 such that $db2st_{\mathcal{S}}(\mathbf{d}_1) = s_1$, then ϕ is a conjunction of fluents and hypotheticals and:*

$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_1)$$

- *If $n > 1$ and there are ground* `state`*-terms s_1, s_2 such that $db2st_{\mathcal{S}}(\mathbf{d}_1) = s_1$ and $db2st_{\mathcal{S}}(\mathbf{d}_n) = s_2$, then*

$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$$

Proof. The proof relies on the fact that $\mathcal{TR}^{PAD}$ has a sound and complete proof theory developed in Section 3.3. We will prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \vdash \phi \tag{D.1}$$

using the inference system $\mathcal{F}$ from Section 3.3. We will be referring to the inference rules using the same enumeration than the one used in Definition A.0.1.

Base case: $N = 1$. In that case, (D.1) can possibly be derived by the No-op axiom or the premise inference rule. (Note that no other inference rules (even the hypothetical rule) can be used to derive a sequent in the first inference step.) We consider each case in turn.

- Suppose (D.1) was derived using the no-op axiom $\mathbf{P}, \mathbf{d}_1 \vdash ()$. Then the following must be true:
  - the sequent (D.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash ()$
  - ϕ is a the empty serial conjunction ()
  - $db2st_{\mathcal{S}}(\mathbf{d}_1)$ is defined

  By the construction of $\Gamma(\mathbf{P}, \mathcal{S})$, $Holds((), s_1) \in \Gamma(\mathbf{P}, \mathcal{S})$ for any $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$. The claim now follows from this by instantiating the Query rule $Execute(F, S, S) \leftarrow Holds(F, S)$ in $\Gamma(\mathbf{P}, \mathcal{S})$.

- If (D.1) was derived via the premise inference rule of $\mathcal{F}$, it could be derived only using a `state`-premise or a `run`-premise in $\mathcal{S}$:
  - Suppose (D.1) was derived using a *state*-premise $\mathbf{d}_1 \rhd \phi \in \mathcal{S}$. Then the following statements are true:
    - the sequent (D.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \phi$
    - ϕ is a fluent literal
    - $db2st_{\mathcal{S}}(\mathbf{d}_1)$ is defined

    By the construction of $\Gamma(\mathbf{P}, \mathcal{S})$, $Holds(\phi, s_1) \in \Gamma(\mathbf{P}, \mathcal{S})$ for any $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$. The claim now follows from this by instantiating the Query rule $Execute(F, S, S) \leftarrow Holds(F, S)$ in $\Gamma(\mathbf{P}, \mathcal{S})$.
  - Suppose (D.1) was derived using a *run*-premise $\mathbf{d}_1 \overset{\phi}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$ and (D.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \phi$. From the definition of $db2st_{\mathcal{S}}$ it follows that $db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $db2st_{\mathcal{S}}(\mathbf{d}_2)$ are defined and if $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$, then $Result(\phi, s_1) = db2st_{\mathcal{S}}(\mathbf{d}_2)$. By the construction of $\Gamma(\mathbf{P}, \mathcal{S})$, the above *run*-premise in $\mathcal{S}$ ensures that $Execute(\phi, s_1, Result(\phi, s_1))$ is in $\Gamma(\mathbf{P}, \mathcal{S})$, which proves our claim.

Inductive step: $N = k > 1$. Assume that whenever there are ground `state`-terms s_1, s_2 such that $s1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_n)$, and (D.1) can be derived by the proof theory in less than k steps then $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$.

To prove that the same holds also when (D.1) is derived using k steps, note that the last step in the derivation must be an application of one of these rules in $\mathcal{F}$:

- A Horn inference rule.
- The Forward Projection rule.
- The Sequencing rule.
- The Decomposition rule.

We consider each of these cases in turn.

- A Horn inference rule: 1a or 1b.
  - Rule 1a. The sequent (D.1) was derived because $\phi = a \otimes \psi$, $a \leftarrow \eta \in \mathbf{P}$, and $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 ... \mathbf{d}_2 \vdash \eta \otimes \psi$ was derived previously in less than k steps. Suppose that $db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $db2st_{\mathcal{S}}(\mathbf{d}_2)$ are defined. By the inductive assumption, there are states $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_2)$, such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\eta \otimes \psi, s_1, s_2)$. By the Sequencing rule in $\Gamma(\mathbf{P}, \mathcal{S})$, it follows that there is some `state`-term s such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\eta, s_1, s)$ and $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\psi, s, s_2)$. But then, by the Unfolding rule in $\Gamma(\mathbf{P}, \mathcal{S})$, we can derive $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(a, s_1, s)$. Finally, using the Sequencing rule again, we derive $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(a \otimes \psi, s_1, s_2)$.
  - Rule 1b: The sequent (D.1) was derived because $\phi = \Diamond\beta \otimes \gamma$, and both $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 ... \mathbf{d}' \vdash \beta$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 ... \mathbf{d}_2 \vdash \gamma$ were derived previously in less than k steps. Suppose that $db2st_{\mathcal{S}}(\mathbf{d}_1)$,$db2st_{\mathcal{S}}(\mathbf{d}_2)$, and $db2st_{\mathcal{S}}(\mathbf{d}')$ are defined. By the inductive assumption, there are states $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$, $s' = db2st_{\mathcal{S}}(\mathbf{d}')$ and $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_2)$, such that $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\beta, s_1, s')$, and $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\gamma, s_1, s_2)$. This and the rules for hypotheticals and sequencing in $\Gamma(\mathbf{P}, \mathcal{S})$ yield $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$.

- The Forward Projection rule. Suppose (D.1) was derived because there is a PAD $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \in \mathbf{P}$, where ϕ is b_3 or b_4, and

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1$$
$$\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \vdash b_2$$
$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \mathbf{d}_2 \vdash \alpha$$

were derived previously in less than k steps. Suppose that $db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $db2st_{\mathcal{S}}(\mathbf{d}_2)$ are non empty. The inductive assumption ensures that there are states $s_1 = \mathbf{d}_1$ and $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_2)$ such that

$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$$
$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(b_1, s_1, s_1)$$
$$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(b_2, s_2, s_2)$$

Now the inductive claim follows from these facts and the rules for forward projection, querying, and sequencing in $\Gamma(\mathbf{P}, \mathcal{S})$.

- The Sequencing rule. Suppose the sequent (D.1) was derived because

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d} \vdash \psi$$
$$\mathbf{P}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_2 \vdash \eta$$

were derived previously, in less than k steps and $\phi = \psi \otimes \eta$. Suppose that $db2st_{\mathcal{S}}(\mathbf{d}_1)$, $db2st_{\mathcal{S}}(\mathbf{d})$, and $db2st_{\mathcal{S}}(\mathbf{d}_2)$ are defined. By the inductive assumption, there are states $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$ and $s = db2st_{\mathcal{S}}(\mathbf{d})$ such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\psi, s_1, s)$ and there is a state $s_2 = db2st_{\mathcal{S}}(\mathbf{d}_2)$ such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\psi, s, s_2)$. The inductive claim now follows from this and the sequencing rule in $\Gamma(\mathbf{P}, \mathcal{S})$.

- The Decomposition rule. Suppose the sequent (D.1) was derived because either $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \phi \otimes \eta$ or $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \eta \otimes \phi$ was derived previously in less than k steps. By the inductive assumption, there is a state $s_1 = db2st_{\mathcal{S}}(\mathbf{d}_1)$, we have $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi \otimes \eta, s_1, s_1)$ or $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\eta \otimes \phi, s_1, s_1)$. The inductive claim now follows from this and the Decomposition rule in $\Gamma(\mathbf{P}, \mathcal{S})$. This concludes the proof of completeness. □

Appendix E

Proof for the Well-founded Semantics of $\mathcal{TR}^{PAD}$

In this chapter we prove that $\mathcal{TR}^{PAD}$ specifications have a unique minimal model. For simplicity we present a ground version of the proofs. Lifting to the non-ground case is done in a standard way (cf. [11]).

We assume that the transaction base is a set of serial Horn rules, the action base is a set of PADs, and that the set of premises are *state*- and *run*-premises.

First we prove that $\mathcal{TR}^{PAD}$ specifications which do not contain neither **not** nor the special proposition $\mathbf{u}^{\pi}$ have a least (2-valued) model, and it can be constructed iteratively in a bottom-up fashion. Afterwards, we prove that general $\mathcal{TR}^{PAD}$ specifications have a least (3-valued) partial model.

Least models for $\{\mathbf{not}, \mathbf{u}^{\pi}\}$-free $\mathcal{TR}^{PAD}$ Specifications

In this section we reduce $\{\mathbf{not}, \mathbf{u}^{\pi}\}$-free $\mathcal{TR}^{PAD}$ specifications to $\{\mathbf{not}, \mathbf{u}^{\pi}\}$-free Horn-$TR$ programs.

$\mathcal{TR}$ is parametrized by data and transition *oracles*, denoted $\mathcal{O}^D$ and $\mathcal{O}^T$ respectively. The data oracles specifies a set of primitive database *queries*, i.e., the static semantics of states; and the transition oracle specifies a set of primitive database *updates*, i.e., the dynamic semantics of states. $\mathcal{TR}^{PAD}$, does not use *any* oracles—data or transition— but the semantics of Horn-TR programs. Details can be found in [11].

Recall that the symbols $\mathbf{u}^{\pi}$ are special propositions that are unknown in π and false in any other path. With this reduction we prove that $\{\mathbf{not}, \mathbf{u}^{\pi}\}$-free $\mathcal{TR}^{PAD}$ specification have a least (2-valued) model (LM). Recall that $\{\mathbf{not}, \mathbf{u}^{\pi}\}$-free Horn-$TR$ programs have a unique minimal model [11].

In the remainder, given a serial conjunction of fluents b and a data oracle $\mathcal{O}^D$, will slightly abuse the notation and write $b \in \mathcal{O}^D$ meaning that for every conjunct (fluent) f in b, we have that $f \in \mathcal{O}^D$.

Next, in Definitions E.0.26 ,E.0.27, and E.0.28, we iteratively build data and transition oracles that encode the information in the premises and PADs.

E.0.26. Definition. [Oracalization] Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification, $\mathcal{O}^D$ and $\mathcal{O}^T$ data and transition oracles respectively, and $\mathbf{M}$ the LPM of $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. By the **oracalization** of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ given $(\mathcal{O}^D, \mathcal{O}^T)$, written as $\frac{(\mathcal{O}^D, \mathcal{O}^T)}{(\mathcal{E}, \mathbf{P}, \mathcal{S})}$, we mean the new oracles $\mathcal{O}^D_{(\mathcal{S},\mathcal{E})}$ and $\mathcal{O}^T_{(\mathcal{S},\mathcal{E})}$ obtained from $(\mathcal{O}^D, \mathcal{O}^T)$ as follows:

1. For every path π and literal h, if $h \in \mathcal{O}^D(\pi)$ then $h \in \mathcal{O}^D_{(\mathcal{S},\mathcal{E})}(\pi)$
2. For every path π and literal h, if $h \in \mathcal{O}^T(\pi)$ then $h \in \mathcal{O}^T_{(\mathcal{S},\mathcal{E})}(\pi)$
3. For every state-premise $\mathbf{d} \rhd f \in \mathcal{S}$, we have that $f \in \mathcal{O}^D_{(\mathcal{S},\mathcal{E})}(\langle \mathbf{d} \rangle)$
4. For every run-premise $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$, we have that $\alpha \in \mathcal{O}^T_{(\mathcal{S},\mathcal{E})}(\langle \mathbf{d}_1, \mathbf{d}_2 \rangle)$
5. For every path of the form $\langle \mathbf{d}_1 \mathbf{d}_2 \rangle$ and PAD in $\mathcal{E}$ of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ such that:
$$\mathbf{M}(\langle \mathbf{d}_1 \rangle) \models b_1$$
$$\mathbf{M}(\langle \mathbf{d}_2 \rangle) \models b_2$$
$$\mathbf{M}(\langle \mathbf{d}_1 \mathbf{d}_2 \rangle) \models \alpha$$
then the next two statements hold
$$b_3 \in \mathcal{O}^D_{(\mathcal{S},\mathcal{E})}(\langle \mathbf{d}_1 \rangle)$$
$$b_4 \in \mathcal{O}^D_{(\mathcal{S},\mathcal{E})}(\langle \mathbf{d}_2 \rangle)$$

□

E.0.27. Definition. [Oracalizer Operator] The incremental **oracalizer** operator, Υ, for a specifications $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ takes a set of data and transition oracles $(\mathcal{O}^D, \mathcal{O}^T)$ and returns a new set of oracles as follows:

$$\Upsilon((\mathcal{O}^D, \mathcal{O}^T)) = \frac{(\mathcal{O}^D, \mathcal{O}^T)}{(\mathcal{E}, \mathbf{P}, \mathcal{S})}$$

Let $(\mathcal{O}^D_0, \mathcal{O}^T_0)$ be oracles that maps each path π to the empty set. The ordinal powers of the immediate consequence operator Υ are defined inductively as follows:

- $\Upsilon^{\uparrow 0}((\mathcal{O}^D_0, \mathcal{O}^T_0)) = (\mathcal{O}^D_0, \mathcal{O}^T_0)$

- $\Upsilon^{\uparrow n}((\mathcal{O}_0^D, \mathcal{O}_0^T)) = \Upsilon(\Upsilon^{\uparrow n-1}((\mathcal{O}_0^D, \mathcal{O}_0^T))$, for n a successor ordinal
- $\Upsilon^{\uparrow n}(\mathbf{I}_0)(\pi) = \bigcup_{j \leq n} \Upsilon^{\uparrow j}(\mathbf{I}_0)$, if n is a limit ordinal □

The operator Υ is monotonic with respect to the standard inclusion relation, $\subseteq$, when $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is fixed. Because of this, the sequence $\{\Upsilon^{\uparrow n}((\mathcal{O}_0^D, \mathcal{O}_0^T))\}$ has a least fixed point and is computable via transfinite induction.

E.0.28. Definition. [$\mathcal{TR}^{PAD}$ compliant Oracles] The $\mathcal{TR}^{PAD}$ **compliant** oracles of a specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is defined as the limit of the sequence $\{\Upsilon^{\uparrow n}((\mathcal{O}_0^D, \mathcal{O}_0^T))\}$. □

The following technical results are needed to prove that every $\{\mathbf{not}, \mathbf{u}^\pi\}$-free $\mathcal{TR}^{PAD}$ program has a minimal model (c.f. Theorem E.0.32).

E.0.29. Lemma. *Let $(\mathcal{O}^D, \mathcal{O}^T)$ be the $\mathcal{TR}^{PAD}$ compliant oracles of the $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Then the Horn program $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$ has a least Herbrand model.*

Proof. Follows from the fact that every $\{\mathbf{not}, \mathbf{u}^\pi\}$-free Horn program have a least model [11]. □

E.0.30. Lemma. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Let $\mathcal{O}^D$ and $\mathcal{O}^T$ be the $\mathcal{TR}^{PAD}$ compliant data and transition oracles of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Let $\mathcal{M}$ the a model of the Horn program $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. Then $\mathcal{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$.*

Proof. By hypothesis we know that $\mathcal{M} \models \mathbf{P}$. By construction of $\mathcal{O}^D$ and $\mathcal{O}^T$, we know that

- For every state-premise $\mathbf{d} \triangleright f \in \mathcal{S}$, we know that $f \in \mathcal{O}^D(\langle \mathbf{d} \rangle)$ and therefore $\mathcal{M}(\langle \mathbf{d} \rangle)(f) = \mathbf{t}$
- For every run-premise $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$, we know that $\alpha \in \mathcal{O}^T(\langle \mathbf{d}_1, \mathbf{d}_2 \rangle)$ and therefore $\mathcal{M}(\langle \mathbf{d}_1, \mathbf{d}_2 \rangle)(\alpha) = \mathbf{t}$

Thus, we can conclude that $\mathcal{M}$ is a model of $\mathcal{S}$.

In addition, we know that for every path of the form $\langle \mathbf{d_1 d}_2 \rangle$ and PAD in $\mathcal{E}$ of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ such that:

$$\begin{array}{l} \mathcal{M}(\langle \mathbf{d}_1 \rangle) \models b_1 \\ \mathcal{M}(\langle \mathbf{d}_2 \rangle) \models b_2 \\ \mathcal{M}(\langle \mathbf{d}_1, \mathbf{d}_2 \rangle) \models \alpha \end{array}$$

it follows that

$$\begin{array}{l} b_3 \in \mathcal{O}^D(\langle \mathbf{d}_1 \rangle) \\ b_4 \in \mathcal{O}^D(\langle \mathbf{d}_2 \rangle) \end{array}$$

and therefore

$$\mathcal{M}(\langle \mathbf{d}_1 \rangle) \models b_3$$
$$\mathcal{M}(\langle \mathbf{d}_2 \rangle) \models b_4$$

This yields that $\mathcal{M}$ is a model of $\mathcal{E}$. This concludes the proof of the lemma. □

E.0.31. LEMMA. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Let $\mathcal{O}^D$ and $\mathcal{O}^T$ be the $\mathcal{TR}^{PAD}$ compliant data and transition oracles of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Let $\mathcal{M}$ be a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Then $\mathcal{M}$ is a model of $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$.*

Proof. Since $\mathcal{M}$ is a model of $\mathcal{E}, \mathcal{S}$, by construction of the oracles it follows that $\mathcal{M}$ is a model of $\mathcal{O}^D, \mathcal{O}^T$. By hypothesis, $\mathcal{M} \models \mathbf{P}$. Thus $\mathcal{M}$ is a model of $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. □

E.0.32. THEOREM. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Let $\mathcal{O}^D$ and $\mathcal{O}^T$ be the $\mathcal{TR}^{PAD}$ compliant data and transition oracles of $\mathcal{S}$ and $\mathcal{E}$. Let $\mathcal{M}$ be the least Herbrand model of the Horn program $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. Let $\mathcal{N}$ be a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Then*

$$\mathcal{M} \preceq \mathcal{N}$$

Proof. Let $\mathcal{M}$ be the least Herbrand model of the Horn program $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. By Lemma E.0.30, $\mathcal{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Suppose that it is not a minimal model. Thus, there is a model $\mathbf{M}$ of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, such that

$$\mathbf{M} \preceq \mathcal{M}$$

By Lemma E.0.31, $\mathbf{M}$ is a model of $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$, and moreover, it is smaller than $\mathcal{M}$. But this is absurd since $\mathcal{M}$ is the minimal model of $(\mathcal{O}^D, \mathcal{O}^T, \mathbf{P})$. This implies that $\mathcal{M}$ is the minimal model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. □

E.0.33. COROLLARY. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\{\mathbf{not}, \boldsymbol{u}^{\pi}\}$-free $\mathcal{TR}^{PAD}$ specification. Then $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ has a LPM.*

Bottom-up Construction of 2-valued Least Models in Horn-*TR*

In this section we prove that the least model (LM) of **not**-free serial-Horn-programs do not contain the special proposition $\mathbf{u}^{\pi}$ and can be built up in a bottom-up fashion. The least model is defined with respect to the truth ordering $\preceq$. It is worth noting that in the 2-valued universe models are defined as sets of positive atoms. So, $\preceq$ would correspond to subsets of positive atoms.

To avoid tedious repetitions, In this sections we assume that serial-Horn programs in this sections are **not** -free and do not contain the special proposition $\mathbf{u}^{\pi}$.

E.0.34. Definition. [Definite Consequence Operator] Let $\mathbf{I}$ be a Herbrand Path structure. The **consequence operator** , Ξ, for a serial-Horn program $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$, is defined as

$$\Xi(\mathbf{I}) = \mathbf{J}$$

where

1. $\mathbf{J}$ is a Herbrand model (i.e. 2-value model) of $(\mathcal{O}^d, \mathcal{O}^t)$
2. for every path π and literal h such that $\mathbf{I}(\pi)(h) = \mathbf{t}$, we have that $\mathbf{J}(\pi)(h) = \mathbf{t}$
3. for every path π and literal h such that $\mathbf{I}(\pi)(h) = \mathbf{f}$, we have that $\mathbf{J}(\pi)(h) = \mathbf{f}$
4. for every path π, and rule in $\mathbf{P}$ of the form $h \leftarrow G$,

$$\text{if} \quad \mathbf{I}, \pi \models G \quad \text{then} \quad \mathbf{J}, \pi \models h$$

The **ordinal powers** of the consequence operator Ξ are then defined inductively as follows:

- $\Xi^{\uparrow 0}(\mathbf{I}_0) = \mathbf{I}_0$
- $\Xi^{\uparrow n}(\mathbf{I}_0) = \Xi(\Xi^{\uparrow n-1}(\mathbf{I}_0))$, if n is a successor ordinal
- $\Xi^{\uparrow n}(\mathbf{I}_0) = \bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0)$, if n is a limit ordinal □

The operator Ξ is clearly monotonic with respect to the $\leq$ order when $\mathbf{P}$ is fixed. Because of this, the sequence $\{\mathbf{P}^{\uparrow n}(\mathbf{I})\}$ has a least fixed point and is computable via transfinite induction. In the reminder, let $\mathbf{I}_{\emptyset}$ be a Herbrand path structure such that for every literal h and path π

$$\mathbf{I}_{\emptyset}, \pi \not\models h$$

E.0.35. Lemma. $\mathbf{M}$ *is a model of* $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$ *iff* $\Xi(\mathbf{M}) \preceq \mathbf{M}$.

Proof. The proof follows the same outline than the proof of Lemma E.0.44. □

E.0.36. Corollary. *Let* α *be the limit of the consequence operator* Ξ. *Then* $\Xi^{\uparrow \alpha}(\mathbf{I}_0)$ *is a model of* $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$.

E.0.37. Lemma (Minimal Model). *The least model of a serial-Horn program* $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$ *coincides with the limit of the sequence* $\{\Xi^{\uparrow n}(\mathbf{I}_0)\}$. □

Proof. Let $\mathbf{J}$ be the limit of the sequence $\{\Xi^{\uparrow n}(\mathbf{I}_0)\}$, and $\mathbf{M}$ the least model of $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$. We know that such model exists from [31]. By Lemma E.0.36 $\mathbf{J}$ is a model of $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$. We need to proof that it is minimal with respect to $\preceq$. Thus, we will prove by transfinite induction on the number of steps needed to construct $\mathbf{J}$ that

$$\mathbf{J} \preceq \mathbf{M}$$

Base Case: Follows immediately since $\mathbf{I}_0$ is the empty interpretation.

Inductive Claim: For every $k < n$,

$$\Xi^{\uparrow n}(\mathbf{I}_0) \preceq \mathbf{M}$$

Inductive Case (Successor ordinal): Let $\Xi^{\uparrow n}(\mathbf{I}_0) = \mathbf{J}^n$. Suppose $\mathbf{J}^n, \pi \models h$. Then either that data or transaction oracles enforce h to be true on π, or h was derived by a rule. In the first case, since $\mathbf{M}$ is a model of the oracles, it trivially follows that

$$\mathbf{M}, \pi \models h \tag{E.1}$$

Now, suppose $\mathbf{J}^n, \pi \models h$ was derived by a ground instance of rule of the form

$$h \leftarrow G$$

where G is a serial conjunction. This means that $\Xi^{\uparrow n-1}(\mathbf{I}_0), \pi \models G$ was derived in at most $n-1$ steps. Thus, by the inductive assumption

$$\mathbf{M}(\pi)(G) = \mathbf{t}$$

and since $\mathbf{M}$ is a model of $(\mathcal{O}^d, \mathcal{O}^t, \mathbf{P})$ it follows that

$$\mathbf{M}, \pi \models h \tag{E.2}$$

From (E.2) and (E.1), we can conclude that for every n

$$\Xi^{\uparrow n}(\mathbf{I}_0) \preceq \mathbf{M}$$

Inductive Case (limit ordinal): If n is a limit ordinal, then

$$\Xi^{\uparrow n}(\mathbf{I}_0) = \bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0)$$

by inductive hypothesis we know that for every $j < n$

$$\Xi^{\uparrow j}(\mathbf{I}_0) \preceq \mathbf{M}$$

Therefore all **not**-free literals that are true in $\Xi^{\uparrow j}(\mathbf{I}_0)$ (for any $j < n$) are true in $\mathbf{M}$ and all **not**-literals that are true in $\Xi^{\uparrow j}(\mathbf{I}_0)$ are true in $\mathbf{M}$. It follows that

$$\bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0) \preceq \mathbf{M}$$

thus

$$\Xi^{\uparrow n}(\mathbf{I}_0) \preceq \mathbf{M}$$

□

Least Models for $\mathcal{TR}^{PAD}$ Specifications

In this section we prove that $\mathcal{TR}^{PAD}$ specifications have a least partial model.

E.0.38. Definition. Given a **not**-free $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, let $(\mathcal{E}, \mathbf{P}, \mathcal{S})^+$ denote the positive specification obtained from $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ by replacing all the postconditions literals and body literals of the from $\mathbf{u}^\pi$, with $\mathbf{t}^\pi$. Analogously, let $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$ denote the positive program obtained from $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ by removing all the PADs and rules whose body includes $\mathbf{u}^\pi$. □

E.0.39. Theorem (Unique LPM for **not**-free specifications).
If $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is a **not***-free specification, then $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ has a* Least Herbrand Model*, denoted LPM($(\mathcal{E}, \mathbf{P}, \mathcal{S})$).*

Proof.
Let

- $\mathcal{M}^-$ be the least (2-valued) model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$
- $\mathcal{M}^+$ be the least (2-valued) model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^+$

We know that such models exist by Corollary E.0.33. Clearly $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$ is a sub-specification of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^+$, thus $\mathcal{M}^+$ is also a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$. But since $\mathcal{M}^-$ is the LM of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$, we can conclude that

$$\mathcal{M}^- \preceq \mathcal{M}^+ \tag{E.3}$$

Recall that this implies that all **not**-free literals that are true in $\mathcal{M}^-$ and also true in $\mathcal{M}^+$. We construct the least model $\mathcal{M}$ of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ as the path structure such that any path π, and **not**-free literal h

$$\begin{array}{lll} \mathcal{M}(\pi)(h) = \mathbf{t} & \text{iff} & \mathcal{M}^-(\pi) \models h \\ \mathcal{M}(\pi)(h) = \mathbf{f} & \text{iff} & \mathcal{M}^+(\pi) \not\models h \\ \mathcal{M}(\pi)(h) = \mathbf{u} & \text{iff} & \text{otherwise} \end{array} \tag{E.4}$$

We now prove that $\mathcal{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. First, it is clear that $\mathcal{M}$ is well defined, since by E.3 it cannot be that for some path π, $\mathcal{M}^-(\pi) \models h$ and $\mathcal{M}^+(\pi) \not\models h$.

We proceed to show that $\mathcal{M}$ is a model of $\mathcal{S}, \mathcal{E}$ and $\mathbf{P}$

- Since $\mathcal{M}^-$ is a model of $\mathcal{S}$ it follows that

  if $\pi \rhd h \in \mathcal{S}$ then $\mathcal{M}^-(\pi) \models h$ and therefore $\mathcal{M}(\pi)(h) = \mathbf{t}$
  if $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$ then $\mathcal{M}^-(\langle \mathbf{d}_1 \mathbf{d}_2 \rangle) \models h$ and therefore $\mathcal{M}(\langle \mathbf{d}_1 \mathbf{d}_2 \rangle)(h) = \mathbf{t}$

  Thus, $\mathcal{M}$ is a model of $\mathcal{S}$

- Let $\lambda = b_1 \otimes \alpha \otimes b_2 \to b_3 \otimes \alpha \otimes b_4$ be a PAD in $\mathcal{E}$. We consider the cases where λ is $\mathbf{u}$-free and where it is not.

  - Suppose that λ is $\mathbf{u}$-free. Then λ is in both $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$ and $(\mathcal{E}, \mathbf{P}, \mathcal{S})^+$. Suppose that there is some path π, $\mathcal{M}(\pi) \models b_1 \otimes \alpha \otimes b_2$. It follows that $\mathcal{M}^-(\pi) \models b_1 \otimes \alpha \otimes b_2$. Since $\mathcal{M}^-$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^-$, it follows that $\mathcal{M}^-(\pi) \models b_3 \otimes \alpha \otimes b_4$, and therefore $\mathcal{M}(\pi) \models b_3 \otimes \alpha \otimes b_4$. This implies that $\mathcal{M}(\pi) \models \lambda$.

    Suppose that for some path π, $\mathcal{M}(\pi) \not\models b_1 \otimes \alpha \otimes b_2$, then trivially $\mathcal{M}(\pi) \models \lambda$.

    Suppose that for some path π, $\mathcal{M}(\pi)(b_1 \otimes \alpha \otimes b_2) = \mathbf{u}$. It follows that π has the form $\mathbf{d}_1 \mathbf{d}_2$, and there a (at least one) conjunct b in b_1 (or b_2) such that $\mathcal{M}(\mathbf{d}_1)(b) = \mathbf{u}$ ($\mathcal{M}(\mathbf{d}_2)(b) = \mathbf{u}$ respectively). For concreteness and simplicity suppose that there is only one b such that $\mathcal{M}(\mathbf{d}_1)(b) = \mathbf{u}$, and b is in b_1. Therefore, for every conjunct $b' \neq b$ in b_1 (or b_2), we have that $\mathcal{M}(\mathbf{d}_1)(b') = \mathbf{t}$ ($\mathcal{M}(\mathbf{d}_2)(b') = \mathbf{t}$ respectively). Thus, we can conclude that $\mathcal{M}^+(\mathbf{d}_1)(b) = \mathbf{t}$, and moreover $\mathcal{M}^+(\pi) \models b_1 \otimes \alpha \otimes b_2$. Since $\mathcal{M}^+$ is a model of $\mathcal{E}$, it follows that $\mathcal{M}^+(\pi) \models b_3 \otimes \alpha \otimes b_4$. And from E.4 we can conclude that $\mathcal{M}(\pi)(b_1 \otimes \alpha \otimes b_2) \geq \mathbf{u}$. This yields that $\mathcal{M}(\pi) \models \lambda$.

  - Now suppose that λ is not $\mathbf{u}$-free. Thus then λ is in $(\mathcal{E}, \mathbf{P}, \mathcal{S})^+ \setminus (\mathcal{E}, \mathbf{P}, \mathcal{S})^-$. This implies that there is some conjunct b in b_1 or b_2 equal to $\mathbf{u}^\pi$. If $\mathcal{M}(\pi) \not\models b_1 \otimes \alpha \otimes b_2$, again, $\mathcal{M}(\pi) \models \lambda$ trivially. Clearly, it is not possible that $\mathcal{M}(\pi) \models b_1 \otimes \alpha \otimes b_2$. Suppose $\mathcal{M}(\pi)(b_1 \otimes \alpha \otimes b_2) = \mathbf{u}$. By E.4 we can conclude that $\mathcal{M}^+(\pi)(b_1 \otimes \alpha \otimes b_2) = \mathbf{t}$. Therefore, since $\mathcal{M}^+$ is a model of $\mathcal{E}$, it follows that $\mathcal{M}^+(\pi)(b_3 \otimes \alpha \otimes b_4) = \mathbf{t}$. This yields that $\mathcal{M}(\pi)(b_3 \otimes \alpha \otimes b_4) \geq \mathbf{u}$, which proves that $\mathcal{M}(\pi) \models \lambda$.

From the previous facts we can conclude that $\mathcal{M}$ is a model of $\mathcal{E}$,

- The proof that $\mathcal{M}$ is a model of $\mathbf{P}$ is analogous to the proof above.

The proof that $\mathcal{M}$ is minimal and unique, is analogous to the proof of Theorem 1 in [31] □

For convenient reference, we repeat the main definitions from Section 3.7.

E.0.40. Definition. [$\mathcal{TR}^{PAD}$-quotient] Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification, and $\mathbf{I}$ a Herbrand path structure. By **$\mathcal{TR}^{PAD}$-quotient** of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ **modulo I** we mean a new specification, $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{I}}$, which is obtained from $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ by

- Replacing every literal of the form $\mathbf{not}\, b$ in $\mathbf{P} \cup \mathcal{E}$ with

$$\begin{array}{l} \mathbf{t}^{\pi} \text{ for every path } \pi \text{ such that } \mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{t} \\ \mathbf{u}^{\pi} \text{ for every path } \pi \text{ such that } \mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{u} \end{array}$$

- And then removing all the remaining rules and PADs that have a literal of the form $\mathbf{not}\, b$ in the body such that $\mathbf{I}(\pi)(\mathbf{not}\, b) = \mathbf{f}$ for some path π. □

E.0.41. Definition. [$\mathcal{TR}^{PAD}$ Immediate Consequence Operator] The consequence operator, Γ, for a $\mathcal{TR}^{PAD}$ specification is defined by analogy with the classical case:

$$\Gamma(\mathbf{I}) = \mathrm{LPM}(\frac{(\mathcal{E}, \mathbf{P}, \mathcal{S})}{\mathbf{I}})$$

Suppose $\mathbf{I}_{\emptyset}$ is the path structure that maps each path π to the empty Herbrand interpretation in which all atoms are undefined. That is, for every path π and literal f, we have $\mathbf{I}_{\emptyset}(\pi)(f) = \mathbf{u}$. The ordinal powers of the consequence operator Γ are then defined inductively as follows:

- $\Gamma^{\uparrow 0}(\mathbf{I}_0) = \mathbf{I}_0$
- $\Gamma^{\uparrow n}(\mathbf{I}_0) = \Gamma(\Gamma^{\uparrow n-1}(\mathbf{I}_0))$, if n is a successor ordinal
- $\Gamma^{\uparrow n}(\mathbf{I}_0)(\pi) = \bigcup_{j \le n} \Gamma^{\uparrow j}(\mathbf{I}_0)$, if n is a limit ordinal □

E.0.42. Definition. [Well-founded Model] The **well-founded** model of a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, written $\mathrm{WFM}((\mathcal{E}, \mathbf{P}, \mathcal{S}))$, is defined as a limit of the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_0)\}$. □

E.0.43. Lemma. *The operator Γ is monotonic with respect to the information order relation $\le$ when $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is fixed. That is,*

$$\textit{If} \quad \mathbf{I} \le \mathbf{I}' \quad \textit{then} \quad \Gamma(\mathbf{I}) \le \Gamma(\mathbf{I}')$$

Proof. The proof of this lemma is by transfinite induction on the iterations of the Γ operator.

Base Case (n=0): Follows from definition of $\Gamma^{\uparrow 0}$.

Inductive Case (n=k+1, successor ordinal): For any $m \leq n$, let $\mathbf{M}^m = \Gamma^{\uparrow m}(\mathbf{I})$, and $\mathbf{N}^m = \Gamma^{\uparrow m}(\mathbf{I}')$. We need to show that

$$\mathbf{M}^n \leq \mathbf{N}^n \tag{E.5}$$

Let

- $\mathcal{M}^{n-}$ be the model of $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^-$
- $\mathcal{M}^{n+}$ be the model of $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^+$
- $\mathcal{N}^{n-}$ be the model of $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^-$
- $\mathcal{N}^{n-}$ be the model of $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^+$

Observe that

$$\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^- \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^+ \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^- \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^+ \tag{E.6}$$

are $\{\mathbf{not}\,, \mathbf{u}^\pi\}$-free.[1] Thus, adapting the results about least models of **not**-free programs in [31], together with Corollary E.0.33 above, it follows that the least model $\mathbf{M}^n$ and $\mathbf{N}^n$ are defined as follows: For any path π, and **not**-free literal f

$$\begin{array}{lll} \mathcal{M}^n(\pi)(f) = \mathbf{t} & \text{iff} & \mathcal{M}^{n-}(\pi) \models f \\ \mathcal{M}^n(\pi)(f) = \mathbf{f} & \text{iff} & \mathcal{M}^{n+}(\pi) \not\models f \\ \mathcal{M}^n(\pi)(f) = \mathbf{u} & \text{iff} & \text{otherwise} \end{array}$$

$$\begin{array}{lll} \mathcal{N}^n(\pi)(f) = \mathbf{t} & \text{iff} & \mathcal{N}^{n-}(\pi) \models f \\ \mathcal{N}^n(\pi)(f) = \mathbf{f} & \text{iff} & \mathcal{N}^{n+}(\pi) \not\models f \\ \mathcal{N}^n(\pi)(f) = \mathbf{u} & \text{iff} & \text{otherwise} \end{array}$$

The previous fact implies that it is enough to show that for any path π:

$$\begin{array}{l} \mathcal{M}^{n-}(\pi) \subseteq \mathcal{N}^{n-}(\pi) \\ \mathcal{M}^{n+}(\pi) \supseteq \mathcal{N}^{n+}(\pi) \end{array} \tag{E.7}$$

[1] Recall that the models $\mathcal{M}^{n-}$, $\mathcal{M}^{n+}$, $\mathcal{N}^{n-}$, and $\mathcal{N}^{n+}$ are 2-valued.

From Lemmas E.0.30 and E.0.31, we can conclude that the $\mathcal{TR}^{PAD}$ specifications in (E.6) are equivalent to Horn-TR programs. This implies that these specification are monotonic, in the sense that for any two specification A and B, such that A is included in B, the models of A are also models of B. Thus, to prove (E.7) we can prove that

$$\begin{array}{l} \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{-} \subseteq \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^{-} \\ \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{+} \supseteq \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^{+} \end{array} \tag{E.8}$$

We explore each of these cases in turn:

- Recall that $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}$ does not contain any rule or PAD with a **not**-literal l in the body such that $\mathbf{M}^{n-1}(\pi)(l) = \mathbf{f}$. In addition, recall that the **not**-literals l in $(\mathcal{E},\mathbf{P},\mathcal{S})$ are replaced by $\mathbf{u}^{\pi}$ or $\mathbf{t}^{\pi}$ in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}$ depending on the truth value of $\mathbf{M}^{n-1}(\pi)(l)$.

  Let

  $$b'_1 \otimes \alpha \otimes b'_2 \rightarrow b'_3 \otimes \alpha \otimes b'_4 \in \frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}$$

  where each conjunct b' in b'_i is the result of the transformation after taking the quotient by $\mathbf{M}^{n-1}$. That is, if b is not a **not**-literal, $b' = b$, and if b is a **not**-literal then b' is equal to $\mathbf{t}^{\pi}$ or $\mathbf{u}^{\pi}$ depending on the truth value of $\mathbf{M}^{n-1}(\pi)(b)$, with π a path. Therefore, if such rule is in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{-}$ as well, it means then for every **not**-literal b' in the in the pre/post-condition of the PAD,[2] we have that $\mathbf{M}^{n-1}(\pi)(l) = \mathbf{t}^{\pi}$ for some path π. By the inductive assumption we have that if $\mathbf{M}^{n-1}(\pi)(l) = \mathbf{t}$ then $\mathbf{N}^{n-1}(\pi)(l) = \mathbf{t}$. From the previous fact we can conclude that if a PAD is in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{-}$, then it is also in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^{-}$. Following a similar reasoning, we can conclude that if a rule is in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}^{-}$, then it is also in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^{-}$. Thus,

  $$\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{-} \subseteq \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^{-}$$

- Following a similar reasoning as above, it is easy to show that a rule or a PAD is in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^{+}$ if for every **not**-literal l in the body of the rule, or in the pre/post-condition of the PAD, we have that $\mathbf{M}^{n-1}(\pi)(l) \geq \mathbf{u}$ for some path π.

[2] Recall that **not**-literals cannot occur in pre-effects or effects of PADs

By the inductive assumption we have that

$$\text{if } \mathbf{M}^{n-1}(\pi)(l_i) = \mathbf{t} \text{ then } \mathbf{N}^{n-1}(\pi)(l_i) = \mathbf{t}$$

Recall that $\mathbf{M}^{n-1}(\pi) \leq \mathbf{N}^{n-1}(\pi)$ if all **not**-free literals that are true in $\mathbf{M}^{n-1}(\pi)$ are true in $\mathbf{N}^{n-1}(\pi)$ and all **not**-literals that are true in $\mathbf{M}^{n-1}(\pi)$ are true in $\mathbf{N}^{n-1}(\pi)$. Thus, if $\mathbf{M}^{n-1}(\pi)(l_i) = \mathbf{u}$ then $\mathbf{N}^{n-1}(\pi)(l_i)$ can be either $\mathbf{f}, \mathbf{u}$ or $\mathbf{t}$. The cases where $\mathbf{N}^{n-1}(\pi)(l)$ is $\mathbf{u}$ or $\mathbf{t}$ does not affect the presence of the rules in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^+$. On the other hand, if $\mathbf{N}^{n-1}(\pi)(l) = \mathbf{f}$, whichever rule/PAD that has l in its body will not be in $\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^+$.

From the previous facts we can conclude that

$$\left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{M}^{n-1}}\right)^+ \supseteq \left(\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathcal{N}^{n-1}}\right)^+$$

The previous facts prove (E.8), therefore E.7 follows. This yields (E.5) which in turn implies the claim of the lemma for the successor ordinal.
Inductive Case (n, limit ordinal): If n is a limit ordinal, then

$$\Xi^{\uparrow n}(\mathbf{I}) = \bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I})$$

by inductive hypothesis we know that for every $j < n$

$$\Xi^{\uparrow j}(\mathbf{I}) \leq \Xi^{\uparrow j}(\mathbf{I}')$$

therefore it follows that all **not**-free literals that are true in $\Xi^{\uparrow j}(\mathbf{I})$ are true in $\Xi^{\uparrow j}(\mathbf{I}')$ and all **not**-literals that are true in $\Xi^{\uparrow j}(\mathbf{I})$ are true in $\Xi^{\uparrow j}(\mathbf{I}')$. Thus we can conclude that

$$\bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}) \leq \bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}')$$

from the previous fact we can conclude that

$$\Xi^{\uparrow n}(\mathbf{I}) \leq \Xi^{\uparrow n}(\mathbf{I}')$$

□

all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$

E.0.44. Lemma. $\mathbf{M}$ *is a model of* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *iff* $\Gamma(\mathbf{M}) \preceq \mathbf{M}$.

Proof.

1. $(\rightarrow)$:

   Suppose that $\mathbf{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, we will show that $\mathbf{M}$ is a model of $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$. Since $\Gamma(\mathbf{M})$ is the least model of $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$, the claim of the lemma will follow straightforwardly. To prove our claim, we check that $\mathbf{M}$ models the set of premises, PADs, and rules in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$ in turn:

   - Since $\mathcal{S}$ remains untouched after taking the quotient of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, and $\mathbf{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, we can conclude that $\mathbf{M}$ is a model of $\mathcal{S}$
   - By definition of quotient, each PAD in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$ corresponds to a PAD in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$.

     Let $b'_1 \otimes \alpha \otimes b'_2 \rightarrow b'_3 \otimes \alpha \otimes b'_4$ be a PAD in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$, where each conjunct b' in b'_i is the result of the transformation after taking the quotient by $\mathbf{M}$. That is, if b is not a **not**-literal, $b' = b$, and if b is a **not**-literal then b' is equal to $\mathbf{t}^\pi$ or $\mathbf{u}^\pi$ depending on the truth value of $\mathbf{M}(\pi)(b)$, with π a path.

     Since $\mathbf{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, we know that for every path $\langle \mathbf{d}_1, \mathbf{d}_2 \rangle$

     $$\begin{array}{c} \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_1 \rangle)(f) \mid f \in b_1\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_2 \rangle)(f) \mid f \in b_2\}\} \\ \leq \\ \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_1 \rangle)(f) \mid f \in b_3\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_2 \rangle)(f) \mid f \in b_4\}\} \end{array}$$

     By inspection of the quotient transformation, it is clear that for every **not**-literal b in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, which is replaced by b' after taking the quotient modulo $\mathbf{M}$

     $$\mathbf{M}(\pi)(b') = \mathbf{M}(\pi)(b)$$

     This implies that

     $$\begin{array}{c} \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b'_1\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b'_2\}\} \\ \leq \\ \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b'_3\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b'_4\}\} \end{array}$$

     Therefore $\mathbf{M}$ is a model of each PAD in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$, and therefore it is a model of $\mathcal{E}$.
   - The proof that $\mathbf{M}$ is a model of each rule in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$ is analogous to the proof above.

   From the previous facts we can conclude that $\mathbf{M}$ is a model of $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$. Since $\Gamma(\mathbf{M})$ is the least model of $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$, the claim follows.

2. ($\leftarrow$) :

   Now suppose $\Gamma(\mathbf{M}) \preceq \mathbf{M}$. We need to show that $\mathbf{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. We check that $\mathbf{M}$ models the set of premises, PADs, and rules in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ in turn:

   - Since by hypothesis $\Gamma(\mathbf{M}) \preceq \mathbf{M}$, and $\Gamma(\mathbf{M})$ is a model of $\mathcal{S}$, it follows that every premise in $\mathcal{S}$ is also satisfied in $\mathbf{M}$. Thus, $\mathbf{M}$ is a model of $\mathcal{S}$.
   - Let
     $$\lambda = b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \in (\mathcal{E}, \mathbf{P}, \mathcal{S})$$
     By definition of quotient, one of the following two statements holds:
     (a) There is a PAD $\lambda' = b'_1 \otimes \alpha \otimes b'_2 \rightarrow b'_3 \otimes \alpha \otimes b'_4$ in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$ resulting from transforming λ. Each conjunct b' in b'_i in λ' is the result of the transformation after taking the quotient by $\mathbf{M}$.[3]
     (b) The *PAD* λ is not transformed into any λ' in $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{M}}$, and for every path π there is some **not**-literal b in b_1 or b_2 such that $\mathbf{M}(\pi)(b) = \mathbf{f}$.

     We explore each case in turn:

     – Suppose 2a holds for λ. Since for every **not**-literal b in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, which is replaced by b' after taking the quotient modulo $\mathbf{M}$ we know that
     $$\mathbf{M}(\pi)(b') = \mathbf{M}(\pi)(b) \tag{E.9}$$
     and $\Gamma(\mathbf{M}) \models \lambda'$, we know that
     $$\begin{array}{c} \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_1 \rangle)(f) \mid f \in b_1\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_2 \rangle)(f) \mid f \in b_2\}\} \\ \leq \\ \mathsf{min}\{\mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_1 \rangle)(f) \mid f \in b_3\}, \mathsf{min}\{\mathbf{M}(\langle \mathbf{d}_2 \rangle)(f) \mid f \in b_4\}\} \end{array}$$
     And therefore, $\mathbf{M} \models \lambda$.
     – Suppose 2b holds for λ. Then, since the antecedent in λ is false in every path, from (E.9) we can conclude that for every path π, $\mathbf{M}(\pi) \models \lambda$.

     Therefore $\mathbf{M}$ is a model of each PAD in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, and therefore it is a model of $\mathcal{E}$.
   - The proof that $\mathbf{M}$ is a model of each rule in $\mathbf{P}$ is analogous to the proof above.

[3] Observe that for each PAD in $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, the quotient generate possibly infinite numbers of PADs

The previous facts imply that $\mathbf{M}$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ □

E.0.45. COROLLARY. *$WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$.*

Proof. By Definition 3.7.14, we have that

$$\Gamma(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))) \;=\; WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$$

Thus, from Lemma E.0.44, we can conclude that $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. □

The following definition is similar to the definition of unfounded sets in [79].

E.0.46. DEFINITION. [**N**-unsupported] Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. A set **S** of pairs of the form (*atom,path*) is **unsupported** relative to **N**, denoted **N**-unsupported, if for every pair (h, π) in **S**, $\mathbf{N}(\pi)(h) = \mathbf{f}$, or the following conditions hold

1. The *state*-premise $\pi \rhd h \notin \mathcal{S}$

2. if h is a *pda*, and π has the form $\langle \mathbf{d}_1, \mathbf{d}_2 \rangle$, then $\mathbf{d}_1 \overset{h}{\rightsquigarrow} \mathbf{d}_2 \notin \mathcal{S}$

3. for every Horn-rule in **P** of the form $h \leftarrow g_1 \otimes \ldots \otimes g_k$, if π has a split of of the form $\pi = \pi_1 \circ \ldots \circ \pi_k$, then for some body atom g_i, the corresponding pair (g_i, π_i) also belongs to **S** or $\mathbf{N}(\pi_i)(g_i) = \mathbf{f}$

4. if h is a fluent literal and π has the form $\langle \mathbf{d} \rangle$, then for every PAD in $\mathcal{E}$ of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$, the following must hold:

   - if h is a conjunct in b_3 then for every path $\rho = \langle \mathbf{d}, \mathbf{d}' \rangle$
     - either (α, ρ) also belongs to **S**, or $\mathbf{N}(\rho)(\alpha) = \mathbf{f}$; or
     - b_1 has a conjunct g such that $\mathbf{N}(\pi)(g) = \mathbf{f}$ or $(g, \pi) \in \mathcal{S}$ (recall that $\pi = \langle \mathbf{d} \rangle$ here); or
     - b_2 has a conjunct g such that $\mathbf{N}(\langle \mathbf{d}' \rangle)(g) = \mathbf{f}$ or $(g, \langle \mathbf{d}' \rangle) \in \mathcal{S}$.
   - if h is a conjunct in b_4 then for every path $\rho = \langle \mathbf{d}', \mathbf{d} \rangle$:
     - either (α, ρ) also belongs to **S**, or $\mathbf{N}(\rho)(\alpha) = \mathbf{f}$; or
     - b_1 has a conjunct g such that $\mathbf{N}(\langle \mathbf{d}' \rangle)(g) = \mathbf{f}$ or $(g, \langle \mathbf{d}' \rangle) \in \mathcal{S}$; or

– b_2 has a conjunct g such that $\mathbf{N}(\pi)(g) = \mathbf{f}$ or $(g, \pi) \in \mathcal{S}$. □

E.0.47. Definition. The **greatest unsupported set** relative to $\mathbf{N}$, denoted $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{N})$, is the union of all sets that are unsupported relative to $\mathbf{N}$. □

E.0.48. Lemma. *Let* $\mathbf{I}$ *and* $\mathbf{I}'$ *two partial Herbrand structures.*

$$\textit{If} \quad \mathbf{I} \leq \mathbf{I}' \quad \textit{then} \quad \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I}) \leq \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I}')$$

Proof. The claim of the follows straightforwardly from the definition of greatest unsupported set. □

For the next results we need the following notation. Let $\mathcal{N}$ be a partial Herbrand Structure and π a path. Then

$$\begin{array}{lcl} \texttt{True}(\mathcal{N})(\pi) & = & \{h \mid \mathcal{N}(\pi)(h) = \mathbf{t} \text{ and } h \text{ is } \textbf{not}\text{-free}\} \\ \texttt{False}(\mathcal{N})(\pi) & = & \{h \mid \mathcal{N}(\pi)(h) = \mathbf{f} \text{ and } h \text{ is } \textbf{not}\text{-free}\} \\ \texttt{Ukn}(\mathcal{N})(\pi) & = & \{h \mid \mathcal{N}(\pi)(h) = \mathbf{u} \text{ and } h \text{ is } \textbf{not}\text{-free}\} \end{array}$$

In addition, we will overload the definitions of the sets `True` and `False` by using them with 2-valued Herbrand structures.

We say that a literal a **depends** on a literal b if there is a rule of the form $a \leftarrow G$ in $\mathbf{P}$, and either *(i)* b is a conjunct in G, or *(ii)* there is a conjunct c in G such that c depends on b.

The following lemma states that the unsupported set do no contain pairs of the form (h, π) such that h is true in π.

In the following, let $(\mathcal{E}, \mathbf{P}, \mathcal{S})_N = \frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\Gamma^{\uparrow N-1}(\mathbf{I}_0)}$

E.0.49. Lemma. *Let* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *be a* $\mathcal{TR}^{PAD}$*. Then*

$$\texttt{True}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi) \cap \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))\} = \emptyset$$

Proof. In the following, given an ordinal N, let us do some abuse of notation and write $\Gamma^{\uparrow N}$ to refer to $\Gamma^{\uparrow N}(\mathbf{I}_0)$. Since *(i)* $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is the limit of the Γ operator, *(ii)* Γ is monotonic by Lemma E.0.43, and *(iii)* by Lemma E.0.48, if $\mathbf{I} \leq \mathbf{I}'$ then $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I}) \leq \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I}')$, we can conclude that it is enough to prove that for every ordinal N, and every path π:

$$\texttt{True}(\Gamma^{\uparrow N})(\pi) \cap \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})_N}(\Gamma^{\uparrow N})\} = \emptyset$$

Let A be any set of pairs (literal,path) such that

$$\texttt{True}(\Gamma^{\uparrow N})(\pi) \cap \{h \mid (h, \pi) \in A\} \neq \emptyset \tag{E.10}$$

We will prove by contradiction that A is not Γ^N-unsupported. This implies that $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})_N}(\Gamma^{\uparrow N})$ cannot intersect with $\{(h, \pi) \mid h \in \texttt{True}(\Gamma^{\uparrow N})(\pi)\}$ for any π.

We know that

$$\texttt{True}(\Gamma^{\uparrow N})(\pi) = LM((\mathcal{E}, \mathbf{P}, \mathcal{S})^{-}_{N})(\pi)$$

Recall that $(\mathcal{E}, \mathbf{P}, \mathcal{S})^{-}_{N}$ is **not**-free and it does not contain propositions of the form $\mathbf{u}^{\rho}$ for any path ρ, thus it has a least Herbrand (2-valued) model. In the previous sections we proved that such model can be constructed iteratively in a bottom-up fashion.

Let $(\mathcal{O}^{d-}, \mathcal{O}^{t-})$ be the compliant oracles of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^{-}_{N}$. Let $\mathbf{P}^{-}$ be the transaction bases of $(\mathcal{E}, \mathbf{P}, \mathcal{S})^{-}_{N}$. Recall that by Lemmas E.0.30 and E.0.31, the specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})^{-}_{N}$ has the same least model as $(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$.

Suppose that A is Γ^{N}-unsupported. Let us choose the earliest step n in the construction of the LM of $(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$ such that for some literal h and path π we can derive

$$LM(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})(\pi) \models h \text{ and } (h, \pi) \in A \tag{E.11}$$

It follows that one of the next two statements hold

1. Neither the transaction nor the data oracle, $(\mathcal{O}^{d-}, \mathcal{O}^{t-})$, enforce h to be true in path π in $LM(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$.

2. Either the transaction or the data oracle, $(\mathcal{O}^{d-}, \mathcal{O}^{t-})$, enforce h to be true in path π in $LM(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$.

The proofs for both cases are similar. The former is by transfinite induction on the number of steps needed to construct $LM(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$ and the later is by induction on the number of steps needed to construct the oracles $(\mathcal{O}^{d-}, \mathcal{O}^{t-})$. For concreteness, suppose that oracles $(\mathcal{O}^{d-}, \mathcal{O}^{t-})$ do not enforce h to be true in π.

It follows that there is a rule in $\mathbf{P}^{-}$ of the form $h \leftarrow G$, and G is true in the $n-1$ step in the path π. Since A is Γ^{N}-unsupported and G is not false in π, we can conclude from (E.11) that there is some conjunct $g_i \in G$ and a path π_i in some split of π such that $(g_i, \pi) \in A$. However, by hypothesis, A does not intersect with the Herbrand structure in the $n-1$ step of the construction of $LM(\mathcal{O}^{d-}, \mathcal{O}^{t-}, \mathbf{P}^{-})$. Thus, A is not Γ^{N}-unsupported.

From the previous facts, we can conclude that for every π

$$\texttt{True}(\Gamma^{\uparrow N})(\pi) \cap \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E}, \mathbf{P}, \mathcal{S})_N}(\Gamma^{N})\} = \emptyset$$

□

The following lemma states that the unsupported set do no contain pairs of the form (h, π) such that h is undefined in π.

E.0.50. LEMMA. *Let* $(\mathcal{E},\mathbf{P},\mathcal{S})$ *be a* $\mathcal{TR}^{PAD}$*. Then*

$$\texttt{Ukn}(WFM((\mathcal{E},\mathbf{P},\mathcal{S})))(\pi)\cap\{h\mid(h,\pi)\in\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E},\mathbf{P},\mathcal{S}))\}=\emptyset$$

Proof. Since *(i)* $WFM((\mathcal{E},\mathbf{P},\mathcal{S}))$ is the limit of the Γ operator, *(ii)* Γ is monotonic by Lemma E.0.43, and *(iii)* by Lemma E.0.48, if $\mathbf{I}\leq\mathbf{I}'$ then $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I})\leq\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(\mathbf{I}')$, we can conclude that it is enough to prove that for every ordinal N, and every path π:

$$\texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)\cap\{h\mid(h,\pi)\in\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})_N}(\Gamma^{\uparrow N})\}=\emptyset$$

Observe that by construction of $\Gamma^{\uparrow N+1}$,

$$\begin{array}{c}\texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)\\=\\\texttt{False}(LM((\mathcal{E},\mathbf{P},\mathcal{S})^{-}_{N+1}))(\pi)\cap\texttt{True}(LM((\mathcal{E},\mathbf{P},\mathcal{S})^{+}_{N+1}))(\pi)\end{array}\tag{E.12}$$

Let A be any set of pairs (literal,path) such that

$$\texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)\cap\{h\mid(h,\pi)\in A\}\neq\emptyset\tag{E.13}$$

We will prove by contradiction that A is not Γ^{N+1}-unsupported. This implies that $\{h\mid(h,\pi)\in\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})_N}(\Gamma^{\uparrow N})\}$ cannot intersect with $\texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)$ for any π.

Suppose that A is Γ^{N}-unsupported. Before continuing with the proof, let us switch to the oracalized version of $(\mathcal{E},\mathbf{P},\mathcal{S})^{+}_{N+1}$ and $(\mathcal{E},\mathbf{P},\mathcal{S})^{-}_{N+1}$. Let $(\mathcal{O}^{d+},\mathcal{O}^{t+})$ and $(\mathcal{O}^{d-},\mathcal{O}^{t-})$ the $\mathcal{TR}^{PAD}$ compliant oracles of $(\mathcal{E},\mathbf{P},\mathcal{S})^{+}_{N+1}$ and $(\mathcal{E},\mathbf{P},\mathcal{S})^{-}_{N+1}$ respectively. Let $\mathbf{P}^{+}$ and $\mathbf{P}^{-}$ be the transaction bases of $(\mathcal{E},\mathbf{P},\mathcal{S})^{+}_{N+1}$ and $(\mathcal{E},\mathbf{P},\mathcal{S})^{-}_{N+1}$ respectively. Recall that by Lemmas E.0.30 and E.0.31, the specifications $(\mathcal{E},\mathbf{P},\mathcal{S})^{+}_{N+1}$ and $(\mathcal{E},\mathbf{P},\mathcal{S})^{-}_{N+1}$ have the same least model as $(\mathcal{O}^{d+},\mathcal{O}^{t+},\mathbf{P}^{+})$ and $(\mathcal{O}^{d-},\mathcal{O}^{t-},\mathbf{P}^{-})$ respectively.

Now, suppose that $(h,\pi)\in\texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)\cap\{h\mid(h,\pi)\in A$. Let us choose the earliest step n in the construction of the LM of $(\mathcal{O}^{d+},\mathcal{O}^{t+},\mathbf{P}^{+})$ such that we can derive

$$\begin{array}{ll}(1) & LM(\mathcal{O}^{d+},\mathcal{O}^{t+},\mathbf{P})(\pi)\models h\\(2) & LM(\mathcal{O}^{d-},\mathcal{O}^{t-},\mathbf{P})(\pi)\not\models h\end{array}\tag{E.14}$$

We know that such step exists since $(h,\pi)\in\texttt{Ukn}(\Gamma^{\uparrow N+1})$. From (1) above, it follows that one of the next two statements hold

1. Neither the transaction nor the data oracle, $(\mathcal{O}^{d+},\mathcal{O}^{t+})$, enforce h to be true in path π in $LM(\mathcal{O}^{d+},\mathcal{O}^{t+},\mathbf{P}^{+})$.

2. Either the transaction or the data oracle, $(\mathcal{O}^{d+},\mathcal{O}^{t+})$, enforce h to be true in path π in $LM(\mathcal{O}^{d+},\mathcal{O}^{t+},\mathbf{P}^{+})$.

Both proofs are similar, the former one is by transfinite induction on the number of steps needed to construct $LM(\mathcal{O}^{d+}, \mathcal{O}^{t+}, \mathbf{P}^+)$ and the later one is by induction on the number of steps needed to construct the oracles $(\mathcal{O}^{d+}, \mathcal{O}^{t+})$. For concreteness, suppose that oracles $(\mathcal{O}^{d+}, \mathcal{O}^{t+})$ do not enforce h to be true in π.

It follows that there is a rule in $\mathbf{P}^+$ of the form $h \leftarrow G$, and G is true in the $n-1$ step of the LM construction in the path π. Observe that because Γ is monotonic, it cannot be that $\Gamma^N(\pi)(h) = \mathbf{f}$, given that $\Gamma^{N+1}(\pi)(h) = \mathbf{u}$. Thus, since A is Γ^N-unsupported, we can conclude that there is some conjunct $g_i \in G$ and a path π_i in some split of π such that $(g_i, \pi_i) \in A$. However, by hypothesis, A does not intersect with the Herbrand structure in the $n-1$ step of the construction of $LM(\mathcal{O}^{d+}, \mathcal{O}^{t+}, \mathbf{P}^+)$. Thus, A is not Γ^N-unsupported.

This implies that $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})_N}(\Gamma^{\uparrow N})\}$ cannot contain a pair (h, π) such that $h \in \texttt{Ukn}(\Gamma^{\uparrow N+1})(\pi)$ for any path π □

E.0.51. Lemma. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a $\mathcal{TR}^{PAD}$ specification. Then,*

$$\texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi) \subseteq \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))\}$$

Proof. Follows from the definition of unsupported set. □

E.0.52. Proposition. *Let $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ be the well-founded model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$. Let $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))$ be the greatest unsupported set relative to $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$. Then, for every path π*

$$\{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))\} = \texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi)$$

Proof. From Lemma E.0.51, it follows that

$$\texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi) \subseteq \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))\}$$

and from Lemmas E.0.49 and E.0.50, we can conclude that
$\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))$ does not contain any pair (h, π) such that

$$WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))(\pi)(h) \geq \mathbf{u}$$

Thus, $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))$ contains the literals and paths which are false in $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ and nothing else. That is

$$\texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi) = \{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))\}$$

□

E.0.53. THEOREM. *$WFM((\mathcal{E},\mathbf{P},\mathcal{S}))$ is the least model of $(\mathcal{E},\mathbf{P},\mathcal{S})$.*

Proof. By Corollary E.0.45, $WFM((\mathcal{E},\mathbf{P},\mathcal{S}))$ is a model of $(\mathcal{E},\mathbf{P},\mathcal{S})$. Suppose that $\mathbf{N}$ is a model of $(\mathcal{E},\mathbf{P},\mathcal{S})$. We need to show that for every literal h and path π.

$$WFM((\mathcal{E},\mathbf{P},\mathcal{S}))(\pi)(h) \preceq \mathbf{N}(\pi)(h)$$

To prove this, we will show that

1. $\texttt{True}(WFM((\mathcal{E},\mathbf{P},\mathcal{S}))(\pi)) \subseteq \texttt{True}(\mathbf{N}(\pi))$
2. $\texttt{False}(\mathbf{N}(\pi)) \subseteq \texttt{False}(WFM((\mathcal{E},\mathbf{P},\mathcal{S}))(\pi))$.

We prove each of these items in turn:

- Claim 1 can be shown using the monotonicity of the immediate consequence operator Γ. That is,

$$\begin{array}{c}\texttt{True}(I_\emptyset(\pi)) = \emptyset \subseteq \texttt{True}(\mathbf{N})(\pi)\\ \texttt{True}(\Gamma^{\uparrow 1}(I_\emptyset))(\pi) \subseteq \texttt{True}(\Gamma^{\uparrow 1}(\mathbf{N}))(\pi)\\ \vdots\\ \texttt{True}(\Gamma^{\uparrow n}(I_\emptyset))(\pi) \subseteq \texttt{True}(\Gamma^{\uparrow n}(\mathbf{N}))(\pi)\end{array}$$

  In addition, by Lemma E.0.43 we have that

$$\begin{array}{c}\texttt{True}(\Gamma^{\uparrow 1}(\mathbf{N}))(\pi) \subseteq \texttt{True}(\mathbf{N})(\pi)\\ \vdots\\ \texttt{True}(\Gamma^{\uparrow n}(\mathbf{N}))(\pi) \subseteq \texttt{True}(\Gamma^{\uparrow n-1}(\mathbf{N}))(\pi)\end{array}$$

  Therefore

$$\texttt{True}(\Gamma^{\uparrow n}(I_\emptyset))(\pi) \subseteq \texttt{True}(\Gamma^{\uparrow n}(\mathbf{N}))(\pi) \subseteq \texttt{True}(\mathbf{N})(\pi)$$

  and if α is a limit ordinal then

$$\texttt{True}(\Gamma^{\uparrow \alpha}(I_\emptyset))(\pi) \subseteq \texttt{True}(\mathbf{N})(\pi)$$

  Since $\texttt{True}(WFM((\mathcal{E},\mathbf{P},\mathcal{S})))(\pi)$ is the limit of the monotonically growing sequence of $\texttt{True}(\Gamma^{\uparrow\alpha}(I_\emptyset))(\pi)$'s, we conclude

$$\texttt{True}(WFM((\mathcal{E},\mathbf{P},\mathcal{S})))(\pi) \subseteq \texttt{True}(\mathbf{N})(\pi) \tag{E.15}$$

- To prove claim 2, we will show that the set

$$\{(h,\pi) \mid h \in \texttt{False}(\mathbf{N})(\pi)\} \tag{E.16}$$

  is unfounded with respect to $\mathbf{I}_0$. For convenience, we will denote $\{(h,\pi) \mid h \in \texttt{False}(\mathbf{N})(\pi)\}$ with $\mathcal{U}$. To prove(E.16), let h be a **not**-free literal such that $(h,\pi) \in \mathcal{U}$.

1. If $\pi \rhd h$ were a state premise in $\mathcal{S}$ then h would have been true on π and thus (h, π) would not be in $\mathcal{U}$.
2. if $\mathbf{d}_1 \overset{h}{\rightsquigarrow} \mathbf{d}_2$ (where $\pi = \langle \mathbf{d}_1, \mathbf{d}_2 \rangle$) were a run-premise in $\mathcal{S}$ then h would have been true on $\langle \mathbf{d}_1, \mathbf{d}_2 \rangle$, in $\mathbf{N}$ and thus (h, π) would not be in $\mathcal{U}$.
3. Suppose $h \leftarrow g_1 \otimes \ldots \otimes g_k \in \mathbf{P}$ and consider a split $\pi = \pi_1 \circ \ldots \circ \pi_k$. Suppose that for none of the g_is it is the case that $(g_i, \pi) \in \mathcal{U}$. Then it must hold that $\mathbf{N}(\pi)(g_i)$ is either $\mathbf{t}$ or $\mathbf{u}$ and thus so is $\mathbf{N}(\pi)(h)$, contrary to the assumption that $(h, \pi) \in \mathcal{U}$.
4. Suppose that $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ has a PAD of the form $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$, $\pi = \langle \mathbf{d} \rangle$, and h is a conjunct in b_3 or b_4.
   - If h is a conjunct in b_3, then for every path $\rho = \langle \mathbf{d}, \mathbf{d}' \rangle$ it must be the case that $\mathbf{N}(\rho)(b_3 \otimes \alpha \otimes b_4) = \mathbf{f}$. Hence $\mathbf{N}(\rho)(b_1 \otimes \alpha \otimes b_2) = \mathbf{f}$. But then, it must be the case that either $\mathbf{N}(\rho)(\alpha) = \mathbf{f}$, or there is a conjunct b' in b_1 such that $\mathbf{N}(\langle \mathbf{d} \rangle)(b') = \mathbf{f}$, or for some conjunct b' in b_2 it must be the case that $\mathbf{N}(\langle \mathbf{d}' \rangle)(b') = \mathbf{f}$. By Definition E.0.46, a
   - If h is a conjunct in b_4, we can similarly prove that for every path $\rho = \langle \mathbf{d}', \mathbf{d} \rangle$ either $\mathbf{N}(\rho)(\alpha) = \mathbf{f}$, or there is a conjunct b' in b_1 or b_2 such that $\mathbf{N}(\langle \mathbf{d} \rangle)(b') = \mathbf{f}$ or $\mathbf{N}(\langle \mathbf{d}' \rangle)(b') = \mathbf{f}$ (whichever applies).

   The above four cases cover exactly what is required by Definition E.0.46 for $\mathcal{U}$ to be unsupported. If $\mathcal{U}$ is unsupported relative to $\mathbf{I}_0$ then it is also an unsupported relative to any other path structure, $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ in particular. Since $\mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))$ is the greatest unsupported set, we know that

$$\mathcal{U} \subseteq \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))$$

   By Proposition E.0.52, for every path π

$$\{h \mid (h, \pi) \in \mathbf{U}_{(\mathcal{E},\mathbf{P},\mathcal{S})}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))\} = \texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi)$$

   From this and (E.16) we conclude that for any path π,

$$\texttt{False}(\mathbf{N})(\pi) \subseteq \texttt{False}(WFM((\mathcal{E}, \mathbf{P}, \mathcal{S})))(\pi)$$

   which concludes the proof. □

Relation between the action theories in $\mathcal{TR}^{PAD}$ with and without not

In this section we establish the relation between the action theory in $\mathcal{TR}^{PAD}$ with respect to the action theory in $\mathcal{TR}^{PAD}$ with default negation. We repeat the main definitions for convenient reference:

E.0.54. Definition. [Simple Specification] We say that a specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is simple if

- It makes no use of default negation.
- It has no interloping actions.
- There is no literal f such that there exists a path π where $\mathcal{E}, \mathbf{P}, \mathcal{S}, \pi \models f$ and $\mathcal{E}, \mathbf{P}, \mathcal{S}, \pi \models \mathbf{neg}\, f$.
- For every database state identifier $\mathbf{d}$ and literal f, $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models f$ or $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models \mathbf{neg}\, f$. □

We say that a partial Herbrand path structure $\mathbf{I}$ is 2 valued, if and only if for every path π and **not**-free literal h, either $\mathbf{I}(\pi)(h) = \mathbf{t}$, or $\mathbf{I}(\pi)(h) = \mathbf{f}$.

E.0.55. Definition. We define the function $2val$ from simple partial Herbrand path structures, to Herbrand path structures as follows:

$$2val(\mathbf{I})(\pi) = \{f \mid \mathbf{I}(\pi)(f) = \mathbf{t}\}$$

□

Since in $\mathcal{TR}^{PAD}$ with default negation we allow interloping action, in the general case we cannot have neither Causality frame axioms nor Backward Projection. However, since now we want to compare these two different action theories for simple specifications, we add the Causality frame axioms and Backward Projection to $\mathcal{A}(\mathcal{E})$ as defined in Section 3.4.

E.0.56. Lemma. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a simple specification. Let $\mathbf{I}$ be a model of $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$. For every* **not***-free serial conjunction G,*

$$\textit{if } \mathbf{I}, \pi \models G \textit{ then } 2val(\mathbf{I}), \pi \models G$$

Proof. Follows straightforwardly from Definition 3.8.2. □

E.0.57. Theorem. *Let $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ be a simple specification. Let $\mathbf{I}$ be a model of $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$. Then $2val(\mathbf{I})$ is a model of $(\mathcal{A}(\mathbf{P} \cup \mathcal{E}), \mathcal{S})$.*

Proof. Since $\mathbf{I}$ is a model of $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$, and $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$ is simple, it follows that

$$\mathbf{I}(\pi)(\mathbf{not}\, inconsistent) = \mathbf{t} \tag{E.17}$$

for every path π. Therefore, since **not** occurs in $(\mathcal{A}(\mathcal{E}), \mathbf{P}, \mathcal{S})$ only in $Frame(\mathcal{E})$ preceding the fluent *inconsistent* we can build a new equivalent specification $\mathbf{not} - free$. This can be done by replacing every **not**-literal of the form $\mathbf{not}\, inconsistent$ by $\mathbf{t}^{\pi}$ for every path π. It is easy to see that $\mathbf{I}$ does not contains undefined facts. From the previous fact and Lemma E.0.56, we can conclude that $2val(\mathbf{I}) \models \mathcal{E}$, $2val(\mathbf{I}) \models \mathbf{P}$ and $2val(\mathbf{I}) \models \mathcal{S}$. To conclude the proof of the theorem we will show that

$$2val(\mathbf{I}) \models \mathcal{A}(\mathbf{P} \cup \mathcal{E})$$

We consider each frame axiom in turn:

- Forward disablement frame axiom: We know that $\mathbf{I}$ is a model of the following forward disablement frame axiom in $\mathcal{A}(\mathcal{E})$:

  $$(inertial(g) \wedge \mathbf{not}\, f_g^1 \wedge \cdots \wedge \mathbf{not}\, f_g^n) \otimes g \otimes \alpha \otimes \mathbf{not}\, \mathit{inconsistent} \rightarrow \alpha \otimes g$$

  Since we do not have interloping actions, there can be at most one partially defined action $\mathbf{p}_g$ with the primitive effect g. Let f_g be the precondition of $\mathbf{p}_g$. Then, the PAD above is equivalent modulo $\mathbf{I}$ to

  $$(inertial(g) \wedge \mathbf{not}\, f_g) \otimes g \otimes \alpha \otimes \mathbf{not}\, \mathit{inconsistent} \rightarrow \alpha \otimes g$$

  By (E.17), we can reformulate this frame axiom as

  $$(inertial(g) \wedge \mathbf{not}\, f_g) \otimes g \otimes \alpha \rightarrow \alpha \otimes g \tag{E.18}$$

  Since every state is complete and consistent in $\mathbf{I}$ (by definition of simple specifications), we can still rewrite axiom (E.18) as

  $$(inertial(g) \wedge \mathbf{neg}\, f_g) \otimes g \otimes \alpha \rightarrow \alpha \otimes g \tag{E.19}$$

  Let $\mathbf{p}_{\mathbf{neg}\, g}$ the PAD with the primitive effect $\mathbf{neg}\, g$. Let $f_{\mathbf{neg}\, g}$ be the precondition of $\mathbf{p}_{\mathbf{neg}\, g}$. It is clear that

  $$\mathbf{I}(\pi)(inertial(g) \wedge \mathbf{neg}\, f_g) \geq \mathbf{I}(\pi)(inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g})$$

  Therefore, if $\mathbf{I}$ models the Forward Inertia Axiom in $\mathcal{A}(\mathcal{E})$, by Lemma E.0.56 $2val(\mathbf{I})$ has to model the the Forward Inertia Axiom in $(\mathcal{A}(\mathbf{P} \cup \mathcal{E}), \mathcal{S})$.

- Backward disablement frame axiom: Analogous to the previous case.

- Forward Inertia axiom: By (E.17), we can reformulate this frame axiom as

$$(inertial(h) \wedge h) \otimes \alpha \rightarrow \alpha \otimes h \tag{E.20}$$

  Therefore, by Lemma E.0.56 it follows that $2val(\mathbf{I})$ models the Forward Inertia axiom in $(\mathcal{A}(\mathbf{P} \cup \mathcal{E}), \mathcal{S})$ as well.

- Backward Inertia axiom: Analogous to the previous case.

- Causality and Backward Projection. Trivially follows since they are syntactically identical to the ones in $(\mathcal{A}(\mathbf{P} \cup \mathcal{E}), \mathcal{S})$.

□

Appendix F

Proof of the reduction of the action language $\mathcal{L}_1$ to $\mathcal{TR}^{PAD}$

In this appendix we prove soundness and completeness of the reduction of the action language $\mathcal{L}_1$ to $\mathcal{TR}^{PAD}$ developed in Chapter 4.

Soundness of the $\mathcal{L}_1$ reduction to $\mathcal{TR}^{PAD}$

Recall that the action theory $\mathcal{A}(\mathbf{P})$ for a transaction base $\mathbf{P}$ is defined as $\mathbf{P} \cup Frame(\mathbf{P})$. In addition, recall that in this section $inertial(f)$ is "always true" for every fluent and every state. Thus, to avoid tedious repetition, we remove the *inertial* predicate from every rule in the action theory.

Recall that the query language in $\mathcal{L}_1$ consist of hypothesis of the form

$$q = f \texttt{ after } [a_1, a_2, \ldots, a_n] \texttt{ at } s_i$$

Therefore, to check soundness we restrict our query language to statements of the form: $a_1 \otimes a_2 \otimes \cdots \otimes a_n \otimes f$.

F.0.58. THEOREM. *(**Soundness**) Let $\mathcal{D}$ be a simple domain description. Let $\Lambda(\mathcal{D}) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}^{PAD}$ reduction of $\mathcal{D}$, and α be a serial conjunction of actions. Suppose that $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models \alpha \otimes f$. Then $\mathcal{D} \models f$ `after` α `at` s_i.*

Proof. The proof relies on the fact that $\mathcal{TR}^{PAD^+}$ Transaction Logic has a sound and complete proof theory. Observe that $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models a_2 \otimes \cdots \otimes a_n$ if and only if the set of premises $\{\mathbf{d}_i \overset{a_j}{\rightsquigarrow} \mathbf{d}_{j+1} \mid i \leq j < n\}$ is a subset of $\mathcal{S}$; and by construction this happens if and only if $\mathcal{D}$ contains the set of occurrence facts $\{a_j \texttt{ occurs\_at } s_j \mid i \leq j < n\}$. From thee previous fact we can conclude that

$\mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models \alpha$ if and only if $D \models \alpha$ `occurs_at` $\mathtt{s_i}$. Thus, we only need to prove that

$$\text{if } \mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f_1 \ldots f_m \text{ then } \mathcal{D} \models f_1 \ldots f_m \texttt{ at } s_n$$

We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f_1 \ldots f_m \tag{F.1}$$

using the inference rules and the axioms presented in Section 3.3.
To avoid tedious repetitions, we write $D \models \mathtt{f_1} \ldots \mathtt{f_m}$ `at` $\mathtt{s_n}$ to represent the set of statements $D \models \mathtt{f_1}$ `at` $\mathtt{s_n}$,...,$D \models \mathtt{f_m}$ `at` $\mathtt{s_n}$.

Base case: $N = 1$. In that case, (F.1) can only be derived by the premise inference rule, and (F.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f$ for some fluent literal f. (Note that the No-Op axiom can not be used since () is not a fluent, and no other inference rules (even the hypothetical rule) can be used to derive a sequent in the first inference step.) By the construction of Λ, it follows that $\mathtt{f\ at\ s_n} \in D$. By definition of satisfaction in $\mathcal{L}_1$ we can conclude that $D \models \mathtt{f\ at\ s_n}$.

Induction step: $N = k > 1$ and we assume that whenever (F.1) can be derived by the proof theory in less than k steps, then for every $i = 1 \ldots m$, $D \models \mathtt{f_i\ at\ s_n}$.

To prove that the same holds also when (F.1) is derived using k steps, note that the last step in the derivation must be an application of one of these rules in $\mathcal{F}$:

- The Forward Projection rule.
- The Decomposition rule.
- The Sequencing rule.

We are not considering the Horn inference rule because we do not have complex actions or defined fluents in our reduction. We consider each of the cases listed above in turn.

- ***Forward Projection:*** Suppose that (F.1) was derived via the Forward Projection rule. This means that (F.1) was derived using a PAD $\mathbf{p} \in \mathbf{P}$ that belongs to one of the following types of rules:
  - Causal PAD
  - Forward Inertia
  - Causality
  - Forward Disablement

– Backward Disablement
– Backward Inertia
– Backward Projection

Let us examine these cases one by one

– *Causal PAD:* Suppose (F.1) was derived via the Forward Projection rule and **p** is a causal PAD. This implies that:
  1. **p** has the form $b_1 \otimes \alpha \rightarrow \alpha \otimes b_2$,
  2. $b_2 = f_1 \otimes \cdots \otimes f_m$, and,
  3. the following statements were derived in less than k steps:
     (a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1}, \mathbf{d}_n \vdash \alpha$
     (b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1} \vdash b_1$

  Item 1 implies that there is a causal law in D of the form

  $$\alpha \texttt{ causes } \mathtt{b_2} \texttt{ if } \mathtt{b_1}$$

  and b_2 is a fluent literal. From Item 3a we can conclude that there is a run-premise $\mathbf{d}_{n-1} \overset{\alpha}{\rightsquigarrow} \mathbf{d}_n \in \mathcal{S}$. Thus α `occurs_at` $\mathtt{s_{n-1}} \in D$. By the inductive hypothesis on Item 3b, we have that $D \models \mathtt{b_1}$ `at` $\mathtt{s_{n-1}}$. This fact implies that b_2 is an immediate effect[1] of α in $act2st(sit2act(s_{n-1}))$. This means that $b_2 \in E^+(act2st(sit2act(s_{n-1})))$ or $b_2 \in E^-(act2st(sit2act(s_{n-1})))$ depending on whether b_2 is preceded by $\neg$ or not. Therefore, the definition of *Res* yields $b_2 \in act2st(sit2act(s_{n-1}) \circ \alpha)$ if b_2 is a positive fluent, or
  $b_2 \notin act2st(sit2act(s_{n-1}) \circ \alpha)$ otherwise. From the previous facts, it follows that $D \models \mathtt{b_2}$ `at` $\mathtt{s_n}$.

– *Forward Inertia:* Suppose (F.1) was derived via the Forward Projection rule and **p** is a Forward Inertia rule. This implies that:
  1. **p** has the form: $f \otimes \alpha \rightarrow \alpha \otimes f$
  2. (F.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f$
  3. neither f nor **neg** f is a primitive effect of α, and,
  4. the following statements were derived in less than k steps:
     (a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1} \vdash f$
     (b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1}, \mathbf{d}_n \vdash \alpha$

[1] Recall that a fluent literal f is an immediate effect of an action a in a state σ, if there is a causal law `a causes f if` $\mathtt{f_1} \ldots \mathtt{f_m}$ in D, whose preconditions $f_1 \ldots f_m$ hold in σ.

Following the same reasoning as before, we can conclude that

$$\alpha \texttt{ occurs\_at } \texttt{s}_{\texttt{n}-1} \in \texttt{D}$$

and by the inductive hypothesis $D \models \texttt{f at } \texttt{s}_{\texttt{n}-1}$. The definition of satisfaction in $\mathcal{L}_1$ ensures that $f \in act2st(sit2act(s_{n-1}))$. From Item 3 we have that neither f nor $\neg f$ are effects of α; thus the definition of E^- yields $f \notin E^-(act2st(sit2act(s_{n-1})))$. From the definition of Res, we can conclude that $f \in act2st(sit2act(s_{n-1}) \circ \alpha)$, and hence, $D \models f \texttt{ at } s_n$.

– *Causality:* Suppose (F.1) was derived via the Forward Projection rule and $\mathbf{p}$ is a Causality rule. This implies that:

1. $\mathbf{p}$ has the form: $\mathbf{neg}\, h \otimes \beta \otimes h \rightarrow b_1 \otimes \beta$,
2. The following statements were derived in less than k steps:
   (a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_n, \mathbf{d}_{n+1} \vdash \beta$
   (b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \neg h$
   (c) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n+1} \vdash h$

Item 1 implies that there is a causal law $b_1 \otimes \beta \rightarrow \beta \otimes b_2$ and $h = b_2$. From 2a we can conclude that there is a premise $\mathbf{d}_{n-1} \overset{\beta}{\rightsquigarrow} \mathbf{d}_n$ in $\mathcal{S}$. Thus, $\beta \texttt{ occurs\_at } \texttt{s}_\texttt{n}$ is in D. By the inductive assumption on 2b and 2c, we have that $D \models \neg\texttt{h at } \texttt{s}_\texttt{n}$ and $D \models \texttt{h at } \texttt{s}_{\texttt{n+1}}$. Since in $\mathcal{L}_1$ semantics, changes in the values of fluents can only be caused by execution of actions it must be the case that $h \in E^+(act2st(sit2act(s_n)))$. By the definition of Res it follows that $b_1 \in act2st(sit2act(s_n))$. The previous fact implies that for every $f \in b_1$, $D \models \texttt{f at } \texttt{s}_\texttt{n}$.

– *Forward Disablement :* Suppose (F.1) was derived via the Forward Projection rule and $\mathbf{p}$ is a Forward Disablement rule.
Recall that due to the interloping assumption, there can be at most one Causal PAD $\mathbf{p}_f$ with the primitive effect f and at most one PAD $\mathbf{p}_{\mathbf{neg}\, f}$ with the primitive effect $\mathbf{neg}\, f$. Let $\mathbf{q}_f$ be the precondition of $\mathbf{p}_f$ and $\mathbf{q}_{\mathbf{neg}\, f}$ the precondition of $\mathbf{p}_{\mathbf{neg}\, f}$. This implies that:

1. $\mathbf{p}$ has the form: $\mathbf{neg}\, \mathbf{q}_f \wedge \mathbf{neg}\, \mathbf{q}_{\mathbf{neg}\, f} \otimes f \otimes \alpha \rightarrow \alpha \otimes f$
2. (F.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash f$
3. The following statements were derived in less than k steps:
   (a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1}, \mathbf{d}_n \vdash \alpha$, and
   (b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n-1} \vdash \mathbf{neg}\, \mathbf{q}_f \wedge \mathbf{neg}\, \mathbf{q}_{\mathbf{neg}\, f} \otimes f$.

(1) implies that there is a causal law in D of the form α $\texttt{causes f if}$ $\mathbf{q_f}$. From Item 3a we can conclude that α $\texttt{occurs\_at}$ $\mathtt{s_{n-1}} \in D$, and by the inductive hypothesis on 3b,

* $D \models \neg \mathbf{q_f}$ $\texttt{at}$ $\mathtt{s_{n-1}}$.
* $D \models \neg \mathbf{q_{neg}}_f$ $\texttt{at}$ s_{n-1}.
* $D \models f$ $\texttt{at}$ s_{n-1}.

Clearly f is not an immediate effect of α in $(act2st(sit2act(s_{n-1})))$. In particular, the definition of E^- yields $f \notin E^-(act2st(sit2act(s_{n-1})))$. Thus, by the definition of Res $f \in act2st(sit2act(s_{n-1}) \circ \alpha)$. From the previous fact it follows that $D \models f$ $\texttt{at}$ s_n.

– *Backward Disablement :* This case is completely symmetrical to the Forward Disablement case.

– *Backward Inertia :* Suppose (F.1) was derived via the Forward Projection rule and **p** is a Backward Inertia rule.

1. **p** has the form: $\beta \otimes \mathbf{neg}\, f \rightarrow \mathbf{neg}\, f \otimes \beta$
2. (F.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \mathbf{neg}\, f$[2].
3. The following statements were derived in less than k steps:
   (a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_n, \mathbf{d}_{n+1} \vdash \beta$, and
   (b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n+1} \vdash f$.

From Item 3a we can conclude that β $\texttt{occurs\_at}$ $\mathtt{s_n} \in D$, and by the inductive hypothesis on 3b, we have that $D \models \neg$ $\texttt{f}$ $\texttt{at}$ $\mathtt{s_{n+1}}$. The cases where f is a positive literal or a negative literal are completely symmetric. For concreteness, suppose f is a positive literal. Clearly f is not an immediate effect of β in $(act2st(sit2act(s_n)))$. In particular, the definition of E^- yields $f \notin E^-(act2st(sit2act(s_n)))$. Thus, by from the previous facts and the definition of Res, we can conclude that $f \notin act2st(sit2act(s_n))$. From the definition of satisfaction in $\mathcal{L}_1$ It follows that $D \models \neg f$ $\texttt{at}$ s_n.

– *Backwards Projection:* Suppose (F.1) is derived by a Backwards Projection rule **p** coming from a Backwards Projection rule.
This implies that:

1. **p** has the form: $(\wedge_{i=1, i\neq j}^{k} b^i) \otimes \beta \otimes b_2 \rightarrow \mathbf{neg}\, b^j \otimes \beta$
2. (F.1) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \mathbf{neg}\, b^j$
3. The following statements were derived in less than k steps:

[2]Recall that $\mathbf{neg}\,\mathbf{neg}\, f = f$

(a) $\mathbf{P}, \mathcal{S}, \mathbf{d}_n, \mathbf{d}_{n+1} \vdash \beta$
(b) $\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash (\wedge_{i=1, i \neq j}^{k} b^i)$
(c) $\mathbf{P}, \mathcal{S}, \mathbf{d}_{n+1} \vdash b_2$

(1) implies that there is a causal law in D of the form β `causes` $\mathtt{b_2}$ `if` $(\wedge_{\mathtt{i=1}}^{\mathtt{k}} \mathtt{b^i})$. From Item 3a it follows that there is a premise $\mathbf{d}_{n-1} \overset{\beta}{\rightsquigarrow} \mathbf{d}_n$ in $\mathcal{S}$. Thus, β `occurs_at` $\mathtt{s_n}$ is in D. By inductive hypothesis, $D \models \mathtt{b_2\ at\ s_{n+1}}$ and $D \models (\wedge_{\mathtt{i=1, i \neq j}}^{\mathtt{k}} \mathtt{b^i})$ `at` $\mathtt{s_n}$. Suppose by contradiction that $b^j \in act2st(sit2act(s_n))$. By the definition of Res, it must be the case that $b_2 \in E^+(act2st(sit2act(s_n)))$, and hence $D \models b_2$ `at` s_{n+1}. This contradicts the inductive hypothesis. Thus, it follows that $b^j \notin act2st(sit2act(s_n))$ and therefore $D \models \mathtt{b^j\ at\ s_n}$.

- ***Decomposition Rule***: Suppose (F.1) was derived via the Decomposition Rule rule. This implies that one of the following statements was derived in less than k steps:

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \psi \otimes \phi \quad \text{or} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \phi \otimes \psi$$

  where ϕ and ψ are serial conjunction of fluent literals. The two cases where $\phi = f_1 \otimes \cdots \otimes f_m$ or $\psi = f_1 \otimes \cdots \otimes f_m$ are completely symmetric. For concreteness assume that it is the first case. Then by the inductive hypothesis $D \models \phi \wedge \psi$ `at` $\mathtt{s_n}$. The inductive claim now follows by definition of satisfaction in $\mathcal{L}_1$.

- ***Sequencing Rule***: Suppose (F.1) was derived via the Sequencing Rule rule. This implies that the following statements were derived in less than k steps:

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \phi$$
$$\mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash \psi$$

  where $\phi \otimes \psi = f_1 \otimes \cdots \otimes f_m$. The inductive claim trivially follows from the inductive hypothesis. This concludes the soundness proof. □

Completeness with respect to Π_D

Recall that in the LP reduction of $\mathcal{L}_1$, the symbol $\neg$ is replaced by **neg**.

F.0.59. THEOREM. *(**Completeness with Respect to** Π_D) Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}_D^{PAD}$ reduction of D. Suppose $\Pi_D \models true\_after(f, r, s_i)$. Then $\mathbf{P}, \mathcal{S}, \mathbf{d}_i \models r \otimes f$.*

The proof uses a similar technique than the one used in [8] to prove soundness of their LP reduction Π. Relying in the fact that $\mathcal{TR}_D^{PAD}$ has a sound and complete reduction to LP, we can reformulate Theorem F.0.59 in terms of the LP reduction of $\mathcal{TR}_D^{PAD}$. Since we will use the mapping, $db2st_{\mathcal{S}}$, between database states and sets of `state`-terms in the completeness theorem; we repeat the definition of $db2st_{\mathcal{S}}$ here for convenient reference.

First we define $db2st_{\mathcal{S}}$, as a correspondence between database states and `state`-terms, as follows:

- $db2st_{\mathcal{S}}(\mathbf{d}) = s_{\mathbf{d}}$, if $\mathbf{d}$ occurs in a *run*- or *state*-premise in $\mathcal{S}$ and $\mathcal{S}$ has *no* *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, for some state $\mathbf{d}_0$. Here $s_{\mathbf{d}}$ is the unique $\mathcal{L}_{LP}$ state constant that corresponds to the $\mathcal{TR}_D^{PAD}$ state identifier $\mathbf{d}$ and α is a pda.
- $db2st_{\mathcal{S}}(\mathbf{d}) = Result(\alpha, s)$, if $\mathcal{S}$ has a *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, and $db2st_{\mathcal{S}}(\mathbf{d}_0) = s$.

F.0.60. Theorem. *Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}_D^{PAD}$ reduction of D, and let $\Gamma(\mathbf{P}, \mathcal{S})$ be the LP reduction of $\Lambda(D)$. Suppose $\Pi_D \models true\_after(f, r, t_i)$. Then $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i))$*

In order to prove Theorem F.0.60 we first prove the following lemmas. To avoid repeating the same statement again and again, we assume that for every database state $\mathbf{d}_i$, $db2st_{\mathcal{S}}(\mathbf{d}_i) = t_i$.

F.0.61. Lemma. *Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}_D^{PAD}$ reduction of D. Let I be a collection of formulas of the form $true\_at(f, s_i)$ s.t. $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, t_i)$. For any $0 \leq i \leq k$, if the program $H_i \cup I$[3] entails $true\_after(f, r, s_i)$ then $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i))$*

Proof. The proof is by induction on the length of r. Since $\Lambda(D)$ is sound with respect to D, and Γ is sound with respect to Λ, it follows that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, t_i)$ implies that $D \models$ `f at s`$_i$. Hence, by Proposition 4.1.12, the program $H_i \cup I$ has a unique answer set $\mathbf{S}$.

Base Case: Length of r is 0.
Suppose that $true\_after(f, [], s_i) \in \mathbf{S}$, then from the rule (e_1) in Π_D, we can conclude that $true\_at(f, s_i) \in \mathbf{S}$ as well. The previous fact implies that $true\_at(f, s_i) \in I$ or $true\_at(f, s_i) \in H_i$. In the former case the claim of the lemma follows from our assumption that $true\_at(f, s_i) \in I$ implies that

[3] Recall that H_i is the set of all ground instantiation of all rules of Π_D except for the Second Inertia Axiom (SI) and the Description of the Explicit Path (AP), not containing any other situation constants except s_i.

$\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, t_i)$. In the latter case, i.e., $true\_at(f, s_i) \in H_i$, by inspection of the rules in Π_D, and by the rule (BC) in Π_D, it follows that `f at s`$_i$ $\in D$. Therefore, we know that $\mathbf{d}_i \rhd f \in \mathcal{S}$. Thus, by the rule Premises in Γ, $Holds(f, t_i) \in \Gamma(\mathbf{P}, \mathcal{S})$. It follows that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, t_i)$.

Inductive Case: Suppose the claim of the lemma is true when the length of r is less than n. Let us prove that it is true when the length of r is n.

Since the length of r is greater then zero, let us assume $r = [a \mid r']$. Suppose that $true\_after(f, r, s_i) \in \mathbf{S}$. By inspection of the axioms in Π_D and Proposition 4.1.11, it is clear that $true\_after(f, r, s_i)$ could be derived via the *effect of actions* rule (e_2), or by the *inertia axioms* rule (i_1). We consider each case in turn:

- Suppose that $true\_after(f, r, s_i)$ was derived via a ground instantiation of the *effect of actions* rule of the form

  $$true\_after(f, [a \mid r'], s_i) :- causes(a, f, p), all\_true\_after(p, r', s_i)$$

  This implies that:

  - $causes(a, f, p) \in \mathbf{S}$, and
  - $true\_after(g, r', s_i) \in \mathbf{S}$ for every $g \in p$.

  From the inductive hypothesis we have that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(g, Result(r', t_i))$ for every $g \in p$. Since D contains `a causes f if p`, it follows that $p \otimes a \rightarrow a \otimes f \in \mathbf{P}$. Thus

  $$Holds(f, Result(a, Result(r', t_i))) :- Holds(p, Result(r', t_i)) \in \Gamma(\mathbf{P}, \mathcal{S})$$

  From the previous facts it follows that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i))$.

- Suppose that $true\_after(f, r, s_i)$ was derived via a ground instantiation of the *inertia axioms* rule of the form

  $$true\_after(f, [a \mid r'], s_i) :- true\_after(f, r', s_i), \mathbf{not}\, ab(f, a, r', s_i)$$

  This implies that:

  - $true\_after(f, r', s_i) \in \mathbf{S}$ and
  - $ab(f, a, r', s_i) \notin \mathbf{S}$

  From the inductive hypothesis we have that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r', t_i))$. In addition we know that $ab(f, a, r', s_i) \notin \mathbf{S}$. From this last fact, it follows that $one\_false\_after(p, r', s_i) \in \mathbf{S}$. This means that for any pair of action

and fluents a, p, if $causes(a, \mathbf{neg}\, f, p) \in \mathbf{S}$ then there is some $g \in p$ s.t. $true\_after(\mathbf{neg}\, g, r', s_i) \in \mathbf{S}$ (from i_2 and Proposition 4.1.11). Thus, if $causes(a, \mathbf{neg}\, f, p) \in S$, from the inductive assumption we can conclude that there is some $g \in p$ s.t. $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(\mathbf{neg}\, g, Result(r', t_i))$. By the Forward Disablement rule in $Frame(\mathbf{P})$ and the definition of Γ, it follows that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i))$. □

F.0.62. LEMMA. *Let* $0 \leq i \leq m \leq k$. *Suppose* $T_m \models true\_after(f, r, s_i)$[4]. *Then* $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i))$

Proof. By induction on m

Base Case: Length of m is 0.
Conclusion of the lemma follows immediately from Lemma F.0.61.

Inductive Case: Clearly,

$$T_m = T_{m-1} \cup (l_m \cup H_m). \tag{F.2}$$

Notice that by Proposition 4.1.13 and the definition of splitting set (c.f. Section 2.2) we can conclude that

- T_{m-1} has a unique answer set B_{m-1}
- T_m has an unique answer set B_m
- $U = head(T_{m-1})$ forms a splitting set for T_m

In particular, the bottom of T_m is T_{m-1} and the top is $(l_m \cup H_m)$. By Proposition 2.2.18, it follows that B_m is an answer set of T_m iff

$$B_m = B_{m-1} \cup C$$

where C is an answer set of the partial evaluation $e_U(l_m \cup H_m, B_{m-1})$[5] of $(l_m \cup H_m)$ with respect to B_{m-1}.

By inductive hypothesis we have that

$$true\_after(f, r, s_i) \in B_{m-1} \quad \text{then} \quad \Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i)) \tag{F.3}$$

[4] Recall that $T_m = H_0 \cup (l_1 \cup H_1) \cup \cdots \cup (l_m \cup H_m)$ where l_i $(0 \leq 0 \leq k)$ is the rule $true\_at(f, s_i) : -true\_after(f, [a_{i-1}], s_{i-1})$

[5] Recall that $e_U(\Pi, X)$ is defined as follows. For each rule $r \in \Pi$ such that $(pos(r) \cap U) \subset X$ and $(neg(r) \cap U) \cap X = \emptyset$ put in $e_U(\Pi, X)$ all the rules r' that satisfy the following property $head(r') = head(r), pos(r') = pos(r) \setminus U, neg(r') = neg(r) \setminus U$

Therefore, to prove the claim of the theorem it is enough to prove that

$$true\_after(f, r, s_i) \in C \quad \text{then} \quad \Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i)) \tag{F.4}$$

which is equivalent to

$$\begin{gathered} e_U(l_m \cup H_m, B_{m-1}) \models true\_after(f, r, s_i) \\ \text{then} \\ \Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i)) \end{gathered} \tag{F.5}$$

Observe that since U is a splitting set of T_m, and all the facts in the body of the rules in $T_{m-1} \cup l_m$ are in U, it is easy to see that

$$e_U(l_m \cup H_m, B_{m-1}) = I_m \cup H_m \tag{F.6}$$

where

$$I_m = \{true\_at(f, s_m) : T_{m-1} \models true\_after(f, [a_{m-1}, s_{m-1}])\}$$

In other words, the partial evaluation consists of the rules in H_m together the head of the rules in $T_{m-1} \cup l_m$ which are satisfied by B_{m-1}. By inductive hypothesis if

$$I_m \models true\_after(f, r, s_i) \quad \text{then} \quad \Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t_i)) \tag{F.7}$$

Therefore, I_m satisfies the conditions of Lemma F.0.61 and hence

$$\text{if} \;\; I_m \cup H_m \models true\_at(f, s_m), \;\; \text{then } \Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(a_{i-1}, t_{i-1})) \tag{F.8}$$

This, together with (F.6) and (F.7) above, prove F.5 and therefore the claim of this Lemma. □

Now we are ready to prove our theorem stated at the beginning of this section.

F.0.63. THEOREM. *Let D be a simple domain description. Let $\Lambda(D) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{TR}_D^{PAD}$ reduction of D, and let $\Gamma(\mathbf{P}, \mathcal{S})$ be the LP reduction of $\Lambda(D)$. Suppose $\Pi_D \models true\_after(f, r, s_i)$. Then $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, s))$*

Proof. It is easy to see that T_k as defined in Lemma F.0.62 is the same as Π_1[6]. Hence from Lemma F.0.62 it follows that, $\Pi_1 \models true\_after(f, r, s)$ implies that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t))$. By Proposition 4.1.11 Π_1 and Π_D are equivalent. Hence $\Pi_D \models true\_after(f, r, s)$, then $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, Result(r, t))$ □

[6] Recall that Π_1 is the program obtained from Π_D by replacing (SI) and (AP) by $true\_at(F, s_i) : -true\_after(F, a_{i-1}, s_{i-1})$

Appendix G

Proofs of the reduction of PSs semantics to $\mathcal{TR}^{PAD}$

In this appendix we prove soundness of the reduction of the semantics of production systems to $\mathcal{TR}^{PAD}$ developed in Section 5.4.

In the remainder let $\mathsf{PS} = (\mathcal{T}, \mathsf{L}, R)$ be a production system, WM_0 a consistent[1] initial working memory, and $\Lambda_{\mathsf{PS}} = (\mathcal{E}, \mathbf{P}, \mathcal{S})$ the $\mathcal{TR}^{PAD}$ embedding of PS.

For simplicity we present a ground version of the proofs. Lifting to the non-ground case is done in a standard way (cf. [11]). Since in this setting we do not have variables, we will disregard variable assignments.

G.0.64. Definition. We denote $\Lambda^{\text{elem}}_{\mathsf{PS}}$ the specification $(\mathcal{E}, \mathbf{P}', \mathcal{S})$ where the transaction base $\mathbf{P}' \subset \mathbf{P}$ consists of:

- The rules encoding the ontology (c.f. Item 2 in Section 5.4); and
- The inconsistency rules (c.f. Item 7 in Section 5.4).

We will usually omit the subscript in $\Lambda^{\text{elem}}_{\mathsf{PS}}$. □

Intuitively, $\Lambda^{\text{elem}}_{\mathsf{PS}}$ filters out from Λ_{PS} all the rules that do not affect transitions: the Horn rules for the production rules, and the auxiliary rules for random actions.

G.0.65. Definition. Let $\mathcal{M}$ be a Herbrand path structure in $\mathcal{L}_{\text{PS}}$, WM a working memory, and $\mathbf{d}$ a state identifier We define the sets $\mathsf{facts}_{\mathcal{M}}(\mathbf{d})$, $\mathsf{facts}(\mathbf{d})$,

[1] Recall that a working memory WM is consistent with the ontology $\mathcal{T}$ if there is a $\mathcal{T}$-structure whose working memory is WM.

$\mathsf{facts}(\mathsf{WM}), \mathsf{inertials}(\mathbf{d})$ and $\mathsf{inertials}_{\mathcal{M}}(\mathbf{d})$ to be the following set of PS fluents.

$$\begin{array}{ll}
\mathsf{facts}_{\mathcal{M}}(\mathbf{d}) & = \{f \mid \mathcal{M}(\langle \mathbf{d} \rangle) \models f \text{ and } f \in \mathcal{L}_{\mathrm{PS}}\} \\
\mathsf{facts}(\mathbf{d}) & = \{f \mid \mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models f \text{ and } f \in \mathcal{L}_{\mathrm{PS}}\} \\
\mathsf{facts}(\mathsf{WM}) & = \{f \mid \mathcal{T}, \mathsf{WM} \models f \text{ and } f \text{ is an atom in } \mathcal{L}_{\mathrm{PS}}\} \\
\mathsf{inertials}(\mathbf{d}) & = \{f \mid \mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models \mathit{inertial}(f)\} \\
\mathsf{inertials}_{\mathcal{M}}(\mathbf{d}) & = \{f \mid \mathcal{M}\langle \mathbf{d} \rangle \models \mathit{inertial}(f)\}
\end{array}$$

□

G.0.66. Lemma (Initial Equivalence).

$$\begin{array}{l}
\mathit{facts}(\mathsf{WM}_0) \equiv \mathit{facts}(\mathbf{d}_0) \\
\mathit{inertials}(\mathbf{d}_0) = \mathsf{WM}_0
\end{array} \tag{G.1}$$

Proof. Recall that the well-founded model of a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is defined as the limit of the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)\}$ as stated in Definition 3.7.14. The proof of this lemma is by transfinite induction on the iterations of the Γ operator. We show that for every model $\mathcal{M}_i = \Gamma^{\uparrow i}(\mathbf{I}_\emptyset)$, claim (G.1) holds.

Let $\mathbf{I}_\emptyset$ be a Herbrand path structure such that for every literal h and path π

$$\mathbf{I}_\emptyset(\pi)(h) = \mathbf{u}$$

$\Gamma^{\uparrow 1}(\mathbf{I}_\emptyset)$ The quotient of $\Lambda^{\mathrm{elem}}_{\mathsf{PS}}$ (Λ from now on) by $\mathbf{I}_\emptyset$, $\Lambda_1 = \frac{\Lambda}{\mathbf{I}_\emptyset}$, is composed by the following premises, rules and PADs:

1. **Ontology, Inconsistency rules, Initial Database,and Actions**: remain as in Λ since they are **not**-free.
2. **Frame Axioms**: For every path π, predicate $p \in \mathcal{P}$ and *pda* α, the frame axioms have one of the following forms

$$\left.\begin{array}{r}
\mathit{inertial}(p(\vec{c})) \wedge p(\vec{c}) \otimes \alpha \otimes \mathbf{u}^\pi \rightarrow \\
\qquad \alpha \otimes p(\vec{c}) \wedge \mathit{inertial}(p(\vec{c})) \\
\mathit{inertial}(p(\vec{c})) \wedge p(\vec{c} \wedge \vec{c} \neq \vec{e}) \otimes \alpha \otimes \mathbf{u}^\pi \rightarrow \\
\qquad \alpha \otimes p(\vec{c}) \wedge \mathit{inertial}(p(\vec{c}))
\end{array}\right\} \in \mathcal{E}$$

   where the vector $\vec{e}$ is the one used to ground α.

Since Λ_1 is **not**-free, by Theorem E.0.39 we know that it has a unique least model

$$\mathcal{M}_1 = LPM\,(\Lambda_1)$$

Let $F_{\mathcal{S}} = \{f \mid \mathbf{d}_0 \rhd f \in \mathcal{S}\}$ and $\mathsf{dtg}(\mathcal{T})$ the Datalog program encoding the ontology. From construction of Λ, we know that for every ground fluent f

$$\begin{array}{l}
f \in F_{\mathcal{S}} \text{ iff } f \in \mathsf{WM}_0 \\
f \in \mathsf{inertials}_{\mathcal{M}_1}(\mathbf{d}_0) \text{ iff } f \in \mathsf{WM}_0
\end{array}$$

Now, to continue the proof of the lemma, we explore the cases where the fluents predicate symbols are in $\mathcal{P}_{\text{DL}}$ and $\mathcal{P}_{\text{PS}}$.

- Let g be a $\mathcal{P}_{\text{DL}}$ literal. Since
  - there are no PAD in $\mathcal{E}$ that has pre-effects, and
  - our DL language is Datalog rewritable (c.f. Definition 5.1.1),

  it follows that

$$F_{\mathcal{S}} \cup \mathsf{dtg}(\mathcal{T}) \models g \;\text{ iff }\; \mathcal{T}, \mathsf{WM}_0 \models g$$

- Let g be a $\mathcal{P}_{\text{PS}}$ literal. Since there are no PAD in $\mathcal{E}$ that has pre-effects, and by definition of satisfaction in PS, it follows that

$$F_{\mathcal{S}} \cup \mathsf{dtg}(\mathcal{T}) \models g \;\text{ iff }\; \mathcal{T}, \mathsf{WM}_0 \models g \;\text{ iff }\; g \in \mathsf{WM}_0$$

From the previous facts we can conclude that

$$\mathsf{facts}_{\mathcal{M}_1}(\mathbf{d}_0) \equiv \mathsf{facts}(\mathsf{WM}_0)$$
$$\mathsf{inertials}_{\mathcal{M}_1}(\mathbf{d}_0) = \mathsf{WM}_0$$

Inductive Claim $n = k + 1$ **(successor ordinal):** Suppose that the claim of the lemma holds for the first k iterations

$\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)$**:** Let $\mathcal{M}_n = \Gamma^{\uparrow n}(\mathbf{I}_\emptyset)$. Let $\mathcal{M}_n^+$ and $\mathcal{M}_1^-$ be the least models of Λ_n^+ and Λ_n^- respectively as stated in Definition E.0.38. As shown in [31], the least model $\mathcal{M}_n$ of Λ_n is defined as follows: For any path π, and **not**-free literal f

$$\begin{array}{lll} \mathcal{M}_n(\pi)(f) = \mathbf{t} & \text{iff} & \mathcal{M}_n^-(\pi)(f) = \mathbf{t} \\ \mathcal{M}_n(\pi)(f) = \mathbf{f} & \text{iff} & \mathcal{M}_n^+(\pi)(f) = \mathbf{f} \\ \mathcal{M}_n(\pi)(f) = \mathbf{u} & \text{iff} & \text{otherwise} \end{array}$$

Following a similar reasoning as above, we can see that the truth value of fluents in $\mathcal{M}_n^-(\mathbf{d}_0)$ and $\mathcal{M}_n^+(\mathbf{d}_0)$ depend only on the premises and $\mathsf{dtg}(\mathcal{T})$.

Since *(i)* the premises and $\mathsf{dtg}(\mathcal{T})$ stay as they are after taking the quotient, $\frac{\Lambda}{\Gamma^{\uparrow n-1}(\mathbf{I}_\emptyset)}$, and *(ii)* the claim of the lemma follows for $\Gamma^{\uparrow n-1}(\mathbf{I}_\emptyset)$ by inductive assumption, we can conclude that the claim follows for $\mathcal{M}_n$ as well.

Inductive Claim (n, limit ordinal): If n is a limit ordinal, then

$$\Xi^{\uparrow n}(\mathbf{I}_0) = \bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0)$$

By inductive hypothesis we know that for every $0 < j < n$

$$\begin{array}{l} \mathsf{facts}(\mathsf{WM}_0) \equiv \mathsf{facts}_{\Xi^{\uparrow j}(\mathbf{I}_0)}(\mathbf{d}_0) \\ \mathsf{inertials}_{\Xi^{\uparrow j}(\mathbf{I}_0)}(\mathbf{d}_0) = \mathsf{WM}_0 \end{array} \tag{G.2}$$

Then it is clear that

$$\begin{array}{l} \mathsf{facts}(\mathsf{WM}_0) \equiv \mathsf{facts}_{\bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0)}(\mathbf{d}_0) \\ \mathsf{inertials}_{\bigcup_{j \leq n} \Xi^{\uparrow j}(\mathbf{I}_0)}(\mathbf{d}_0) = \mathsf{WM}_0 \end{array} \tag{G.3}$$

Thus,

$$\begin{array}{l} \mathsf{facts}(\mathsf{WM}_0) \equiv \mathsf{facts}_{\Xi^{\uparrow n}(\mathbf{I}_0)}(\mathbf{d}_0) \\ \mathsf{inertials}_{\Xi^{\uparrow n}(\mathbf{I}_0)}(\mathbf{d}_0) = \mathsf{WM}_0 \end{array} \tag{G.4}$$

□

To avoid tedious repetitions and to ease the presentations of the proofs, in the reminder of the paper we will assume that the frame axioms have the general form:

$$\begin{array}{l} \mathit{inertial}(p(\vec{c})) \wedge p(\vec{c}) \otimes \alpha \otimes \mathbf{not}\ \mathit{inconsistent} \rightarrow \\ \qquad \alpha \otimes p(\vec{c}) \wedge \mathit{inertial}(p(\vec{c})) \end{array}$$

Observe that we do not lose generality with this assumption, since after the grounding the second type of frame axiom can be seen as the PAD above.

G.0.67. Lemma (Soundness of Actions). *Let β be a sequence of ground actions and α an action. Let $\mathbf{d}_\beta$ and $\mathbf{d}_{\beta,\alpha}$ be database identifiers. Then there is a working memory WM_i such that::*

1. $\mathsf{WM}_0 \overset{\beta}{\twoheadrightarrow} \mathsf{WM}_i$,
2. $\mathit{facts}(\mathbf{d}_\beta) \subseteq \mathit{facts}(\mathsf{WM}_i)$,
3. *If $\alpha = \mathsf{Ins}_p(\vec{c})$ then*

$$\begin{array}{ll} \mathit{facts}(\mathbf{d}_{\beta,\alpha}) & \subseteq \mathit{facts}((\mathsf{WM}_i \cup \{p(\vec{c})\}) \setminus \{\mathbf{neg}\, p(\vec{c})\}) \\ \mathit{inertials}(\mathbf{d}_{\beta,\alpha}) & \subseteq (\mathsf{WM}_i \cup \{p(\vec{c})) \setminus \{\mathbf{neg}\, p(\vec{c})\} \end{array} \tag{G.5}$$

4. *If $\alpha = \mathsf{del}_p(\vec{c})$ and $p \in \mathcal{P}_{DL}$ then*

$$\begin{array}{ll} \mathit{facts}(\mathbf{d}_{\beta,\alpha}) & \subseteq \mathit{facts}((\mathsf{WM}_i \cup \{\mathbf{neg}\, p(\vec{c})\}) \setminus \{p(\vec{c})\}) \\ \mathit{inertials}(\mathbf{d}_{\beta,\alpha}) & \subseteq (\mathsf{WM}_i \cup \{\mathbf{neg}\, p(\vec{c})) \setminus \{p(\vec{c})\} \end{array} \tag{G.6}$$

5. *If $\alpha = \mathsf{del}_p(\vec{c})$ and $p \in \mathcal{P}_{PS}$ then*

$$\begin{array}{ll} \mathit{facts}(\mathbf{d}_{\beta,\alpha}) & \subseteq \mathit{facts}(\mathsf{WM}_i \setminus \{p(\vec{c})\}) \\ \mathit{inertials}(\mathbf{d}_{\beta,\alpha}) & \subseteq \mathsf{WM}_i \setminus \{p(\vec{c})\} \end{array} \tag{G.7}$$

Proof. Recall that the well-founded model of a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is defined as the limit of the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)\}$ as stated in Definition 3.7.14. The proof of this lemma consists in showing that for every model $\mathcal{M}_i = \Gamma^{\uparrow i}(\mathbf{I}_\emptyset)$ the claim of the lemma holds. However, since we show that we reach a fix point after the second iteration, it is enough to show that the claim holds in the first two steps of of the construction of the well-founded model. In the following, we will omit the subscript and superscript in $\Lambda^{\text{elem}}_{\mathsf{PS}}$, and refer to it as Λ.

Let $\mathbf{I}_\emptyset$ be an Herbrand path structure such that for every literal h and path π:

$$\mathbf{I}_\emptyset(\pi)(h) = \mathbf{u}$$

$\Gamma^{\uparrow 1}(\mathbf{I}_\emptyset)$ The quotient of Λ by $\mathbf{I}_\emptyset$, $\Lambda_1 = \frac{\Lambda}{\mathbf{I}_\emptyset}$, is composed by the following rules and PADs:

1. **Ontology, Inconsistency rules, Initial Database, and Actions**: remain untouched since they are **not**-free.
2. **Frame Axioms**: For every path π, *pda* α, and predicate $p \in \mathcal{L}_{\mathsf{PS}}$, the frame axioms have the form

$$\left.\begin{array}{r} \mathit{inertial}(p(\vec{X})) \otimes p(\vec{X}) \otimes \alpha \otimes \mathbf{u}^\pi \rightarrow \\ \alpha \otimes p(\vec{X}) \otimes \mathit{inertial}(p(\vec{X})) \end{array}\right\} \in \mathcal{E}$$

Since Λ_1 is **not**-free, it has a unique least model

$$\mathcal{M}_1 = LPM\,(\Lambda_1)$$

Let $\mathcal{M}_1^+$ and $\mathcal{M}_1^-$ be the least models of Λ_1^+ and Λ_1^- respectively as stated in Definition E.0.38. By Theorem E.0.39, the least model $\mathcal{M}_1$ of Λ_1 is defined as follows: For any path π, and **not**-free literal f

$$\begin{array}{lll} \mathcal{M}_1(\pi)(f) = \mathbf{t} & \text{iff} & \mathcal{M}_1^-(\pi)(f) = \mathbf{t} \\ \mathcal{M}_1(\pi)(f) = \mathbf{f} & \text{iff} & \mathcal{M}_1^+(\pi)(f) = \mathbf{f} \\ \mathcal{M}_1(\pi)(f) = \mathbf{u} & \text{iff} & \text{otherwise} \end{array}$$

Now we prove the claim of the lemma for the first iteration by induction on the length n of the sequence of actions β, α.

Base Case $n = 0$**:** Follows from Lemma G.0.66.

Inductive Claim $n = k + 1$**:** Suppose that the claim of the lemma holds for all sequences β such that its length is smaller than n

Inductive Case $n+1$ (β, α) By definition of $\mathbf{I}_\emptyset$

$$\mathbf{I}_\emptyset(\mathbf{d}_{\beta,\alpha})(\mathit{inconsistent}) = \mathbf{u}$$

Therefore, we can conclude that for any fluent g and action α, the frame axioms for the path $\langle \mathbf{d}_i, \mathbf{d}_{i+1} \rangle$ have the form:

$$\left.\begin{array}{r} \mathit{inertial}(g) \otimes g \otimes \alpha \otimes \mathbf{u}^{\langle \mathbf{d}_i, \mathbf{d}_{i+1} \rangle} \rightarrow \\ \alpha \otimes g \otimes \mathit{inertial}(g) \end{array}\right\} \in \Lambda_1$$

Recall that all the rules and PADs with $\mathbf{u}^\pi$ in the body are removed from Λ_1^-. This means that Λ_1^- does not contain any frame axiom. Thus, for any fluent $q(\vec{w})$, if $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(q(\vec{w})) = \mathbf{t}$ (and therefore ${\mathcal{M}_1}^-(\mathbf{d}_{\beta,\alpha})(q(\vec{w})) = \mathbf{t}$) then $q(\vec{w})$ must be a consequence of α together with $\mathsf{dtg}(\mathcal{T})$.

We consider each possible action α in turn:

1. Suppose $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{PS}}$. Observe that neither the frame axioms nor α can assert neither $\mathcal{L}_{\mathrm{PS}}$ facts nor inertial facts. From the previous fact we can conclude that

   $$\mathsf{inertials}_{\mathcal{M}_1}(\mathbf{d}_{\beta,\alpha}) = \emptyset \subseteq \mathsf{WM}_i \setminus \{p(\vec{c})\}$$

   Thus, for every fluent f in $\mathcal{L}_{\mathrm{PS}}$, if $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(f) = \mathbf{t}$, it means that the Datalog program $\mathsf{dtg}(\mathcal{T})$ entails f. This yields that

   $$\mathsf{facts}_{\mathcal{M}_1}(\mathbf{d}_{\beta,\alpha}) = \{f \mid \mathsf{dtg}(\mathcal{T}) \text{ entails } f\} \subseteq \mathsf{facts}(\mathsf{WM}_i \setminus \{p(\vec{c})\})$$

   and from the inductive hypothesis and the definition of rule application it follows that

   $$\mathsf{WM}_0 \overset{\beta}{\twoheadrightarrow} \mathsf{WM}_i \overset{\mathit{del}_p^{\mathrm{DL}}(\vec{c})}{\twoheadrightarrow} \mathsf{WM}_i \setminus \{p(\vec{c})\}$$

2. Suppose $\alpha = \mathit{Ins}_p(\vec{c})$. since there are no frame axioms in Λ_1^-, for any fluent $q(\vec{w})$ if $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(q(\vec{w})) = \mathbf{t}$, it follows that either
   - $q(\vec{w}) = p(\vec{c})$ or
   - $\{p(\vec{c})\} \cup \mathsf{dtg}(\mathcal{T}) \models q(\vec{w})$.

   In any of the two cases above, together with the observation that only α can assert inertial facts, we can conclude that

   $$\begin{array}{l} \mathsf{facts}_{\mathcal{M}_1}(\mathbf{d}_{\beta,\alpha}) \subseteq \mathsf{facts}(\{p(\vec{c})\}) \subseteq \mathsf{WM}_i \cup \{p(\vec{c})\} \\ \mathsf{inertials}_{\mathcal{M}_1}(\mathbf{d}_{\beta,\alpha}) = \{p(\vec{c})\} \subseteq \mathsf{WM}_i \cup \{p(\vec{c})\} \end{array}$$

   and from the inductive hypothesis and the definition of rule application it follows that

   $$\mathsf{WM}_0 \overset{\beta}{\twoheadrightarrow} \mathsf{WM}_i \overset{\mathit{Ins}_p^{\mathrm{DL}}(\vec{c})}{\twoheadrightarrow} \mathsf{WM}_i \cup \{p(\vec{c})\}$$

3. If $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{DL}}$, the proof is analogous to the case for *Ins* above.

This proves that claim of the lemma follows for $\Gamma^{\uparrow 1}(\mathbf{I}_\emptyset)$.

$\Gamma^{\uparrow 2}(\mathbf{I}_\emptyset)$ Now take $\Lambda_2 = \frac{\Lambda}{\mathcal{M}_1}$, and let $\mathcal{M}_2^+$ and $\mathcal{M}_2^-$ be the models of Λ_2^+ and Λ_2^- respectively. As before, we prove the claim of the lemma by induction on the length n of β, α.

Base case $n = 0$**:** Follows from Lemma G.0.66.

Inductive case $n + 1$ (β, α)**:** We explore the cases where $\alpha = \mathit{Ins}_p(\vec{c})$ and $\alpha = \mathit{del}_p(\vec{c})$. To know which fluents are affected by the frame axioms in the path $\langle \mathbf{d}_\beta, \mathbf{d}_{\beta,\alpha} \rangle$, in each case we have to further explore all the possible truth values of the fluent *inconsistent* in $\mathcal{M}_1$.

Now, we explore each case in turn:

1. Suppose $\alpha = \mathit{Ins}_p(\vec{c})$. Now let us consider the possible truth values of the fluent *inconsistent* in $\mathcal{M}_1$.
   (a) Suppose that $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models \mathit{inconsistent}$ and therefore $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(\mathbf{not}\ \mathit{inconsistent})$=$\mathbf{f}$. This means that every frame axiom for the path $\langle \mathbf{d}_\beta, \mathbf{d}_{\beta,\alpha} \rangle$ is removed from Λ when we take the quotient modulo $\mathcal{M}_1$. In addition, observe that

$$\mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models \mathit{inconsistent} \text{ iff } \Lambda_1^-, \mathbf{d}_{\beta,\alpha} \models \mathit{inconsistent}$$

   Since Λ_1^- does not contain any frame axiom in that path $\langle \mathbf{d}_\beta, \mathbf{d}_{\beta,\alpha} \rangle$, this means that the state $\mathbf{d}_{\beta,\alpha}$ is inconsistent independently of the frame axioms. Moreover, the truth value of the fluents in $\mathcal{L}_{\mathrm{PS}}$ for the path $\langle \mathbf{d}_\beta, \mathbf{d}_{\beta,\alpha} \rangle$ depend only on the action α and $\mathsf{dtg}(\mathcal{T})$. That is, for any fluent $q(\vec{w})$

$$\begin{array}{c} \text{if } \ \mathcal{M}_2(\mathbf{d}_{\beta,\alpha})(q(\vec{w})) = \mathbf{t} \ \ \mathit{then} \\ \text{either } \ p(\vec{c}) = q(\vec{w}) \ \text{ or } \ \{p(\vec{c})\} \cup \mathsf{dtg}(\mathcal{T}) \models q(\vec{w}). \end{array}$$

   From the previous facts we can conclude that

$$\mathcal{M}_2(\mathbf{d}_{\beta,\alpha}) \models q(\vec{w}) \text{ iff } \mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models q(\vec{w})$$

   Thus, since the claim of the lemma follows for $\mathcal{M}_1$, we know that the claim follows for $\mathcal{M}_2$.

(b) Suppose that $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models \textbf{not}\ \textit{inconsistent}$. Observe that:

$$\mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models \textbf{not}\ \textit{inconsistent}\ \text{iff}\ \Lambda_1^+, \mathbf{d}_{\beta,\alpha} \models \textbf{not}\ \textit{inconsistent}$$

Since Λ_1^+ contains all the frame axioms in the path $\langle \mathbf{d}_\beta, \mathbf{d}_{\beta,\alpha} \rangle$, this means that there is no inconsistency between the action α, the ontology, and all the frame axioms. It follows that for every fluent g, except $\textbf{neg}\, p(\vec{c})$. the quotient of Λ by $\mathcal{M}_1$ contains the frame axioms:[2]

$$\left.\begin{array}{r} \textit{inertial}(g) \otimes g \otimes \textit{Ins}_p(\vec{c}) \otimes \mathbf{t}^{\langle \mathbf{d}_i, \mathbf{d}_{i+1} \rangle} \rightarrow \\ \textit{Ins}_p(\vec{c}) \otimes g \otimes \textit{inertial}(g) \end{array}\right\} \in \mathcal{E}\ \text{of}\ \Lambda_2$$

Thus,

$$\text{if}\ \mathcal{M}_2(\mathbf{d}_\beta) \models \textit{inertial}(g) \wedge g\ \text{then}\ \mathcal{M}_2(\mathbf{d}_{\beta,\alpha}) \models \textit{inertial}(g) \wedge g$$

The previous fact, together with the inductive hypothesis and soundness of $\mathsf{dtg}(\mathcal{T})$ with respect to $\mathcal{T}$, implies that

$$\begin{array}{l} \mathsf{facts}_{\mathcal{M}_2}(\mathbf{d}_{\beta,\alpha}) \subseteq \mathsf{facts}(\mathsf{WM}_i \cup \{p(\vec{c})\}) \\ \mathsf{inertials}_{\mathcal{M}_2}(\mathbf{d}_{\beta,\alpha}) \subseteq (\mathsf{WM}_i \cup \{p(\vec{c})\}) \end{array}$$

and from the inductive hypothesis and the definition of rule application it follows that

$$\mathsf{WM}_0 \overset{\beta}{\Rightarrow} \mathsf{WM}_i \overset{\textit{Ins}_p(\vec{c})}{\Rightarrow} \mathsf{WM}_i \cup \{p(\vec{c})\}$$

(c) Suppose that $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(\textit{inconsistent}) = \mathbf{u}$. Observe as that

$$\begin{array}{c} \mathcal{M}_1(\mathbf{d}_{\beta,\alpha})(\textit{inconsistent}) = \mathbf{u} \\ \text{iff} \\ \Lambda_1^-, \mathbf{d}_{\beta,\alpha} \models \textit{inconsistent}\ \ \text{and}\ \ \Lambda_1^+, \mathbf{d}_{\beta,\alpha} \models \textbf{not}\ \textit{inconsistent} \end{array}$$

This means that the state $\mathbf{d}_{\beta,\alpha}$ is inconsistent because of the frame axioms, since Λ_1^+ and Λ_1^- agree on everything else. The rest of the proof is analogous to the proof in Item 1a above.

2. Suppose $\alpha = \textit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{DL}}$. This case completely symmetric to the case 1 above.
3. Suppose that $\alpha = \textit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{PS}}$. It is easy to see that $\mathcal{M}_2(\mathbf{d}_{\beta,\alpha}) \models q(\vec{w})$ if and only if
   - $p(\vec{c}) \neq q(\vec{w})$

[2] Observe that since we are considering the ground version of the reduction, the two different types of frame axioms collapse in one

- $\mathcal{M}_1(\mathbf{d}_{\beta,\alpha}) \models$ **not** *inconsistent*, and
- $\mathcal{M}_1(\mathbf{d}_{\beta}) \models q(\vec{w}) \otimes \mathit{inertial}(q(\vec{w}))$

The claim follows by a similar reasoning as above using the previous facts, the inductive hypotheses, and the definition of action application.

This concludes the proof that the claim of the lemma follows for $\Gamma^{\uparrow 2}(\mathbf{I}_{\emptyset})$.

$\Gamma^{\uparrow 3}(\mathbf{I}_{\emptyset})$**:** Next we show that $\mathcal{M}_2$ is a fixpoint of Γ. Take $\Lambda_3 = \frac{\Lambda}{\mathcal{M}_2}$, and let $\mathcal{M}_3$, $\mathcal{M}_3^+$ and $\mathcal{M}_3^-$ be the least models of Λ_3, Λ_3^+ and Λ_3^- respectively. We need to show that for every path π

$$\mathcal{M}_2(\pi) = \mathcal{M}_3(\pi)$$

Observe that for every n-path abstraction π ($n > 1$) and literal g, if $\mathcal{M}_2(\pi) \models g$ it means that

- g is an action atom
- π has the form $\langle \mathbf{d}_1 \mathbf{d}_2 \rangle$, and there is a *run*-premise of the form $\mathbf{d}_1 \overset{g}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$

Since *run*-premises remain unchanged after taking the quotient of Λ, the claim of the lemma follows straightforwardly for n-paths. Now we consider paths of length 1.

We prove that for every database identifier $\mathbf{d}$ and sequence of actions ξ.

$$\mathcal{M}_2(\mathbf{d}_{\xi}) = \mathcal{M}_3(\mathbf{d}_{\xi}) \tag{G.8}$$

The proof is by induction on the length n of ξ

Base Case $n = 0$**:** Follows from Lemma G.0.66.

Inductive Claim $n = k + 1$**:** Suppose that

$$\mathcal{M}_2(\mathbf{d}_{\xi}) = \mathcal{M}_3(\mathbf{d}_{\xi})$$

for all sequences ξ of length smaller than n

Inductive Case (n) Follows by a simple reasoning by cases as above, exploring the case where *inconsistent* is $\mathbf{t}, \mathbf{f}$ or $\mathbf{u}$ in $\mathcal{M}_2(\mathbf{d}_{\beta,\alpha})$, and $\alpha = \mathit{Ins}(f)$, or $\alpha = \mathit{del}(f)$. Intuitively, when *inconsistent* is $\mathbf{t}$ or $\mathbf{u}$ in $\mathcal{M}_2(\mathbf{d}_{\beta,\alpha})$, only the effect of the action (if any) and the consequences from the ontology are true in both, $\mathcal{M}_2(\mathbf{d}_{\beta,\alpha})$, and $\mathcal{M}_3(\mathbf{d}_{\beta,\alpha})$. On the other hand, if *inconsistent* is $\mathbf{f}$ in $\mathcal{M}_2(\mathbf{d}_{\beta,\alpha})$, by applying inductive hypothesis, using the frame axioms, and the definition of α one can prove (G.8).

This concludes the proof of the lemma. □

G.0.68. Lemma (State Equivalence). *Let $\Lambda^{elem} = (\mathcal{E}, \mathbf{P}, \mathcal{S})$, β be a sequence of ground actions and α a ground action. Suppose that the following holds:*

(1) $\mathsf{WM}_i \equiv \mathit{inertials}(\mathbf{d}_\beta)$
(2) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_\beta \models \mathbf{not}\ \mathsf{inconsistent}$
(3) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models \mathbf{not}\ \mathsf{inconsistent}$

Then

1. *If $\alpha = \mathit{Ins}_p(\vec{c})$ then*
$$\mathsf{WM}_i \cup \{p(\vec{c})\} \setminus \{\mathbf{neg}\, p(\vec{c})\} \equiv \mathit{inertials}(\mathbf{d}_{\beta,\alpha})$$
2. *If $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{DL}$ then*
$$\mathsf{WM}_i \cup \{\mathbf{neg}\, p(\vec{c})\} \setminus \{p(\vec{c})\} \equiv \mathit{inertials}(\mathbf{d}_{\beta,\alpha})$$
3. *If $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{PS}$ then*
$$\mathsf{WM}_i \setminus \{p(\vec{c})\} \equiv \mathit{inertials}(\mathbf{d}_{\beta,\alpha})$$

Proof. Let $\mathcal{M}$ be the well-founded model of Λ^{elem} (Λ from now on). We explore the cases where $\alpha = \mathit{Ins}_q(\vec{w})$ and $\alpha = \mathit{del}_p(\vec{c})$.

1. Suppose $\alpha = \mathit{Ins}_p(\vec{c})$. By hypothesis $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models \mathbf{neg}\, \mathit{inconsistent}$, and since $\alpha = \mathit{Ins}_p(\vec{c})$, we have that for every fluent $g \neq \mathbf{neg}\, p(\vec{c})$
$$\left.\begin{array}{r} \mathit{inertial}(g) \otimes g \otimes \mathit{Ins}_p(\vec{c}) \otimes \mathbf{t}^{\langle \mathbf{d}_i, \mathbf{d}_{i+1} \rangle} \rightarrow \\ \mathit{Ins}_p(\vec{c}) \otimes g \otimes \mathit{inertial}(g) \end{array}\right\} \in \frac{(\mathcal{E}, \mathbf{P}, \mathcal{S})}{\mathcal{M}} \tag{G.9}$$
Recall that by the hypothesis
$$\mathsf{inertials}(\mathbf{d}_\beta) = \mathsf{WM}_i$$
From the previous facts it follows that for every fluent $g \neq \mathbf{neg}\, p(\vec{c})$
$$g \in \mathsf{WM}_i \text{ implies that } \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models \mathit{inertial}(g) \tag{G.10}$$
And the definition of $\mathit{Ins}_p(\vec{c})$ yields that
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models \mathit{inertial}(p(\vec{c})) \tag{G.11}$$
From Lemma G.0.67 it follows that for every fluent g
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models \mathit{inertial}(g) \text{ implies that } g \in \mathsf{WM}_i \cup \{p(\vec{c})\} \setminus \{\mathbf{neg}\, p(\vec{c})\}$$
The previous fact together with (G.10)and (G.11) above, imply that
$$\mathsf{inertials}(\mathbf{d}_{\beta,\alpha}) = \mathsf{WM}_i \cup \{p(\vec{c})\} \setminus \{\mathbf{neg}\, p(\vec{c})\}$$

2. Suppose that $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{DL}}$. This case completely symmetric to the case 1 above.

3. Suppose that $\alpha = \mathit{del}_p(\vec{c})$ and $p \in \mathcal{P}_{\mathrm{PS}}$. Since by hypothesis $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{\beta,\alpha} \models$ **neg** *inconsistent*, for every fluent $g \neq p(\vec{c})$.

$$\left.\begin{array}{r} \mathit{inertial}(g) \otimes g \otimes \mathit{Ins}_q(\vec{w}) \otimes \mathbf{t}^{\mathbf{d}_\beta \mathbf{d}_{\beta,\alpha}} \rightarrow \\ a \otimes g \otimes \mathit{inertial}(g) \end{array}\right\} \in \frac{(\mathcal{E}, \mathbf{P}, \mathcal{S})}{\mathcal{M}}$$

From these facts, and following a similar reasoning as above we can conclude that

$$\mathsf{inertials}(\mathbf{d}_{\beta,\alpha}) = \mathsf{WM}_i \setminus \{p(\vec{c})\}$$

□

G.0.69. Lemma (Entailment Equivalence). *Let* $\mathbf{d}$ *be a state identifier. Suppose that*

(1) $\mathsf{WM}_i \equiv \mathit{inertials}(\mathbf{d})$
(2) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models$ **not** inconsistent

Then

$$\mathit{facts}(\mathsf{WM}_i) \equiv \mathit{facts}(\mathbf{d})$$

Proof. Since

- the DL language used for the ontology $\mathcal{T}$ is Datalog rewritable,
- by hypotheses the fluents in $\mathsf{inertials}(\mathbf{d})$ are consistent with the ontology, and
- satisfaction in $\mathcal{TR}^{PAD}$ coincides with satisfaction in regular LP when considering fluents literals,

it follows that for any fluent h

$$\mathsf{dtg}(\mathcal{T}), \mathsf{inertials}(\mathbf{d}) \models h \text{ if and only if } \mathcal{T}, \mathsf{WM}_i \models h$$

Therefore

$$\mathit{facts}(\mathsf{WM}_i) \equiv \mathsf{facts}(\mathbf{d})$$

□

G.0.70. Lemma (Preservation: Λ^{ELEM}-Λ^{PS}). *Suppose*

- $\Lambda_{\mathsf{PS}} = (\mathcal{E}, \mathbf{P}, \mathcal{S})$ *and*

- $\Lambda_{\mathsf{PS}}^{elem} = (\mathcal{E}, \mathbf{P}', \mathcal{S})$

Then for every database identifier $\mathbf{d}$

$$\{f \mid f \in \mathcal{L}_{PS} \text{ and } \mathcal{E}, \mathbf{P}', \mathcal{S}, \mathbf{d} \models f\} = \{f \mid f \in \mathcal{L}_{PS} \text{ and } \mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d} \models f\}$$

Proof. Observe that the difference between Λ_{PS} and Λ^{elem} are the rules in $\mathbf{P} \backslash \mathbf{P}'$. These rules define production rules (including the rule for random production rules), and auxiliary fluents and actions like $\mathit{fireable}_r$ and loop_r used in the rule and complex action definitions. By inspection of the rules in Λ_{PS}, it easy to see that the literals defined by these rules appear neither in the body nor in the head of any frame axiom, action definition, rule in $\mathsf{dtg}(\mathcal{T})$, or in the definition of *inconsistence*. Thus, the truth value of the literals defined in $\mathbf{P} \setminus \mathbf{P}'$ do not affect the truth value of $\mathcal{L}_{\text{PS}}$ fluents in any 1-path π. The previous fact yields the claim of the lemma. □

In the remainder of the section. we assume that the sets facts and inertials use Λ_{PS} instead of Λ^{elem}.

G.0.71. Lemma (Soundness of Sequences of actions). *Let* WM_i *be a working memory,* $\alpha_1 \dots \alpha_n$ *actions, and* $\mathbf{d}_i, \mathbf{d}_{i+1} \dots \mathbf{d}_n$ *state identifiers such that the following holds:*

(1) $\mathit{facts}(\mathsf{WM}_i) \equiv \mathit{facts}(\mathbf{d}_i)$
(2) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \dots \mathbf{d}_n \models \alpha_1 \otimes \dots \otimes \alpha_n$
(3) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \models \mathbf{not}\ \mathsf{inconsistent}$ *with* $(j = i \dots n)$

Then there is a working memory WM_n *such that*

$$\mathsf{WM}_i \overset{\alpha_1 \dots \alpha_n}{\twoheadrightarrow} \mathsf{WM}_n \text{ and}$$
$$\mathit{facts}(\mathsf{WM}_n) \equiv \mathit{facts}(\mathbf{d}_n)$$

Proof. The claim of the lemma follows straightforwardly from soundness of actions (Lemma G.0.67) and state and entailment equivalence (Lemmas G.0.68 and G.0.69), and preservation (Lemma G.0.71). □

Before continuing with the following lemma, recall that if $\phi = f_1 \wedge \dots \wedge f_n \wedge \mathbf{neg}\, l_1 \dots \mathbf{neg}\, l_m$ is a conjunction of fluent literals, then $\widehat{\phi}$ denotes the *TR*-serial conjunction $\widehat{\psi} = f_1 \wedge \dots \wedge f_m \wedge \sim l_1 \wedge \dots \wedge \sim l_m$, where

$$\sim l_i = \begin{cases} \mathbf{not}\, l_i & \text{if } l_i \text{ is a } \mathcal{P}_{\text{PS}} \text{ atom} \\ \mathbf{neg}\, l_i & \text{if } l_i \text{ is a } \mathcal{P}_{\text{DL}} \text{ atom} \end{cases}$$

G.0.72. LEMMA. *Let ϕ be a ground conjunction of fluent literals in $\mathcal{L}_{PS}$, WM be a working memory, and $\mathbf{d}$ a state identifier such that the following holds:*

$$\mathsf{WM} \equiv \mathit{inertials}(\mathbf{d})$$
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models \mathbf{not}\ \mathsf{inconsistent}$$

then

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models \widehat{\phi} \quad \textit{if anf only if} \quad \mathcal{T}, \mathsf{WM} \models \phi$$

Proof. Since ϕ is a conjunction of fluents literals, it is enough to check that the lemma holds for $\phi = f$ where f is a fluent literal. Observe that from the hypothesis and Lemma G.0.69 we can conclude that

$$\mathsf{facts}(\mathsf{WM}) \equiv \mathsf{facts}(\mathbf{d})$$

If f is a positive literal, or a DL negative literal, the claim follows from the definition of facts. Suppose $f = \mathbf{neg}\, g$ and g is a positive literal whose predicate symbol is in $\mathcal{P}_{\mathrm{PS}}$.

- ($\rightarrow$) Suppose
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models \mathbf{not}\, g$$
Since $\mathsf{facts}(\mathsf{WM}) \equiv \mathsf{facts}(\mathbf{d})$, it is clear that $g \notin \mathsf{WM}$. Therefore, from the definition of satisfaction in PS we can conclude that
$$\mathcal{T}, \mathsf{WM} \models \mathbf{neg}\, g$$

- ($\leftarrow$) Suppose
$$\mathcal{T}, \mathsf{WM} \models \mathbf{neg}\, g$$
Since $\mathsf{facts}(\mathsf{WM}) \equiv \mathsf{facts}(\mathbf{d})$ and $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models \mathbf{not}\, \mathit{inconsistent}$, it follows that
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \not\models g$$
But since $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \not\models \mathbf{not}\, \mathit{inconsistent}$ we know that there are no undefined fluents in $\mathcal{L}_{\mathrm{PS}}$. Otherwise $\mathit{inconsistent}$ would be undefined in $\mathbf{d}$. Then it must be the case that
$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d} \models \mathbf{not}\, g$$
This concludes the proof of the lemma. □

G.0.73. PROPOSITION. *Let WM_i be a working memory, and $\mathbf{d}_i, \mathbf{d}_{i+1} \ldots \mathbf{d}_n$ state identifiers such that the following holds:*

$$(i) \quad \mathsf{WM}_i \equiv \mathit{inertials}(\mathbf{d}_i)$$
$$(ii) \quad \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \ldots \mathbf{d}_n \models r$$
$$(iii) \quad \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models \mathbf{not}\ \mathsf{inconsistent}$$

Then there is a working memory WM_n *such that*

$$\begin{array}{l}\mathsf{WM}_i \overset{r}{\hookrightarrow} \mathsf{WM}_n \text{ and} \\ \mathsf{WM}_n \equiv \mathit{inertials}(\mathbf{d}_n)\end{array}$$

Proof. We explore the cases where r is a `IF-THEN` rule and where r is a `FOR` rule.

- Suppose r is a `IF-THEN` rule. Applying the definition of r it follows that

  $$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \dots \mathbf{d}_n \models \mathit{fireable}_r(\vec{c}) \otimes \widehat{\psi}_r(\vec{c})$$

  for some elements $\vec{c}$ in the Herbrand universe. From the pervious fact we can conclude that

  (1) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models \widehat{\phi}_r(\vec{c})$
  (2) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models \bigvee_{p \in \mathcal{P}} \mathit{inertial}(p(\vec{e})) \wedge (\Diamond \widehat{\psi}_r(\vec{c}) \otimes \mathbf{not}\ \mathit{inconsistent} \wedge \mathbf{not}\ \mathit{inertial}(p(\vec{e})))$
  (3) $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \dots \mathbf{d}_n \models \widehat{\psi}_r(\vec{c})$

  Since $\mathsf{WM}_i \equiv \mathsf{inertials}(\mathbf{d}_i)$, from (1) and Lemmas G.0.72 and G.0.69 it follows that

  $$\mathcal{T}, \mathsf{WM}_i \models \phi_r(\vec{c}) \tag{G.12}$$

  and from (2) it follows that

  $$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \models \mathbf{not}\ \mathit{inconsistent} \tag{G.13}$$

  Moreover, from the previous fact, by inspection of the frame axioms in Λ and the definition of *inconsistent*, it is easy to see that if for any $j = i \dots n$, we have that $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \models \mathit{inconsistent}$ then for every $k = j + 1 \dots n$, it cannot be that $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \models \mathbf{not}\ \mathit{inconsistent}$. Thus, we can conclude that for every $j = i \dots n$

  $$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \models \mathbf{not}\ \mathit{inconsistent} \tag{G.14}$$

  Let WM_n the working memory resulting from the application of ψ to WM. Observe that since $\widehat{\psi}_r(\vec{c})$ is a serial conjunction of *pdas*, we know it is deterministic. Thus, from (3), (G.14), our hypothesis, and Lemma G.0.71 (soundness of actions) it follows that

  $$\mathsf{WM}_n \equiv \mathsf{inertials}(\mathbf{d}_n) \tag{G.15}$$

  and from (2) we know that there is a ground fluent g such that

  $$\begin{array}{l} g \in \mathsf{WM}_i \\ g \notin \mathsf{WM}_n \end{array} \tag{G.16}$$

From (G.14), it follows that

$$\mathsf{WM}_n \cup \mathcal{T} \text{ is consistent} \tag{G.17}$$

The previous facts implies that there is a non-trivial consistent transition

$$\mathsf{WM}_i \overset{\psi(\vec{c})}{\twoheadrightarrow} \mathsf{WM}_n$$

From the definition of fireability of simple rules, together with (G.12), (G.16), (G.15) and (G.17) it follows that:

$$\mathsf{WM}_i \overset{r}{\hookrightarrow} \mathsf{WM}_n$$

- Suppose r is a `FOR` rule. Applying the definition of r it follows that

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i\mathbf{d}_{i+1} \dots \mathbf{d}_j \dots \mathbf{d}_n \models \mathit{fireable}_r(\vec{c_1}) \otimes \mathit{add\_used}(\vec{c_1}) \otimes \widehat{\psi}_r(\vec{c_1}) \otimes \mathit{loop\_r}$$

for some elements $\vec{c_1}$ in the domain. From the pervious fact we can conclude that

$$\begin{array}{ll}
(1) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models \widehat{\phi}_r(\vec{c_1}) \\
(2) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models \mathbf{not}\ \mathit{used}(\vec{c_1}) \\
(3) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i \models (\Diamond\widehat{\psi}_r(\vec{c_1}) \otimes \mathbf{not}\ \mathit{inconsistent}) \\
(4) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_i\mathbf{d}_{i+1} \models \mathit{add_used}(\vec{c_1}) \\
(5) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{i+1} \dots \mathbf{d}_j \models \widehat{\psi}_r(\vec{c_1}) \\
(6) & \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \dots \mathbf{d}_n \models \mathit{loop_r}
\end{array}$$

Since $\mathsf{WM}_i \equiv \mathsf{inertials}(\mathbf{d}_i)$, from (1) and Lemmas G.0.72 and G.0.69 it follows that

$$\mathsf{WM}_i, \mathcal{T} \models \phi_r(\vec{c_1}) \tag{G.18}$$

From (3) and again Lemmas G.0.72 and G.0.69 it follows that

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_j \models \mathbf{not}\, \mathit{inconsistent} \tag{G.19}$$

Moreover, from the previous fact, by inspection of the frame axioms in Λ and the definition of *inconsistent*, we can conclude that for every $k = i \dots j$

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_k \models \mathbf{not}\ \mathit{inconsistent} \tag{G.20}$$

Let WM_j the working memory resulting from the application of ψ to WM. From the previous facts, our hypothesis and Lemma G.0.71 it follows that

$$\mathsf{WM}_j \equiv \mathsf{inertials}(\mathbf{d}_j) \tag{G.21}$$

From (G.20), and (G.21) it follows that

$$\mathsf{WM}_j \cup \mathcal{T} \text{ is consistent} \tag{G.22}$$

The previous facts implies that there is a consistent transition

$$\mathsf{WM}_i \overset{\psi(\vec{c_1})}{\twoheadrightarrow} \mathsf{WM}_j$$

Following the same reasoning with the rest of the state identifiers, and observing that the fluent *used* avoids firing the rule with the same assignment twice, we can show that there are different vectors of elements $\vec{c_2} \ldots \vec{c_m}$ such that there are consistent transitions

$$\begin{array}{l} \mathsf{WM}_i \overset{\psi(\vec{c_1})\ldots\psi(\vec{c_m})}{\twoheadrightarrow} \mathsf{WM}_n \\ \mathsf{WM}_n \equiv \mathsf{inertials}(\mathbf{d}_n) \end{array} \tag{G.23}$$

Since we know that

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{n-1}\mathbf{d}_n \models \mathit{loop\_r}$$

it must be that

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_{n-1}\mathbf{d}_n \models \mathit{clean\_used}$$

since *clean_used* does not modify the truth value of any fluent whose predicate symbol is in $\mathcal{L}_{\mathrm{PS}}$, it follows that

$$\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \models \mathbf{not}\, \exists \vec{X} : \mathit{fireable\_r}(\vec{X})$$

This implies that either $\mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \models \mathbf{not}\, \exists \vec{X} : \widehat{\phi(\vec{X})}$ or for every vector $\vec{c}$

$$\begin{array}{c} \text{if } \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \models \widehat{\phi}(\vec{c}) \text{ then} \\ \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \not\models (\Diamond\widehat{\psi}(\vec{c}) \otimes \mathbf{not}\, \mathit{inconsistent}) \text{ or } \mathcal{E}, \mathcal{S}, \mathbf{P}, \mathbf{d}_n \models \mathit{used}(\vec{c}) \end{array}$$

From the previous facts, and Lemmas G.0.72 and G.0.71 it follows that r cannot be fire anymore in WM_n without producing an inconsistency or re-using an assigment. Thus, the definition of fireability of recursive rules, the previous fact, and (G.22) and (G.23) yields that:

$$\mathsf{WM}_i \overset{r}{\hookrightarrow} \mathsf{WM}_n$$

this concludes the proof of the proposition. □

G.0.74. THEOREM. *Let* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *be the* $\mathcal{TR}^{PAD}$ *embedding of a consistent* PS. *Suppose* $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_1 \models r_1 \otimes \cdots \otimes r_n$. *Then there are working memories* $\mathsf{WM}_1 \ldots \mathsf{WM}_m$, *such that*

$$\begin{array}{c} \mathsf{WM}_0 \overset{r_1}{\hookrightarrow} \mathsf{WM}_1 \\ \vdots \\ \mathsf{WM}_{m-1} \overset{r_n}{\hookrightarrow} \mathsf{WM}_m \end{array}$$

Proof. Follows straightforwardly by induction on the length n of the serial conjunction using Proposition G.0.73. □

G.0.75. COROLLARY. *Let* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *be the* $\mathcal{TR}^{PAD}$ *embedding of a consistent* PS. *Suppose* $\mathcal{E}, \mathbf{P}, \mathcal{S}, \mathbf{d}_0 \ldots \mathbf{d}_1 \models \mathit{act} \otimes \cdots \otimes \mathit{act}$. *Then there are working memories* $\mathsf{WM}_1 \ldots \mathsf{WM}_m$, *and rules* $r_1 \ldots r_n$ *such that*

$$\begin{array}{c} \mathsf{WM}_0 \overset{r_1}{\hookrightarrow} \mathsf{WM}_1 \\ \vdots \\ \mathsf{WM}_{m-1} \overset{r_n}{\hookrightarrow} \mathsf{WM}_m \end{array}$$

Proof. Follows straightforwardly by induction on the length n of the serial conjunction using Theorem G.0.74. □

Bibliography

[1] Bonner A. and Kifer M., *Transaction logic programming*, Int'l Conference on Logic Programming (Budapest, Hungary), MIT Press, June 1993, pp. 257–282.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of databases*, Addison-Wesley, 1995.

[3] Darko Anicic, Paul Fodor, Roland Stühmer, and Nenad Stojanovic, *An approach for data-driven logic-based complex event processing*, The 3rd ACM International Conference on Distributed Event-Based Systems (DEBS), 2009.

[4] K. R. Apt and M. H. van Emden, *Contributions to the theory of logic programming*, J. ACM **29** (1982), 841–862.

[5] F. Baader, S. Brandt, and C. Lutz, *Pushing the $\mathcal{EL}$ envelope*, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05 (Edinburgh, UK), Morgan-Kaufmann Publishers, 2005.

[6] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.), *The description logic handbook*, Cambridge University Press, 2003.

[7] C. Baral and M. Gelfond, *Representing concurrent actions in extended logic programming*, Proceedings of the 13th international joint conference on Artifical intelligence - Volume 2 (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1993, pp. 866–871.

[8] C. Baral, M. Gelfond, and A. Provetti, *Representing actions: Laws, observations and hypotheses*, Journal of Logic Programming (1997).

[9] C. Baral and J. Lobo, *Characterizing production systems using logic programming and situation calculus*, `http://www.public.asu.edu/~cbaral/papers/char-prod-systems.ps`, 1995.

[10] A. Bonner and M. Kifer, *Applications of transaction logic to knowledge representation*, Proceedings of the International Conference on Temporal Logic (Bonn, Germany), Lecture Notes in Artificial Inteligence, no. 827, Springer-Verlag, July 1994, pp. 67–81.

[11] A.J. Bonner and M. Kifer, *Transaction logic programming (or a logic of declarative and procedural knowledge)*, Tech. Report CSRI-323, University of Toronto, November 1995, `http://www.cs.sunysb.edu/~kifer/TechReports/transaction-logic.pdf` .

[12] Anthony Bonner and Michael Kifer, *A logic for programming database transactions*, Logics for Databases and Information Systems (J. Chomicki and G. Saake, eds.), Kluwer Academic Publishers, March 1998, pp. 117–166.

[13] Anthony J. Bonner and Michael Kifer, *Applications of transaction logic to knowledge representation*, ICTL, 1994, pp. 67–81.

[14] Alexander Borgida, John Mylopoulos, and Raymond Reiter, *On the frame problem in procedure specifications*, IEEE Trans. Software Eng. **21** (1995), no. 10, 785–798.

[15] TIBCO BusinessEvents, *http://www.tibco.com/products/business-optimization/complex-event-processing/businessevents/businessevents.jsp*.

[16] Baral C. and Gelfond M., *Reasoning about intended actions*, Proceedings of the 20th national conference on Artificial intelligence - Volume 2, AAAI Press, 2005, pp. 689–694.

[17] Baral C. and Gelfond Michael, *Reasoning agents in dynamic domains*, pp. 257–279, Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[18] Diego Calvanese, Jeremy Carroll, Giuseppe De Giacomo, Jim Hendler, Ivan Herman, Bijan Parsia, Peter F. Patel-Schneider, Alan Ruttenberg, Uli Sattler, and Michael Schneider, *OWL 2 web ontology language profiles*, Recommendation, W3C, 2007, http://www.w3.org/2007/OWL/wiki/Profiles.

[19] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov, *Evolution of dl-lite knowledge bases*, Proceedings of the 9th International Semantic Web Conference (ISWC'10) (Shanghai, China) (Peter F. Patel-Schneider, Yue Pan, Pascal Hitzler, Peter Mika, Lei Zhang

0007, Jeff Z. Pan, Ian Horrocks, and Birte Glimm, eds.), Springer, 2010, pp. 112–128.

[20] ONTORULE: Audi Use case, *http://ontorule-project.eu/deliverable-videos/d35-pr-audi-use-case-ibm-video*.

[21] W. Chen, M. Kifer, and D.S. Warren, *HiLog: A foundation for higher-order logic programming*, Journal of Logic Programming **15** (1993), no. 3, 187–230.

[22] Carlos Viegas Damásio, José Júlio Alferes, and João Leite, *Declarative semantics for the rule interchange format production rule dialect*, Proceedings of the 9th international semantic web conference on The semantic web - Volume Part I (Berlin, Heidelberg), ISWC'10, Springer-Verlag, 2010, pp. 798–813.

[23] Hasan Davulcu, Michael Kifer, C. R. Ramakrishnan, and I. V. Ramakrishnan, *Logic based modeling and analysis of workflows*, PODS, 1998, pp. 25–33.

[24] Hasan Davulcu, Michael Kifer, and I. V. Ramakrishnan, *Ctr-s: a logic for specifying contracts in semantic web services*, WWW, 2004, pp. 144–153.

[25] Jos de Bruijn and Martín Rezk, *A logic based approach to the static analysis of production systems*, RR, 2009, pp. 254–268.

[26] G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati, *On instance-level update and erasure in description logic ontologies*, J. Log. and Comput. **19** (2009), 745–770.

[27] Roman Dumitru and Kifer Michael, *Semantic web service choreography: Contracting and enactment*, International Semantic Web Conference, 2008, pp. 550–566.

[28] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits, *Combining answer set programming with description logics for the semantic web*, Artif. Intell. **172** (2008), 1495–1539.

[29] E. Allen Emerson, *Temporal and modal logic*, HANDBOOK OF THEORETICAL COMPUTER SCIENCE, Elsevier, 1995, pp. 995–1072.

[30] R. E. Fikes and N. J. Nilsson, *Strips: A new approach to the application of theorem proving to problem solving*, Readings in Planning (J. Allen, J. Hendler, and A. Tate, eds.), Kaufmann, San Mateo, CA, 1990, pp. 88–97.

[31] Paul Fodor and Michael Kifer, *Transaction logic with defaults and argumentation theories*, ICLP (Technical Communications) (John P. Gallagher and Michael Gelfond, eds.), LIPIcs, vol. 11, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011, pp. 162–174.

[32] Charles Forgy, *Rete: A fast algorithm for the many patterns/many objects match problem*, Artif. Intell. **19** (1982), no. 1, 17–37.

[33] Amalia F.Sleghel, *An optimizing interpreter for conccurent transaction logic*, Master's thesis, University of Toronto, 2000.

[34] M. Gelfond and V. Lifschitz, *The stable model semantics for logic programming*, Proceeding of the Fifth Logic Programming Symposium, 1988, pp. 1070–1080.

[35] M. Gelfond and V. Lifschitz, *Representing action and change by logic programs*, Journal of Logic Programming **17** (1993), 301–322.

[36] Enrico Giunchiglia and Vladimir Lifschitz, *An action language based on causal explanation: Preliminary report*, In Proc. AAAI-98, AAAI Press, 1998, pp. 623–630.

[37] John Grant and Jack Minker, *The impact of logic programming on databases*, Commun. ACM **35** (1992), 66–81.

[38] Turner H., *Representing actions in default logic: A situation calculus approach*, In Proceedings of the Symposium in honor of Michael Gelfond's 50th birthday (also in Common Sense 96, 1996.

[39] S. Hanks and D. McDermott, *Nonmonotonic logic and temporal projection*, Artif. Intell. **33** (1987), no. 3, 379–412.

[40] Steve Hanks and Drew McDermott, *Nonmonotonic logic and temporal projection*, Artif. Intell. **33** (1987), 379–412.

[41] Stijn Heymans, Thomas Eiter, and Guohui Xiao, *Tractable reasoning with dl-programs over datalog-rewritable description logics*, European Conference on Artificial Intelligence, 2010, pp. 35–40.

[42] Ian Horrocks, *Ontologies and the semantic web*, Commun. ACM **51** (2008), 58–67.

[43] Samuel Hung, *Transaction logic prototype*, 1996.

[44] Samuel Y.K. Hung, *Implementation and performance of transaction logic in prolog*, Master's thesis, University of Toronto, 1996.

[45] Ullrich Hustadt, Boris Motik, and Ulrike Sattler, *Reasoning in description logics by a reduction to disjunctive datalog*, J. Autom. Reason. **39** (2007), 351–384.

[46] Daniela Inclezan, *Modular action language alm*, Proceedings of the 25th International Conference on Logic Programming (Berlin, Heidelberg), ICLP '09, Springer-Verlag, 2009, pp. 542–543.

[47] Johan and de Kleer, *An assumption-based tms*, Artificial Intelligence **28** (1986), no. 2, 127 – 162.

[48] McCarthy John, *Situations, actions, and causal laws*, Tech. Report Memo 2, Stanford Artificial Intelligence Project, Stanford University, 1983.

[49] IBM JRules, *http://www.ibm.com/software/integration/business-rule-management/jrules-family/*.

[50] M. Kifer, *FLORA-2: An object-oriented knowledge base language*, The FLORA-2 Web Site, http://flora.sourceforge.net.

[51] R Kowalski and M Sergot, *A logic-based calculus of events*, New Gen. Comput. **4** (1986), 67–95.

[52] Robert Kowalski and Fariba Sadri, *Integrating logic programming and production systems in abductive logic programming agents*, Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems, RR '09, Springer-Verlag, 2009, pp. 1–23.

[53] Markus Krötzsch, Sebastian Rudolph, and Peter H. Schmitt, *On the semantic relationship between datalog and description logics*, Proceedings of the Fourth international conference on Web reasoning and rule systems (Berlin, Heidelberg), RR'10, Springer-Verlag, 2010, pp. 88–102.

[54] Peifung E. Lam, John C. Mitchell, and Sharada Sundaram, *A formalization of HIPAA for a medical messaging system*, Proceedings of the 6th International Conference on Trust, Privacy and Security in Digital Business (Berlin, Heidelberg), TrustBus '09, Springer-Verlag, 2009, pp. 73–85.

[55] Georg Lausen, Bertram Ludaescher, and Wolfgang May, *On active deductive databases: The statelog approach*, In Transactions and Change in Logic Databases, Springer-Verlag, 1998, pp. 69–106.

[56] Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl, *Golog: A logic programming language for dynamic domains*, J. Log. Program. **31** (1997), no. 1-3, 59–83.

[57] Vladimir Lifschitz and Hudson Turner, *Splitting a logic program*, ICLP, 1994, pp. 23–37.

[58] H. Liu, C. Lutz, M. Milicic, and F. Wolter, *Updating description logic ABoxes*, Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06) (Patrick Doherty, John Mylopoulos, and Christopher Welty, eds.), AAAI Press, 2006, pp. 46–56.

[59] John Wylie Lloyd, *Foundations of logic programming*, 2nd ed., Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1993.

[60] J.W. Lloyd, *Foundations of logic programming (second edition)*, Springer-Verlag, 1987.

[61] John McCarthy and Patrick J. Hayes, *Some philosophical problems from the standpoint of artificial intelligence*, Machine Intelligence 4 (B. Meltzer and D. Michie, eds.), Edinburgh University Press, 1969, reprinted in McC90, pp. 463–502.

[62] Thielscher Michael, *Flux: A logic programming method for reasoning agents*, TPLP **5** (2005), no. 4-5, 533–565.

[63] Boris Motik, *Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance?*, Proc. of the 4th Workshop on Methods for Modalities (M4M-4) (Berlin, Germany) (Holger Schlingloff, ed.), Informatik-Berichte der Humboldt-Universität zu Berlin, vol. 194, December 1–2 2005, pp. 246–265.

[64] Boris Motik and Riccardo Rosati, *Reconciling Description Logics and Rules*, Journal of the ACM **57** (2010), no. 5, 1–62.

[65] Erik T. Mueller, *Commonsense reasoning*, Morgan Kaufmann, 2006.

[66] D. Pearce and G. Wagner, *Logic programming with strong negation*, Proceedings of the international workshop on Extensions of logic programming (New York, NY, USA), Springer-Verlag New York, Inc., 1991, pp. 311–326.

[67] T. C. Przymusinski, *On the declarative semantics of deductive databases and logic programs*, pp. 193–216, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[68] Teodor Przymusinski, *Well-founded and stationary models of logic programs*, Annals of Mathematics and Artificial Intelligence **12** (1994), 141–187.

[69] Teodor C. Przymusinski, *Stable semantics for disjunctive programs*, New Generation Computing **9** (1991), 401–424.

[70] Louiqa Raschid, *A semantics for a class of stratified production system programs*, J. Log. Program. **21** (1994), no. 1, 31–57.

[71] Reiter Raymond, *The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression*, pp. 359–380, Academic Press Professional, Inc., San Diego, CA, USA, 1991.

[72] Martín Rezk and Werner Nutt, *Combining production systems and ontologies*, The Fifth International Conference on Web Reasoning and Rule Systems RR'11, Springer, 2011.

[73] Dumitru Roman and Michael Kifer, *Reasoning about the behavior of semantic web services with concurrent transaction logic*, VLDB, 2007, pp. 627–638.

[74] 3-DNS Production Rules, *http://support.f5.com/kb/en-us/archived_products/3-dns/manuals/product/3dns4_5_10ref/3dns_prodrules.html*.

[75] Amalia F. Sleghel, *Concurrent transaction logic prototype*, 2000.

[76] Michael Thielscher, *Reasoning robots: the art and science of programming robotic agents*, Applied logic series, Springer, 2005.

[77] Jeffrey D. Ullman, *Principles of database and knowledge-base systems, volume ii*, Computer Science Press, 1989.

[78] M.H. van Emden and R.A. Kowalski, *The semantics of predicate logic as a programming language*, Journal of ACM **23** (Oct. 1976), no. 4, 733–742.

[79] A. Van Gelder, K.A. Ross, and J.S. Schlipf, *The well-founded semantics for general logic programs*, Journal of the ACM **38** (1991), no. 3, 620–650.

[80] W3C, 2010, Available from `http://www.w3.org/TR/rif-prd/`.

[81] Marianne Winslett, *Updating logical databases*, Cambridge University Press, New York, NY, USA, 1990.

[82] G. Yang, M. Kifer, and C. Zhao, *FLORA-2: A rule-based knowledge representation and inference infrastructure for the Semantic Web*, International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003), Lecture Notes in Computer Science, vol. 2888, Springer, November 2003, pp. 671–688.

[83] Dmitry S. Yershov and Steven M. LaValle, *Sufficient conditions for the existence of resolution complete planning algorithms*, WAFR, 2010, pp. 303–320.

[84] R. M. Young, *Production systems in cognitive psychology*, International Encyclopedia of the Social and Behavioral Sciences (2001), 12143–12146.

[85] Carlo Zaniolo, *A unified semantics for active and deductive databases*, Workshop on Rules In Database Systems (RIDS-93), Springer Verlag, 1993, pp. 271–287.

Publications

Journal Papers

- Martín Rezk and Michael Kifer, *Transaction Logic with Partially Defined Actions*. Submitted to Journal on Data Semantics, 2011.

Refereed Conference Publications

- Martín Rezk and Michael Kifer, *Formalizing Production Systems with Rule-based Ontologies*. To Appear in the proceedings of the Seventh International Symposium on Foundations of Information and Knowledge Systems, 2012.

- Martín Rezk and Michael Kifer, *Reasoning with Actions in Transaction Logic*. In Proc. of the Fifth International Conference on Web Reasoning and Rule Systems, LNCS, 201-216, Ireland, August 29-30, 2011.

- Martín Rezk and Michael Kifer, *On the Equivalence Between the $\mathcal{L}_1$ Action Language and $\mathcal{TR}^{PAD}$*. In Proc. of the Fifth International Conference on Web Reasoning and Rule Systems, LNCS, 185-200, Ireland, August 29-30, 2011..

- Martín Rezk and Werner Nutt, *Combining Production Systems and Ontologies*. In Proc. of the Fifth International Conference on Web Reasoning and Rule Systems, LNCS, 287-293, Ireland, August 29-30, 2011.

- Jos de Bruijn and Martín Rezk, *A Logic Based Approach to the Static Analysis of Production Systems*. In Proc. of the Third International Conference on Web Reasoning and Rule Systems, LNCS, 254-268, Chantilly, VA, USA, October 25-26, 2009.

Project Deliverables

- D3.3, *Complexity and optimization of combinations of rules and ontologies Authors:* Cristina Feier , Hassan Ait-Kaci, Jurgen Angele , Jos de Bruijn

, Hugues Citeau, Thomas Eiter, Adil El Ghali, Volha Kerhet, Eva Kiss, Roman Korf, Thomas Krekeler, Thomas Krennwallner, Stijn Heymans, Alessandro Mosca, Martín Rezk, Guohui Xiao.
URL: `http://ontorule-project.eu/outcomes?func=finishdown&id=46`

- D3.2, *Initial combinations of rules and ontologies* Stijn Heymans, Jos de Bruijn, Martín Rezk, Hassan Ait-Kaci, Hugues Citeau, Roman Korf, Jorg Puhrer, Cristina Feier, Thomas Eiter
URL: `http://ontorule-project.eu/outcomes?func=fileinfo&id=18`

Index