J Data Semant (2012) 1:99–131
DOI 10.1007/s13740-012-0007-8

ORIGINAL ARTICLE

Transaction Logic with Partially Defined Actions

Martín Rezk · Michael Kifer

Received: 6 October 2011 / Revised: 9 June 2012 / Accepted: 9 June 2012 / Published online: 31 July 2012

Abstract In this paper we develop a novel logic formalism, $\mathcal{TR}^{PAD}$ (*Transaction Logic with Partially Defined Actions*), designed for reasoning about the effects of complex actions. $\mathcal{TR}^{PAD}$ is based on a subset of Transaction Logic, but extends it with a new kind of formulas, called *premise*-formulas, which express information about states and the execution of actions. This makes the formalism more suitable for specifying partial knowledge about actions. We develop a sound and complete proof theory for $\mathcal{TR}^{PAD}$ and illustrate the formalism on a number of instructive examples. In addition, we show that an expressive subset of $\mathcal{TR}^{PAD}$ is reducible to standard logic programming and define a precise sense in which this reduction is sound and complete.

Keywords Transaction Logic · Actions · Knowledge representation · Reasoning

1 Introduction

Designing agents that can reason about actions has been a long-standing goal of *Artificial Intelligence*. Of particular interest are agents whose underlying mechanisms are founded on solid logical foundations. A number of sophisticated logical theories for such agents have been developed over the years, including $\mathcal{A}$ [19], $\mathcal{L}_1$ [6], $\mathcal{C}$ [20], $\mathcal{ALM}$ [22], Situation Calculus [23], and Fluent Calculus [39]. However, existing approaches have limitations such as the inability to define complex actions, post-conditions for actions, and recursive actions.

One earlier approach that overcomes many of these limitations is Transaction Logic ($\mathcal{TR}$) [7,9,10], which was intended as a formalism for declarative specification of complex state-changing transactions in logic programming. It has been successfully applied to planning [9], knowledge representation [11], active databases [9], event processing [1], workflow management and Semantic Web services [15,16,35,36], and as a declarative alternative to non-logical features in Prolog [10]. 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.

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, Lam et al. [26] formalized a fragment of these regulations in Prolog, but could not formalize temporal, state-changing regulations. For instance, the language of [26] is not designed to express statements such as, "to be compliant with the law, a DNA test requires a doctor's prescription *after* obtaining the patient's consent." The sequencing operator of $\mathcal{TR}$ enables these kinds of statements naturally.

Although $\mathcal{TR}$ was created to program state-changing transactions, [8] demonstrated that $\mathcal{TR}$ can also do basic, yet interesting reasoning about actions. However, that work failed 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

M. Rezk (✉)
KRDB Research Center, Free University of Bozen-Bolzano, Bolzano, Italy
e-mail: rezk@inf.unibz.it

M. Kifer
Department of Computer Science, Stony Brook University, Stony Brook, NY 11794-4400, USA
e-mail: kifer@cs.stonybrook.edu

(cf. Example 4). In this paper we continue that line of investigation and develop a full-fledged theory, *Transaction Logic with Partially Defined Actions* ($\mathcal{TR}^{PAD}$), which can be used for reasoning about actions over states *in addition* to programming actions. For instance, we can program an action "*do_dna*" that performs a DNA test if the patient gives an ok. In addition, assuming that the hospital was in compliance, if the test was administered we can infer that the patient must have given her prior consent. To carry out this kind of reasoning, we need to *extend* $\mathcal{TR}$ to express information about states. For example, we need to be able to say that in state $\mathbf{D}_2$ the patient consented to a DNA test and that executing the action *do_dna* in state $\mathbf{D}_1$ leads to state $\mathbf{D}_2$. In addition, we need a sound and complete proof system for this new formalism.

Our main focus in this paper is the development of the formalism itself and illustration of its capabilities. $\mathcal{TR}^{PAD}$ supports a great deal of sophistication in action composition, enabling hypothetical, recursive, and nondeterministic actions. Compared with other actions languages like [4–6,19,21,40], $\mathcal{TR}^{PAD}$ supports more general ways of describing actions and can be more selective in when and whether fluents are subject to the laws of inertia.

$\mathcal{TR}^{PAD}$ has been used in [33] to model a production rules language that includes looping constructs and rule-based ontologies. This approach is significantly more expressive and simpler than the earlier attempts, such as [12,14,34]. This approach can be further extended to reasoning about business rules, business process management (BPM), workflow management, and related areas. For instance, in the BPM setting, the execution traces that represent the actual instantiations of processes can be formalized by means of linearly ordered premise formulas. At the same time, similarly to the production rules language mentioned above, partially defined actions can capture very expressive *event-condition-action* business rules. Reasoning can be used to check if the execution traces of business processes are compliant with the business rules being modeled.

We will discuss specific problems that one can model and reason about in $\mathcal{TR}^{PAD}$, but that cannot be handled by the aforementioned action languages. A more detailed study comparing $\mathcal{TR}^{PAD}$ with other formalisms appeared in [31].

Our contribution in this paper is fourfold: (i) extension of $\mathcal{TR}$ with *premise* formulas, which enables us to express information about states and executions in the logic itself and makes the formalism more suitable for specifying partial knowledge about actions; (ii) 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; (iii) development of a sound and complete proof theory for $\mathcal{TR}^{PAD}$; and (iv) a sound and complete reduction of the deterministic subset of $\mathcal{TR}^{PAD}$ to regular logic programming. This last contribution provides an easy way to implement and experiment with the formalism, although a better implementation would use the proof theory directly, similarly to the implementation of the serial-Horn subset of $\mathcal{TR}$ in FLORA-2 [24,42].

A preliminary report on this work appeared in [32], which, however, only sketched most of the definitions and results. The present paper includes additional discussions and examples. It provides further results regarding the correctness of the frame axioms proposed here, the relationship between Horn $\mathcal{TR}$ and logic programming, and includes expanded formulations of theorems and their complete proofs.

This paper is organized as follows: Section 2 presents the necessary background on Transaction Logic. Section 3 deals with the serial-Horn subset of $\mathcal{TR}$ and defines its reduction to regular Horn logic programs. Since Horn $\mathcal{TR}$ is not sufficiently expressive for describing the behavior of actions. Section 4 introduces $\mathcal{TR}^{PAD}$, an extension of Horn $\mathcal{TR}$ that (in some aspects) goes beyond the capabilities of even the full $\mathcal{TR}$, 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. Section 5 introduces a reduction from $\mathcal{TR}^{PAD}$ to Horn logic programs and presents soundness and completeness results for this reduction. Section 6 compares our formalism with other popular action languages. Section 7 concludes the paper. All proofs are given in the appendices.

2 Background

2.1 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 paper. The differences between $\mathcal{TR}^{-}$ and $\mathcal{TR}$ are explained in Sect. 2.1.5.

2.1.1 Syntax

The alphabet of a language, $\mathcal{L}_{TR}$, 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, $\mathcal{P}_{\text{fluents}}$ and $\mathcal{P}_{\text{actions}}$. For easier identification, actions will be written in italics. The former will be used to represent facts in database states and will be called `fluent`-terms. The latter ill be used to represent transactions that change those states.

$()$	(empty conjunction)
$p(t_1 \ldots t_n)$	(*positive atom* or positive literal) where $p \in \mathcal{P}$ and $t_1, \ldots, t_n$ are terms
$\mathbf{neg}\, p(t_1 \ldots t_n)$	(*negative literal*) where $p \in \mathcal{P}_{fluents}$ and $t_1, \ldots, t_n$ are terms
$\phi_1 \wedge \phi_2, \phi_1 \vee \phi_2$	(*conjunction* and *disjunction*)
$\phi_1 \otimes \phi_2$	(*sequencing*)
$\Diamond\phi_1$	(*hypothetical*)
$\phi_1 \rightarrow \phi_2$	(*implication*)
$\exists X.\phi_1(X)$	(*existential quantification*)
$\forall X.\phi_1(X)$	(*universal quantification*)

Fig. 1 Transaction Logic Formulas

- Logical connectives $\wedge$, $\vee$, the implication connectives $\rightarrow$ and $\leftarrow$, 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. *TR* formulas are built as shown in Fig. 1.

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.,

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

We also postulate the usual De Morgan's laws, such as

$$\mathbf{neg}\,\mathbf{neg}\, f \equiv f \quad \text{and} \quad \mathbf{neg}\,(f \wedge g) \equiv \mathbf{neg}\, f \vee \mathbf{neg}\, g$$

This allows us to apply **neg** to complex formulas and not just the atomic ones. For example,

$$\mathbf{neg}\,(f_1 \wedge f_2) \equiv \mathbf{neg}\, f_1 \vee \mathbf{neg}\, f_2$$
$$\mathbf{neg}\,(f_1 \vee f_2) \equiv \mathbf{neg}\, f_1 \wedge \mathbf{neg}\, f_2$$
$$\mathbf{neg}\,(\mathbf{neg}\, f_1 \vee \mathbf{neg}\, f_2) \equiv f_1 \wedge f_2$$

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. For instance, the first part of the following formula

$$\Diamond(\mathit{insert}(\mathsf{vaccinated}, \mathsf{allergic}) \otimes \mathit{bill\_insurance} \otimes \mathsf{has\_paid}) \otimes \mathit{vaccinate}$$

is a hypothetical test to verify that the patient's insurance company will pay in case of an allergic reaction to a vaccine. The actual vaccination is performed only if the test succeeds.

In this paper we will assume that hypothetical formulas contain only serial conjunctions of literals.Implications of the form

$$\phi \leftarrow \psi \tag{1}$$

that can also be written $\psi \rightarrow \phi$, are treated as statements that ϕ is a call to a complex transaction and ψ is a definition for that transaction (i.e., one of the ways to execute it). In Sect. 4, we will see another use for the implication $\rightarrow$ in partial action definitions. If ψ is a fluent literal in (1), we say that (1) is a *fluent rule*. 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.

The following examples illustrate the aforementioned 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.

Example 1 (Blocks World) Suppose we have 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.

In the rules, given below, *move*, *delete*, and *insert* represent actions and on, clear, light, weight are fluents.

$$\begin{aligned}
\mathit{move}(X, Y) \leftarrow\ & \mathsf{on}(X, Z) \otimes \mathsf{clear}(X) \otimes \mathsf{clear}(Y) \otimes \mathsf{light}(X) \\
& \otimes \mathit{delete}(\mathsf{on}(X, Z)) \otimes \mathit{insert}(\mathsf{on}(X, Y)) \otimes \\
& \mathit{delete}(\mathsf{clear}(Y)) \\
\mathsf{light}(X) \leftarrow\ & \mathsf{weight}(X, W) \otimes \mathsf{limit}(L) \otimes W < L \\
?\text{-}\ & \mathit{move}(blk1, blk5) \otimes \mathit{move}(\mathit{SomeBlk}, blk1)
\end{aligned}$$

The first rule is a definition of a 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* which add and remove atomic facts to/from the database. The second rule defines the fluent *light*, which is used in the definition of *move*. The second 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, the second rule could also be written as

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

The last statement in the example is an update transaction, which moves block $blk1$ from its current position to the top of $blk5$ and then finds some other block and moves it on top of $blk1$. For instance, if the current database state is

$$\mathbf{D}_1 = \{clear(blk1), clear(blk5), clear(blk3), on(blk1, table), on(blk3, table), on(blk5, table)\}$$

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

$$\mathbf{D}_2 = \mathbf{D}_1 \setminus \{clear(blk5), on(blk1, table)\} \cup \{on(blk1, blk5)\}$$

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

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

2.1.2 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.

Definition 1 (*State*) A **state** is a set of **ground** (i.e., variable-free) base 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 $\mathbf{D}_3 = \mathbf{D}_2 \cup \{\mathbf{neg}\, c\} \setminus \{c\}$.

In this paper, 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 $\mathcal{L}_{TR}$. 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 universe and Herbrand base are infinite, fixed, and depend only on the language $\mathcal{L}_{TR}$, not on the transaction base. Since this paper 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.

Definition 2 (*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. For instance, consider the database $\mathbf{D} = \{a\}$ and the transaction base $\mathbf{P} = \{b \leftarrow a\}$. In this scenario we can conclude that

$$\mathbf{D} = \{a\} \subset \{a, b\} = \langle \mathbf{D} \rangle$$

Next we define *Herbrand path structures*. Intuitively, Herbrand path structures in *TR* have the same role as transition functions in temporal logics like LTL or μ-Calculus [17]. That is, they are relations between states and actions. However, a transition function takes a state and an action and returns a set of states, while a Herbrand path structure takes paths of the form $\langle \mathbf{D}_1 \ldots \mathbf{D}_n \rangle$ and returns sets of actions that are executable starting along those paths. Actions in *TR* can be non-deterministic, so executions along other paths are also possible (e.g., two different executions of the same action may start at the same state $\mathbf{D}_0$ and end at different states). The definition itself constrains only elementary actions (which are defined over 2-paths), but it does not impose any restrictions on compound actions (or paths of length longer than two). Restrictions for complex actions are defined by the rules that are part of transaction bases.

Definition 3 (*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 $insert(\mathbf{neg}\ p)$.

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.

Definition 4 (*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 $\nu(p) \in \mathcal{M}(\pi)$. For every database state $\mathbf{D}$, $\mathcal{M}, \mathbf{D} \models_\nu ()$

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 $\mathcal{M}$.

Definition 5 (*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.

Definition 6 (*Consistency and completeness of states*) Let $\mathbf{D}$ be a database state. We say that $\mathbf{D}$ is **complete** if and only if for any ground base `fluent`-term f either $f \in \mathbf{D}$ or $\mathbf{neg}\ f \in \mathbf{D}$.

We say that $\mathbf{D}$ is **consistent** if and only if there is *no* ground base `fluent`-term f such that both $f \in \mathbf{D}$ and $\mathbf{neg}\ f \in \mathbf{D}$.

Models that are *not* consistent (i.e., in which p and $\mathbf{neg}\ p$ could be true) are called **paraconsistent**. Although most of the definitions apply to paraconsistent models, our results regarding Horn-$\mathcal{TR}^-$ apply to consistent models only.

2.1.3 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.

Definition 7 (*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

$$\mathbf{P}, \mathbf{D_0} \ldots \mathbf{D_n} \models \phi \qquad (2)$$

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

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

which is true if and only if there is a database sequence $\mathbf{D_0} \ldots \mathbf{D_n}$ that makes (2) true.

Intuitively, (2) says that a successful execution of transaction ϕ can change the database from state $\mathbf{D}_0$ to $\mathbf{D}_1$ ... to $\mathbf{D}_n$.

2.1.4 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 Sect. 3 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

$$b \leftarrow b_1 \otimes \cdots \otimes b_n$$

where the body of the rule is *a serial-Horn goal*, b 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*** is defined as follows:

Definition 8 (*Serial-Horn Goals*)

- A literal is a serial-Horn goal
- if $b_1 \ldots b_n$ are serial-Horn goals, then so is $b_1 \otimes \cdots \otimes b_n$
- if b is a serial-Horn goal, then so is $\Diamond b$.

Recall that a literal can be either a fluent or an action, and action literals are always positive. A ***serial-Horn transaction base*** is a finite set of serial-Horn rules. Note that Example 1 is serial-Horn.

2.1.5 Differences Between $\mathcal{TR}^-$ and $\mathcal{TR}$

For those familiar with $\mathcal{TR}$, we briefly describe the differences between $\mathcal{TR}^-$ and $\mathcal{TR}$. One restriction in $\mathcal{TR}^-$ are that it has only the *explicit* negation $\mathbf{neg}$ (sometimes also called "strong" negation [29]). This negation is weaker than 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* [9]. The data oracle specifies a set of primitive database *queries*, i.e., the static aspect of states, and the transition oracle specifies a set of primitive database *updates*, i.e., the dynamic aspect of states. The restricted nature of $\mathcal{TR}^-$ will enable us, in Sects. 3 and 5, to reduce various interesting subsets of $\mathcal{TR}^-$ to ordinary logic programming.

It is also worth noting that, $\mathcal{TR}^{PAD}$, the formalism developed in Sect. 4, does not use oracles at all. Instead, it introduces a new kind of statements, called *premises*, which generalize the relational oracle and are part of the language of the logic itself and not "black boxes" for the logic (unlike the oracles in $\mathcal{TR}$ and $\mathcal{TR}^{-}$).

2.2 Logic Programs

In this section we briefly remind the basic notions from standard logic programming [28], which will be needed in this paper.

2.2.1 Syntax

The language $\mathcal{L}$ in traditional logic programming is like that of $\mathcal{TR}^{-}$ except that predicates are not partitioned into fluents and actions. The connectives $\otimes$, $\diamondsuit$, $\rightarrow$, and $\vee$ are also omitted (but $\otimes$ and $\diamondsuit$ are later re-introduced as function symbols).

A ***Horn logic program*** is a collection of statements (called ***rules***) of the form

$$l_0 \leftarrow l_1, \ldots, l_k \tag{3}$$

where each l_i is an atom. The atom l_0 is called the ***head*** of the rule r. The set of atoms $\{l_1, \ldots, l_k\}$ is called the ***body*** of r. By a ***clause*** we mean either a rule or a fact.

Later the set of function symbols will be partitioned into several sorts (in the sense of many-sorted logics [18]). We call these programs *sorted Horn logic programs*.

2.2.2 Semantics

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 $\mathbf{B}_P$, is the set of all instantiations of atoms in $\mathbf{P}$ using the terms from $\mathcal{U}$. A ***Herbrand interpretation*** is a subset of the Herbrand base.Satisfaction of a formula ϕ by $\mathbf{M}$, denoted $\mathbf{M} \models \phi$, is defined as follows:

- $\mathbf{M} \models l$, where l is an atom, iff $l \in \mathbf{M}$.
- $\mathbf{M} \models r$, where r is a ground rule of the form (3), iff $l_0 \in \mathbf{M}$ whenever $l_i \in \mathbf{M}$ for all $i = 1, \ldots, k$.
- $\mathbf{M} \models r$, where r is a possibly non-ground rule, if $\mathbf{M} \models r'$ for *every* ground instantiation r' of r.

Queries are statements of the form $\exists \bar{X} a_1 \wedge \cdots \wedge a_k$, where $a_1, \ldots, a_k$ are atoms and $\bar{X}$ are all the variables mentioned in $a_1, \ldots, a_k$. The existential quantifier is usually omitted and comma is used often in lieu of the conjunction symbol $\wedge$. Satisfaction of a query by a Herbrand interpretation, $\mathbf{M}$, is defined as follows:

- $\mathbf{M} \models a_1, \ldots, a_k$, where $a_1, \ldots, a_k$ are ground atoms, iff $\mathbf{M} \models a_i$ for all $i = 1, \ldots, k$.
- $\mathbf{M} \models q$, where q is a non-ground query, iff $\mathbf{M} \models q'$ for *some* ground instantiation of q.

Given a program $\mathbf{P}$, we write $\mathbf{M} \models \mathbf{P}$ if $\mathbf{M} \models r$ for every rule $r \in \mathbf{P}$. In this case we say that $\mathbf{M}$ is a ***model*** of $\mathbf{P}$. It is known that every Horn program $\mathbf{P}$ has a unique ***least model*** [2]—a model $\mathbf{M}_0$ such that for any other model $\mathbf{N}$ of $\mathbf{P}$, $l \in \mathbf{M}_0$ implies $l \in \mathbf{N}$ for any $l \in \mathbf{B_P}$.

If $\mathbf{P}$ is a program and q is a query, we write $\mathbf{P} \models q$ iff $\mathbf{M} \models q$ for every model of $\mathbf{P}$. For Horn programs, this is equivalent to saying that $\mathbf{M}_0 \models q$, where $\mathbf{M}_0$ is the least model of $\mathbf{P}$.

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

In this section we provide a *new reduction* of the serial-Horn subset of $\mathcal{TR}^{-}$ to sorted Horn logic programming and prove its soundness and completeness. This contribution provides an easy way to implement and experiment with the formalism, and it is also part of the reduction of $\mathcal{TR}^{PAD}$ to LP.

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 TR-fluents, i.e., facts or explicitly negated facts (e.g., like $\mathsf{bird}(Tweety)$ or **neg** $\mathsf{bird}(John)$) and the elementary update operations are $\mathit{insert}(f)$ and $\mathit{delete}(f)$, where f is a fluent.

Given a language $\mathcal{L}_{TR}$ 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` × `state`
- $Inertial$ with the signature `fluent` × `action`
- $Execute$ with the signature `action` × `state` × `state`

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

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

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 $\mathcal{L}_{TR}$ have corresponding function symbols in $\mathcal{L}_{LP}$ as explained next:

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

  `constant` × ⋯ × `constant` → `fluent`

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

  `constant` × ⋯ × `constant` → `action`

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

  `constant` × ⋯ × `constant` → `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*.

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 **P** and the initial database state **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: $Execute(\Diamond A, S, S)0 \leftarrow Execute(A, S, S_1)$.

Effect+: For every ground *base* `fluent`-term f, $Holds(f, Result(insert(f), S))$.

Effect-: For every ground *base* `fluent`-term f, $Holds(\mathbf{neg}\, f, Result(delete(f), S))$.

Query: For every ground *base* `fluent`-term f, $Execute(f, S, S) \leftarrow Holds(f, S)$.

Frame Axiom: $\Gamma(\mathbf{P}, \mathbf{D})$ also 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:

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

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.

Execution: For each *elementary* action α, $\Gamma(\mathbf{P}, \mathbf{D})$ includes the following rule:

$$Execute(\alpha, S, Result(\alpha, S))$$

It is easy to see from the above that, for any serial-Horn transaction base **P**, the reduction $\Gamma(\mathbf{P}, \mathbf{D})$ is a Horn logic program. By the well-known result from [41], it has a unique least Herbrand model, which can be computed via a repeated exhaustive application of the rules in $\Gamma(\mathbf{P}, \mathbf{D})$.

Definition 9 (*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

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

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.

Proposition 1 (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 See Appendix A. □

Definition 10 (*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:

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

Theorem 1 (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 10.*

Proof See Appendix A. □

Theorem 2 (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* $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.

Proof See Appendix A. □

4 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 a new kind of formulas, called *premise* formulas, which supply information about states and about execution of actions. Then we propose a sublanguage of the resulting extended formalism. This new formalism, called $\mathcal{TR}^{PAD}$, is a substantial generalization of the serial-Horn subset of $\mathcal{TR}^-$, which was studied in [7,9,10] and briefly described in Sect. 2.1.4. It has a sound and complete proof theory, is much more expressive, and better lends itself to complex representational and reasoning tasks about actions.

$\mathcal{TR}^{PAD}$ consists of *serial-Horn* rules (including *fluent rules*, c.f. Sect. 2.1.4), *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 $\mathcal{TR}^-$). It is worth noting that although one can express PADs in $\mathcal{TR}^-$, there was previously no proof theory to reason about these constructs—only the Horn subset of $\mathcal{TR}^-$ had a proof theory, but that subset did not allow PADs.

Like $\mathcal{TR}^-$, $\mathcal{TR}^{PAD}$ uses only relational states, i.e., they are simply sets of fluents.

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

$$b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \tag{4}$$

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, (4) 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. Note that neither the pre/postcondition nor the pre/effect is mandatory and can be omitted. For instance, the PAD,

$$\mathsf{alive\_turkey} \otimes \mathit{shoot} \otimes \neg\mathsf{alive\_turkey} \rightarrow \mathsf{loaded} \otimes \mathit{shoot}$$

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_4 are conjunctions of fluents, we can use the serial and the classical conjunctions for them interchangeably, as explained in Sect. 2. Each individual conjunct in b_1 will 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***.

$\mathcal{TR}^{PAD}$ makes no use of the built-in actions $\mathit{insert}(f)$ and $\mathit{delete}(f)$ of Sect. 2, since they can be axiomatized by the following PADs:

$\mathit{insert}(f) \rightarrow \mathit{insert}(f) \otimes f$
$g \otimes \mathit{insert}(f) \rightarrow \mathit{insert}(f) \otimes g$ where $g \neq f$ and $g \neq \mathbf{neg}\ f$
$\mathit{delete}(f) \rightarrow \mathit{delete}(f) \otimes \mathbf{neg}\ f$
$g \otimes \mathit{delete}(f) \rightarrow \mathit{delete}(f) \otimes g$ where $g \neq f$ and $g \neq \mathbf{neg}\ f$

Therefore, 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 *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 *TR*, state identifiers are not part of the language, since *TR* formulas never refer to such constants explicitly.In the following, for the sake of simplicity, we will refer to state identifiers just as states.

Definition 11 (*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** 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.

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.

Definition 12 (*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 $\mathcal{M}(\langle \Delta(\mathbf{d}_1) \ \ldots \ \Delta(\mathbf{d}_k) \rangle)$.

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

Definition 13 (*Models*) Let $\mathcal{M}$ 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 $\mathcal{S}$.

Definition 14 (*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

$$\mathbf{P}, \mathcal{S}, \mathbf{d_1} \ldots \mathbf{d_n} \models \phi \tag{5}$$

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

4.1 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 $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash \phi$, 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 [9].

Definition 15 (*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 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* [9,10]*:*
   (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:*

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

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

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

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.

Theorem 3 (Soundness and completeness) *For any serial goal ϕ and a $\mathcal{TR}^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$, the executional entailment $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi$ holds if and only if there is a deduction in $\mathcal{F}$ of the sequent $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash \phi$*

Proof See Appendix B. □

4.2 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.

Example 2 (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 (aids_t(T) and 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 T test (*do_presc*(D, T)), which in turn must be done *after* receiving the consent of P (*rcv_consent*(P, T)). This is expressed as follows:

(1) need_consent(T) ← aids_t(T)
(2) need_consent(T) ← dna_t(T)
(3) *do_cmplnt_test*(T, P, D) ← *rcv_consent*(P, T) ⊗ consent(P, T) ⊗ *do_presc*(D, T) ⊗ presc(T, P, D) ⊗ *do_t*(T, P, D)[1ex]

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. They are *partially defined actions*, which we will define in the following section. Note that compound actions like *do_cmplnt_test* cannot be expressed in action languages like [6,19,40]. In the simple case when compound actions are non-recursive, they can be expressed via relatively simple extensions. For instance, $\mathcal{ALM}$ [22] allows non-recursive compound actions, which are reducible to $\mathcal{AL}$. However, adding recursive actions requires deep changes in the semantics of that language.

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

?– aids_t(wb) ⊗ *do_cmplnt_test*(wb, m, s) ⊗ 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.

Example 3 (Health Insurance, continued) Consider Example 2, and let us now present the three PADs that were left undefined. We also add the fluents dr, matching, and finished.

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

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 description implicitly talks about four main states: the initial state where we have the intent to do the test, but have no requisite permissions,$\mathbf{d}_1$; the state where we have the consent to do the test, but have no prescription (or the other way around),$\mathbf{d}_2$; the state where we have both the consent and the prescription,$\mathbf{d}_3$; and, finally, the state where the test has already been done,$\mathbf{d}_4$. The set of premises, below, shows one particular way the state transitions might have happened:

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

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

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

Let us now consider a popular example in action languages, the Turkey Hunting Problem [6,19,40].

Example 4 (The Turkey Shoot Problem [21]) 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

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

The PADs for the above problem are as follows:

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

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:

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

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:

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

However, common sense reasoners 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 *inertia* (or *frame*) axioms, which say 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.

Example 5 (The Turkey Shoot Problem #2) Consider Example 4 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
- after performing two actions the night falls

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

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

The PADs for the above problem are as follows:

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

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:

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

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

$\mathbf{P}, \mathbf{d}_1 \models \mathbf{neg}\ \mathsf{daylight}$

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

Example 6 (The Turkey Shoot Problem #3) Consider again the scenario described in Example 4. 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 4, and the premises are

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

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:

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

The following example illustrates the use of complex actions:

Example 7 (The Turkey Shoot Problem #4) Again we take Example 4 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

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

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

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

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*.

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

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

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

Examples 4, 5, 6, and 7 illustrate the need for additional axioms to express the common-sense inertia laws.

It is worth noting that the problem described in Examples 3, 4, 5, etc., cannot be expressed in the action language previously cited. For instance, the action language $\mathcal{A}$ [19], does not allow defined fluents, and neither $\mathcal{A}$ nor $\mathcal{AL}$ nor $\mathcal{AC}$ [6,19,40] 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 4 is

$\mathbf{neg}\ \mathsf{loaded} \otimes \mathit{shoot} \otimes \mathbf{neg}\ \mathsf{alive} \rightarrow \mathbf{neg}\ \mathsf{alive} \otimes \mathit{shoot}$

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.

4.3 Axioms of Inertia and Action Theory

We now return to the problem of inertia discussed in Examples 4, 5, 6 and 7. 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 give only the ground version of the action

theory. Lifting to the non-ground case is done in a standard way (cf. [9]).

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 Sect. 5 do not rely on this assumption. Some other action languages (e.g., the $\mathcal{A}$-language of [19]) impose the same restriction.[2] To capture the inertia laws in $\mathcal{TR}^{PAD}$ without the restriction over interloping PADs, one needs a more elaborate theory, which includes default negation [33]. This will be presented in a followup paper.Two PADs for the *same* action α are said to be ***interloping*** if they share a common primitive effect. That is, there is a fluent f, which is a primitive effect of the same partially defined action α in two different PADs. For instance, the following PADs are interloping, as they share a fluent (loaded):

$$\mathsf{has\_bullets} \otimes \mathit{load} \rightarrow \mathit{load} \otimes \mathsf{loaded}$$
$$\mathsf{has\_ammunition} \otimes \mathit{load} \rightarrow \mathit{load} \otimes (\mathsf{loaded} \wedge \mathsf{ready})$$

In this section, we will assume that $\mathcal{TR}^{PAD}$ transaction bases do *not* contain interloping PADs. For conciseness, we will be combining several formulas into one using the usual De Morgan's laws. Note that the 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.

As explained in Example 4, 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 Axiom. For instance, in Example 5 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.[3] 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 $\mathbf{d} \rhd \mathsf{inertial}(f)$.

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:

Unrelatedness: 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 $Frame(\mathbf{P})$:

$$(inertial(h) \wedge h) \otimes \alpha \rightarrow \alpha \otimes h \tag{6}$$

Here it is worth noting that the number of the axioms for unrelatedness is 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 [13] and thus gain the ability to quantify over propositions. In that case, we could replace the aforementioned 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*. Due to the restriction over 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 $Frame(\mathbf{P})$:

$$\begin{aligned}(inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \\ g \otimes \alpha \rightarrow \alpha \otimes g \\ (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \\ \mathbf{neg}\, g \otimes \alpha \rightarrow \alpha \otimes \mathbf{neg}\, g\end{aligned} \tag{7}$$

The following *backward disablement* axioms are also in $Frame(\mathbf{P})$:

$$\begin{aligned}(inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \\ \alpha \otimes g \rightarrow g \otimes \alpha \\ (inertial(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \\ \alpha \otimes \mathbf{neg}\, g \rightarrow \mathbf{neg}\, g \otimes \alpha\end{aligned} \tag{8}$$

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 $\mathbf{neg}\, g$.

Weak Disablement: For each *pda* α and a base literal f such that f is not a primitive effect of α:

$$inertial(f) \otimes \alpha \otimes f \rightarrow f \otimes \alpha \quad \in Frame(\mathbf{P}) \tag{9}$$

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:

[2] In [19], these are called *similar actions*, p. 13.

[3] 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 $\mathsf{inertial}(F)$.

$$\mathbf{neg}\, b' \otimes \alpha \otimes b' \rightarrow b_1 \otimes \alpha \quad \in Frame(\mathbf{P}) \tag{10}$$

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.

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

$$\left.\begin{array}{r}(\wedge_{i=1, i\neq j}^{k} b_1^i) \otimes \alpha \otimes \mathbf{neg}\, b_4 \rightarrow \\ \mathbf{neg}\, b_1^j \otimes \alpha\end{array}\right\} \in Frame(\mathbf{P}) \tag{11}$$

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.

We now return to our examples and show how the aforementioned action theory supports the kinds of reasoning that we desired in Sect. 4.2.

Example 8 (Turkey Shoot, continued) The issue in Example 4 was the inability to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathsf{loaded}$ (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 $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathsf{loaded}$ using the inference system $\mathcal{F}$. The relevant instance of the causality axioms in $Frame(\mathbf{P})$ is

$$\mathsf{alive} \otimes \mathit{shoot} \otimes \mathbf{neg}\, \mathsf{alive} \rightarrow \mathsf{loaded} \otimes \mathit{shoot} \tag{12}$$

Now:

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathsf{alive}$	by the inference rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathsf{alive}$	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)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathsf{loaded}$	by the inference rule 3 (Forward projection), the instance (12) of the causality axiom and the above three sequents

The desired conclusion now follows from the soundness of $\mathcal{F}$ (Theorem 3).

Example 9 (Turkey Shoot 2, continued) In Example 5 we wanted to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathsf{loaded}$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_2 \models \mathbf{neg}\, \mathsf{bullet}$, 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 $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models \mathbf{neg}\, \mathsf{daylight}$, i.e., that initially it was nighttime.

The proof that $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathsf{loaded}$ is the same as in Example 4. Below we show how to prove $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \models \mathbf{neg}\, \mathsf{bullet}$ using the inference system $\mathcal{F}$. The relevant backward projection axiom of $Frame(\mathbf{P})$ is

$$\mathit{load} \otimes \mathbf{neg}\, \mathsf{loaded} \rightarrow \mathbf{neg}\, \mathsf{bullet} \otimes \mathit{load} \tag{13}$$

Now

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathbf{neg}\, \mathsf{loaded}$	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)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathsf{bullet}$	by the inference rule 3 (Forward projection), the instance (13) of the backward projection axiom, and the two sequents above

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

Example 10 (Turkey Shoot 3, continued) The problem in Example 6 was to be able to prove $\mathbf{P}, \mathbf{d}_1 \models \mathbf{neg}\, \mathsf{alive}$. We show how to prove $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models \mathbf{neg}\, \mathsf{alive}$ using the inference system $\mathcal{F}$. The relevant instance of the axioms in $Frame(\mathbf{P})$ is

$$\mathit{inertial}(\mathsf{alive}) \wedge \mathbf{neg}\, \mathsf{loaded} \otimes \mathit{shoot} \otimes \mathbf{neg}\, \mathsf{alive} \rightarrow \mathbf{neg}\, \mathsf{alive} \otimes \mathit{shoot} \tag{14}$$

Next:

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}\, \mathsf{alive}$	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)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathbf{neg}\, \mathsf{loaded}$	by rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathit{inertial}(\mathsf{alive})$	by the inference rule 2 (Premise rule)
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathbf{neg}\, \mathsf{alive}$	by the inference rule 3, the instance (14) of the Backward disablement axiom, and the above sequents

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

Example 11 (Turkey Shoot 4, continued) The problem in Example 7 was to be able to prove

$$\mathbf{P}, \mathbf{d}_1 - \models \mathit{hunt} \otimes \mathbf{neg}\, \mathsf{alive} \tag{15}$$

We show in Fig. 2 how to prove (15) using the inference system $\mathcal{F}$. The relevant instances of the axioms in $Frame(\mathbf{P})$ are

$$\mathit{inertial}(\mathsf{bird\_in\_range}) \wedge \mathsf{bird\_in\_range} \otimes \mathit{hide} \rightarrow \mathit{hide} \otimes \mathsf{bird\_in\_range} \tag{16}$$

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

Example 12 (Health Insurance, continued #2) The issue in Example 3 was to prove

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

We now show a proof for this statement using the inference system $\mathcal{F}$. We assume that all fluents are inertial in every

(1)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash$ *find_location*	by rule 2 (Premise rule)
(2)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash$ bird_in_range	by the inference rule 3 (Forward projection), and sequent (1)
(3)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2\mathbf{d}_3 \vdash$ *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)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash inertial$(bird_in_range)	by rule 2 (Premise rule)
(6)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash$ bird_in_range	by the inference rule 3 (Forward projection), the instance (16) of the Unrelatedness axiom, and sequents (2) an (4)
(7)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash$ hidden $\otimes$ 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)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash$ 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), (9), and (10)
(13)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2\mathbf{d}_3\mathbf{d}_4 \vdash$ *hunt*	by the 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)

Fig. 2 Derivation of $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 - \vdash$ *hunt* $\otimes \mathbf{neg}$ alive

(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*$(m, pr) \otimes$ 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)	$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathbf{neg}$ finished$(m, pr) \wedge \mathbf{neg}$ 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 rules 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)

Fig. 3 Derivation of $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 - \vdash$ *do_cmplnt_test*$(pr, m, s) \otimes$ negative(pr, m)

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) (Unrelatedness)

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

The derivation is shown in Fig. 3.

The required conclusion now follows from the soundness of $\mathcal{F}$ and the definition of entailment in *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 presented in [31] guarantee that the frame axioms introduced above are correct. In that work, we reduce the action language $\mathcal{L}_1$ to $\mathcal{TR}^{PAD}$ and prove the correctness of that reduction. This implies that our action theory in $\mathcal{TR}^{PAD}$ correctly models the inertia laws of $\mathcal{L}_1$.

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 $(\mathbf{P}, \mathbf{D})$.

Definition 16 (*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

$\mathbf{Q} = \mathbf{P}$
$\quad \cup \{$*insert*$(f) \rightarrow$ *insert*$(f) \otimes f \mid$
$\qquad$ for every ground base `fluent`-literal $f\}$
$\quad \cup \{$*delete*$(f) \rightarrow$ *delete*$(f) \otimes \mathbf{neg}\ f \mid$
$\qquad$ for every ground base `fluent`-literal $f\}$

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.

Definition 17 (*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:

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

Proposition 2 (State consistency and completeness) *Let* $(\mathbf{P}, \mathbf{D})$ *be a Horn-*$\mathcal{TR}^-$ *program and* $(\mathbf{Q}, \mathcal{S})_{\boldsymbol{d}}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$. *Let* α *be an action and* $\boldsymbol{d}_1, \boldsymbol{d}_2$ *be state identifiers such that* $\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \alpha$. *If* $\mathbf{D}(\boldsymbol{d}_1)$ *is consistent then so is* $\mathbf{D}(\boldsymbol{d}_n)$. *If, in addition,* $\mathbf{D}(\boldsymbol{d}_1)$ *is complete, then so is* $\mathbf{D}(\boldsymbol{d}_n)$.

Proof See Appendix C. □

Theorem 4 (Soundness) *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\boldsymbol{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$ *and* h *a serial goal. Suppose that* $\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_0 \ldots \boldsymbol{d}_n \models h$. *Then there are relational database states* $\mathbf{D}_1, \ldots, \mathbf{D}_{n-1}$ *(in* $\mathcal{L}_{\mathcal{TR}}$*) such that*

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

where $\mathbf{D}(\boldsymbol{d}_n)$ *is as in Definition 17.*

Proof See Appendix C. □

Theorem 5 (Completeness) *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\boldsymbol{d}_0}$ *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* $\boldsymbol{d}_1 \ldots \boldsymbol{d}_n$ *such that* $\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_0 \ldots \boldsymbol{d}_n \models h$.

Proof See Appendix C. □

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

A set of run-premises is ***converging*** if it has a pair of run-premises that share 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 partially defined actions 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 partially defined actions 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 Sect. 3, all states in $\mathcal{TR}_D^{PAD}$ are relational, i.e., collections of fluents. Given a language $\mathcal{L}_{TR}$ of $\mathcal{TR}_D^{PAD}$, the target language $\mathcal{L}_{LP}$ for the logic programming reduction of $\mathcal{TR}_D^{PAD}$ is defined as in Sect. 3 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 had a *single* `state`-constant s_0, but now we will have a unique `state`-constant $s_{\mathbf{d}}$ for each database state $\mathbf{d}$.

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 Sect. 3, 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 partially defined action.
- $db2st_{\mathcal{S}}(\mathbf{d}) = Result(\alpha, s)$, if $\mathcal{S}$ has a *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, and $db2st_{\mathcal{S}}(\mathbf{d}_0) = s$.

Note 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 $s = db2st_{\mathcal{S}}(\mathbf{d}_1)$:

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

No-op: For each database $\mathbf{D}$ such that $db2st_{\mathcal{S}}(\mathbf{D})$ is non empty, and for any state $s = db2st_{\mathcal{S}}(\mathbf{D})$[4]

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

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

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

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

Decomposition: For every conjunction of `fluent`-terms and hypotheticals 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 following rules:

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

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 Horn rules.

Observe that $\Gamma(\mathbf{P}, \mathcal{S})$ contains one kind of LP rule for each inference rule/axiom in $\mathcal{F}$[5] plus one extra rule that interprets fluents as trivial actions that do not change states. Also note that derived fluents can appear only in $Execute$ statements, and not inside the $Holds$ facts.

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.

Definition 18 (*Correspondence between 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:

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

The following definition relies on the fact that $\mathcal{S}$ has no non-deterministic *run*-premises in $\mathcal{TR}_D^{PAD}$ and that it is well founded.

[5] 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.

The soundness theorem uses the following partial function from `state`-terms to database states. Let $st2db$ be the partial function defined as follows:

Definition 19 Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a relational $\mathcal{TR}^{PAD}_{D}$ specification $(\mathbf{P}, \mathcal{S})$. We define a partial function $st2db$ from `state`-terms to database state identifiers as follows:

- $st2db(s_{\mathbf{d}}) = \mathbf{d}$, 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 $st2db(s_{\mathbf{d}})$ is undefined. Here $s_{\mathbf{d}}$ is the unique state constant that corresponds to the database state $\mathbf{d}$ and α is a partially defined action. Here $s_{\mathbf{d}}$ is the unique state constant that corresponds to the database state $\mathbf{d}$ and α is a partially defined action.
- $st2db(Result(\alpha, s)) = \mathbf{d}$, if $st2db(s)$ exists and $db2st(s) \overset{\alpha}{\rightsquigarrow} \mathbf{d} \in \mathcal{S}$. Otherwise, $st2db(Result(\alpha, s))$ is undefined.

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

Theorem 6 (Soundness) *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}^{PAD}_{D}$ program $(\mathbf{P}, \mathcal{S})$. Suppose that $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$, where s_1 and s_2 are ground `state`-terms and α an action. Then there are relational database states $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_2$ in $\mathcal{L}_{TR}$ such that the following holds:*

(1) $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_2 \models \alpha$
(2) $\boldsymbol{d}_1 = st2db(s_1)$, $\boldsymbol{d}_2 = st2db(s_2)$
(3) $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models D(s_1)$
(4) $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_2 \models D(s_2)$

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

Proof See Appendix D. □

Theorem 7 (Completeness) *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP reduction of a $\mathcal{TR}^{PAD}_{D}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose that $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi$. Then the following holds: – If $n = 1$, and there is a `state`-term s_1 such that $db2st_{\mathcal{S}}(\boldsymbol{d}_1) = s_1$, 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 $db2st_{\mathcal{S}}(\boldsymbol{d}_1) = s_1$ and $db2st_{\mathcal{S}}(\boldsymbol{d}_n) = s_2$, then

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

Proof See Appendix D. □

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

6 Related Work

In this section briefly compare $\mathcal{TR}^{PAD}$ with several well-known action languages.

The $\mathcal{L}_1$ language [6].The alphabet of $\mathcal{L}_1$ consists of three disjoint nonempty sets of symbols: a set of *fluent names* **F**, a set of *action names* **A**, and a set of *situations* **S**. The language $\mathcal{L}_1$ contains two kinds of propositions: *causal laws* and *facts*. In the following table, f, $f_1 \ldots f_n$ are fluent literals, each s_i is a situation, a is an action, and α stands for a sequence of actions.

Causal laws		
(1)	a **causes** f **if** $f_1 \ldots f_n$	(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 $f_1 \ldots f_n$ 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 $\mathcal{D}$. In order to query and reason about domain descriptions, [6] provides a sound, but *incomplete* translation of a fragment of $\mathcal{L}_1$ into logic programming.

There are several 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), etc. However, the semantics of $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$ are completely different and so are some of the capabilities (compound/recursive 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 programming. Further details about the relation between $\mathcal{L}_1$ and $\mathcal{TR}^{PAD}$ can be found in [31].

The $\mathcal{ALM}$ language [22]. 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 also define fluents based on other fluents, one cannot (i) express the execution of actions like the occurrence facts in $\mathcal{L}_1$ and *run*-premises in $\mathcal{TR}^{PAD}$ do, or (ii) to assert information about states, as do the fluent facts in $\mathcal{L}_1$ and *state*-premises in $\mathcal{TR}^{PAD}$. Recursion is disallowed for actions, but it is allowed for fluents.

$\mathcal{TR}^{PAD}$ can express most of the features of $\mathcal{ALM}$ rather 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 using compound actions. However, $\mathcal{TR}^{PAD}$ does not yet handle parallel actions.

The $\mathcal{C}$ language [20]. 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. For instance, **causes** f **if** $g_1 \vee g_2$ **after** h is equivalent to the set of rules **causes** f **if** g_1 **after** h and **causes** f **if** g_2 **after** h. A similar transformation can eliminate disjunction from F and H. In this way, $\mathcal{TR}^{PAD}$ can model non-concurrent domain descriptions of $\mathcal{C}$. In addition, [20] 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 $\mathcal{TR}^{PAD}$. This type of reasoning is done in $\mathcal{TR}^{PAD}$ by exploiting the pre-effects of PADs, which are not available in $\mathcal{C}$. $\mathcal{C}$ also does not support hypothetical tests and hypothetical actions.

Situation Calculus and Golog [23,27,30].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 [23,30].

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 depends on different preconditions. 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 [38,39]. 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 [25]. 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.

In sum, $\mathcal{TR}^{PAD}$ offers a powerful combination of features for action representation most of which are not present in combination in any other formalism. These include recursion, non-determinism, compound and partially defined actions, hypothetical reasoning, forward and backward reasoning in time, logical model theory, and sound and complete proof theory. Nevertheless, $\mathcal{TR}^{PAD}$ does not completely subsume any of the other formalisms discussed in this chapter, for it does not support concurrency and interloping partial action definitions. Enhancing $\mathcal{TR}^{PAD}$ in that direction, including well-founded negation to lift the restriction over interloping actions, and parallel actions as in Concurrent $\mathcal{TR}$ [37], will be the focus of our future work.

7 Conclusions

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 *TR* in FLORA-2 [24].

This work continues the line of research started in [8], which, however, was targeting a different fragment of 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.

In many respects, $\mathcal{TR}^{PAD}$ supports more general ways of describing actions than other related formalisms [4–6,19,21,40] including non-determinism, recursion, and hypothetical suppositions. Uniquely among these formalisms it supports powerful ways of action composition. Nevertheless, as discussed in Sect. 6, $\mathcal{TR}^{PAD}$ does not subsume other works on the subject, as it cannot perform certain reasoning tasks that are possible with formalisms such as [3,5,40].

Enhancing $\mathcal{TR}^{PAD}$ in that direction, including non- monotonic extensions, will be the focus of our future work.

Acknowledgments Martín Rezk was partially supported by the European Commission under the project OntoRule. Michael Kifer was partially supported by NSF grant 0964196.

A Proofs of the Reduction of Serial Horn-$\mathcal{TR}^{PAD}$ to LP

This appendix contains proofs of soundness and completeness of the reduction of serial Horn-$\mathcal{TR}^{-}$ to LP developed in Sect. 3. 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.

Definition 20 (*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

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

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:

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

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

Proposition 3 (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* $\Gamma(\mathbf{P}, \mathbf{D}) \models \mathit{Execute}(\alpha, \hat{s}, s)$ *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, $\mathbf{M}$, so $\Gamma(\mathbf{P}, \mathbf{D}) \models \mathit{Execute}(\alpha, \hat{s}, s)$ 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 **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 **neg** f,[6] $\mathit{Holds}(h, s) \in \mathbf{M}$ 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 **neg** f, the fluent or its negation holds in s. For the remaining fluents

[6] Recall that, by convention, double-negation cancels out.

f and **neg** f, the rule Effect+ yields $Holds(f, s) \in \mathbf{M}$ while $Holds(\mathbf{neg}\ f, s)$ 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 $Execute(\alpha, \hat{s}, s)$ that were derived via $k - 1$ or fewer derivation steps. $Execute(\alpha, \hat{s}, s)$ 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 $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\alpha, \hat{s}, s) \leftarrow Execute(\beta, \hat{s}, s)$$

of the rule Unfolding. This means that $Execute(\beta, \hat{s}, s) \in \mathbf{M}$, and it was derived before $Execute(\alpha, \hat{s}, s)$, i.e., using $< k$ steps. Hence, s is consistent and complete, by the inductive hypothesis.

Sequencing: Suppose that $\alpha = \beta \otimes \gamma$, and $Execute(\alpha, \hat{s}, s)$ was derived via a ground instance

$$Execute(\beta \otimes \gamma, \hat{s}, s) \leftarrow Execute(\beta, \hat{s}, s'), Execute(\gamma, s', s)$$

of the rule Sequencing. This means that $Execute(\beta, \hat{s}, s')$ and $Execute(\gamma, s', s)$ have already been derived in less than k steps. By the inductive hypothesis, 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 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. □

Definition 21 (*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:

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

Theorem 8 (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 10.

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 1,

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 1, 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 was 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 $\mathbf{D}_1, \ldots, \mathbf{D}_n$, 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 both $Execute(\beta, \hat{s}, s')$ and $Execute(\gamma, s', s)$ were derived in less than k steps. Since $\hat{s}$ is consistent and complete, Proposition 1 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 $\mathbf{D}_1, \ldots, \mathbf{D}_n$, 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 10, $f \in \mathbf{D}(\hat{s})$ and, by the definition of executional entailment for fluents in TR, $\mathbf{P}, \mathbf{D}(\hat{s}) \models f$.

Theorem 9 (Completeness) *Let $\Gamma(\mathbf{P}, \mathbf{D})$ be an LP-reduction of a relational serial-Horn TR program $(\mathbf{P}, \mathbf{D})$. Suppose that $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_n\bar{\mathbf{D}} \models \alpha$, 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 $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\alpha, \hat{s}, \bar{s})$.*[7]

Proof The proof relies on the fact that serial-Horn Transaction Logic has a sound and complete proof theory [9]. 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 $\mathbf{D}_1, \ldots, \mathbf{D}_n$, from $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \beta \otimes \gamma$ derive $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \alpha \otimes \gamma$.

TR_2 (hypothetical): From $\mathbf{P}, \mathbf{D}, \mathbf{D}'_1, \ldots, \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, \ldots, \mathbf{D}_m \vdash \gamma$ derive $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, \ldots, \mathbf{D}_m \vdash \Diamond\beta \otimes \gamma$.

TR_3 (query): Suppose f is a fluent such that $f \in \mathbf{D}_1$. Then from $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash f \otimes \beta$.

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 $\mathbf{P}, \mathbf{D}_2 \ldots \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n \vdash u \otimes \beta$.

[7] $\mathbf{D}(\hat{s})$ and $\mathbf{D}(\bar{s})$ were introduced in Definition 10.

We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \hat{\mathbf{D}}, \mathbf{D}_1, \ldots, \mathbf{D}_n, \bar{\mathbf{D}} \vdash \alpha \tag{17}$$

using the above inference rules and the axiom.

Base case: $N = 1$. In that case, (17) 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 (17) 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 (17) is derived using k steps, note that the last step in the derivation must be an application of one of the rules $TR_1, \ldots, TR_4$. We consider each of these cases in turn.

TR_1: (17) was derived because $\alpha \leftarrow \beta \in \mathbf{P}$ and $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_n\bar{\mathbf{D}} \vdash \beta \otimes \gamma$ was derived previously, in less than k steps. By the inductive assumption, $\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\beta \otimes \gamma, \hat{s}, \bar{s})$ 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: (17) was derived because $\alpha = \Diamond\beta \otimes \gamma$ and both $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}'_1 \ldots \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \hat{\mathbf{D}}\mathbf{D}_1 \ldots \mathbf{D}_m\bar{\mathbf{D}} \vdash \gamma$ 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: (17) 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 10 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: (17) 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

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(u, \hat{s}, Result(u, \hat{s})) \tag{18}$$

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

$$\Gamma(\mathbf{P}, \mathbf{D}) \models Execute(\beta, Result(u, \hat{s}), \bar{s}) \tag{19}$$

The inductive claim now follows from (18), (19), and the rule Sequencing. This concludes the proof. □

B Soundness and Completeness Proofs for $\mathcal{F}$

In this Appendix we prove soundness and completeness of the inference system $\mathcal{F}$ developed in Sect. 4.1. For simplicity we present a ground version of the inference system. Lifting to the non-ground case is done in a standard way (cf. [9]).

B.1 Soundness of $\mathcal{F}$

This appendix contains proofs of soundness of the inference system $\mathcal{F}$ developed in Sect. 4.1. 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 11. For convenient reference we reproduce the axioms and inference rules of system $\mathcal{F}$ below.

Definition 22 (*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* [9,10]*:*

(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:*

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

(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

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

(4) *Sequencing:*

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

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

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

Theorem 10 (Soundness of $\mathcal{F}$) *If* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash \phi$ *then* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi$

To prove Theorem 10, 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 [9] after simple adjustments for the existence of PADs (instead of elementary updates defined by transition oracles) in $\mathbf{P}$. Lemma 1 establishes the soundness of the remaining inference rules.

Lemma 1 (Inference Rules)

(1) *Suppose that the premise* $\boldsymbol{d}_1 \overset{\alpha}{\rightsquigarrow} \boldsymbol{d}_2$ *is in* $\mathcal{S}$*. Then* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1\boldsymbol{d}_2 \models \alpha$.

(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*

$$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models b_1 \qquad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_2 \models b_2 \qquad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_1\boldsymbol{d}_2 \models \alpha$$

then $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models b_3$ *and* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_2 \models b_4$.

(3) *Let* ϕ *and* ψ *be serial conjunctions of atoms. If, for some* $1 \leq i \leq n$,

$$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_i \models \phi \qquad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_i \ldots \boldsymbol{d}_n \models \psi$$

then $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi \otimes \psi$.

(4) *Let* ϕ *and* ψ *be serial conjunctions of fluents.*

$$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models \phi \otimes \psi$$

then $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models \phi$ *and* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models \psi$.

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 13 that $\mathbf{M}, \mathbf{d}_1\mathbf{d}_2 \models \alpha$ 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 $\mathbf{M}, \mathbf{d}_1 \mathbf{d}_2 \models \alpha$, the definition of satisfaction in TR implies

$$\mathbf{M}, \langle \mathbf{d}_1 \rangle \models b_3$$
$$\mathbf{M}, \langle \mathbf{d}_2 \rangle \models b_4$$

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 $\otimes$. □

B.2 Completeness of $\mathcal{F}$

To prove completeness of the inference system $\mathcal{F}$ of Sect. 4.1, 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 11.

Definition 23 (*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 12, below. Recall that in $\mathcal{TR}^{PAD}$ we use PADs instead of elementary updates, so elementary updates and transition oracles of TR play no role in our construction.

Lemma 2 *Let $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ be the canonical model of $\mathbf{P}, \mathcal{S}$. Then,*

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \boldsymbol{d} \rangle) \models^c l \quad \textit{for every literal } l \in \boldsymbol{d}$$

Proof If $l \in \mathbf{d}$ then, by the No-op axiom and the 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:

Lemma 3 *If b is a ground atomic formula, then*

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \rangle \models b \quad \textit{iff} \quad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash b$$

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 23, $b \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle)$ if and only if $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash b$. □

We now generalize the aforementioned result to serial conjunctions.

Theorem 11 *Let ϕ be a ground serial conjunction. Then*

$$\textit{if} \quad \mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \rangle \models \phi \quad \textit{then} \quad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash \phi$$

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 $b_1 \otimes \cdots \otimes b_{m-1}$. We have that

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \phi_m \otimes b_m$$

by Definition 13,for some i,

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_i \rangle \models \phi_m \quad \text{and} \quad \mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_i \ldots \mathbf{d}_n \rangle \models b_m$$

by Inductive Hypothesis

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \models b_m$$

by Lemma 3

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_i \ldots \mathbf{d}_n \vdash b_m$$

by Inference Rule 4

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_i \vdash \phi_m \otimes b_m$$

□

Theorem 12 $\mathbf{M}_{\mathbf{P},\mathbf{S}}$ *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 [9], 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

$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \ldots \mathbf{d}_n \rangle \models \alpha$$
$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_1,$$
$$\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_2$$

Then, by Theorem 11,

$$\begin{array}{l}\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \alpha \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_1 \\ \mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash b_2\end{array} \tag{20}$$

and by inference rule 3 we get

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash b_3 \quad \text{and} \quad \mathbf{P}, \mathcal{S}, \mathbf{d}_n \vdash b_4 \tag{21}$$

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 3 it now follows that

$$\begin{array}{l}\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_1' \\ \mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_2'\end{array}$$

and thus, by the definition of $\wedge$, that

$$\begin{array}{l}\mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_1 \rangle \models b_3 \\ \mathbf{M}_{\mathbf{P},\mathbf{S}}, \langle \mathbf{d}_n \rangle \models b_4\end{array}$$

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 $\alpha \in \mathbf{M}_{\mathbf{P},\mathbf{S}}(\langle \mathbf{d}_1\mathbf{d}_2 \rangle)$. □

Corollary 1 (Completeness of $\mathcal{F}$) *Let ϕ be a ground serial conjunction. Then*

$$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi \quad \textit{implies} \quad \mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash \phi$$

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 12
$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \phi$	by Theorem 11

□

C Proof of the Reduction of Horn-$\mathcal{TR}^-$ to $\mathcal{TR}^{PAD}$

This appendix provides a proof that $\mathcal{TR}^{PAD}$ generalizes Horn-$\mathcal{TR}^-$. As a corollary, this means that the frame axioms in the action theory behaves as expected in the relational case. That is, they can model the inertia laws underlying the relational transition oracles.

Proposition 4 (State consistency and completeness) *Let $(\mathbf{P}, \mathbf{D})$ be a Horn-$\mathcal{TR}^-$ program and $(\mathbf{Q}, \mathcal{S})_{\boldsymbol{d}}$ be a relational specification for $(\mathbf{P}, \mathbf{D})$ (see Definition 16). Let α be an action, and $\boldsymbol{d}_1, \boldsymbol{d}_2$ be state identifiers such that $\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \alpha$. If $\mathbf{D}(\boldsymbol{d}_1)$ is consistent then so is $\mathbf{D}(\boldsymbol{d}_n)$. If, in addition, $\mathbf{D}(\boldsymbol{d}_1)$ is complete then so is $\mathbf{D}(\boldsymbol{d}_n)$.*

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

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \alpha \tag{22}$$

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 Weak Disablement frame axiom.

Base Case: $N = 1$. In that case, (22) can be derived only by the run-premise inference rule. Therefore (22) must have the form

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \alpha \tag{23}$$

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 $\mathbf{d}_2 = \mathbf{d}_{1,\textit{insert}(f)}$, where the subindex 1 is a sequence of elementary actions. From definition of satisfaction in path structures we know that for every model $\mathbf{M}$ of $\mathbf{Q}, \mathcal{S}$:

$$\mathbf{M}, \langle \mathbf{d}_1\mathbf{d}_2 \rangle \vdash \textit{insert}(f)$$

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 $\mathbf{M}(\langle \mathbf{d}_2 \rangle)$ are induced by the PADs that exist in the transaction base. Thus, since $\mathbf{M}(\langle \mathbf{d}_0 \rangle)$ is complete and consistent, from the Unrelatedness frame axioms we can conclude that if $\mathbf{M}, \langle \mathbf{d}_1 \rangle \models g$, then $\mathbf{M}, \langle \mathbf{d}_2 \rangle \models g$ for every base fluent g other than f or **neg** f. And from the definition of *insert*(f), we know that $\mathbf{M}, \langle \mathbf{d}_2 \rangle \models f$. It follows that $\mathbf{D}(\mathbf{d}_2)$ is also consistent and complete.

Induction Step: $N = k$, and assume that whenever (22) 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 (22) 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, (22) 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 (22) was derived via the Applying transaction definition Rule. Then we know that

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \beta \tag{24}$$

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 (22) was derived via the Sequencing Rule. Then we know that

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_k \vdash \beta \quad \text{and} \quad \mathbf{Q}, \mathcal{S}, \mathbf{d}_k \ldots \mathbf{d}_n \vdash \gamma \tag{25}$$

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 $\mathbf{D}(\mathbf{d}_n)$. □

Theorem 13 (Soundness) *Let* $\mathbf{P}$ *be a Horn-*$\mathcal{TR}^-$ *transaction base and* $\mathbf{D}$ *a database state. Let* $(\mathbf{Q}, \mathcal{S})_{\boldsymbol{d}_0}$ *be a relational specification of* $(\mathbf{P}, \mathbf{D})$. *Suppose that* $\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_0 \ldots \boldsymbol{d}_n \models h$. *Then there exist relational database states* $\mathbf{D}_1, \ldots, \mathbf{D}_{n-1}$ *(in* $\mathcal{L}_{\mathcal{TR}}$*) such that*

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

where $\mathbf{D}(\boldsymbol{d}_n)$ *is as in Definition 17.*

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

$$\mathbf{Q}, \mathcal{S}, \mathbf{d} \ldots \mathbf{d}_n \vdash h \tag{26}$$

Base Case: $N = 1$. In that case, (26) 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 (26) was derived by the axiom in $\mathcal{F}$, then it follows that (26) has the form: $\mathbf{P}, \mathbf{d} \vdash ()$. The claim follows by the axioms in the Horn-$\mathcal{TR}^-$ proof system.
– Suppose that and (26) 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 (26) 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 $\alpha = \mathit{insert}(f)$. 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 Unrelatedness frame axioms, it follows that for every model $\mathbf{M}$ of $(\mathbf{Q}, \mathcal{S})_{\mathbf{d}_0}$, it holds that

1. For every base fluent g

$$\{g \mid \mathbf{M}, \langle \mathbf{d}_n \rangle \models g \quad \text{and} \quad g \neq f \text{ and } g \neq \mathbf{neg}\, f\} = \{g \mid \mathbf{M}, \langle \mathbf{d}_{n,\alpha} \rangle \models g \quad \text{and} \quad g \neq f \text{ and } g \neq \mathbf{neg}\, f\}$$

2. From the definition of $\mathit{insert}(f)$, we know that

$$\mathbf{M}, \langle \mathbf{d}_{n,\alpha} \rangle \models f$$

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 (26) 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 (26) was derived via the Forward Projection rule. This means that (26) 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 (26) was derived via the Forward Projection rule and $\mathbf{p}$ is a Unrelatedness *pda*. This implies that

1. $\mathbf{p}$ has the form $\mathit{inertial}(f) \wedge f \otimes \alpha \rightarrow \alpha \otimes f$
2. (26) 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 (26) was derived via the Forward Projection rule and **p** is the definition of *insert* or a *delete*. The cases where **p** defines an *insert* or a *delete* are completely symmetrical. For concreteness assume that **p** defines $insert(f)$. This implies that

1. **p** has the form $\alpha \rightarrow \alpha \otimes f$
2. (26) 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 (26) 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 (26) 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 **P** of the form

$$h \leftarrow \beta$$

and (22) was derived via the Applying transaction definition Rule. Then we know that

$$\mathbf{Q}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \vdash \beta \tag{27}$$

was derived in less than k steps. The claim follows from the Inductive Hypothesis. This concludes the soundness proof. □

Theorem 14 (Completeness) *Let* **P** *be a Horn-*$\mathcal{TR}^-$ *transaction base and* **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 $\boldsymbol{d}_1 \ldots \boldsymbol{d}_n$ *such that*

$$\mathbf{Q}, \mathcal{S}, \boldsymbol{d}_0 \ldots \boldsymbol{d}_n \models h$$

Proof The proof relies on the fact that serial-Horn Transaction Logic has a sound and complete proof theory [9]. 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 **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 $\mathbf{D}_1, \ldots, \mathbf{D}_n$, from $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \beta \otimes \gamma$ derive $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \alpha \otimes \gamma$.

TR_2 (hypothetical): From $\mathbf{P}, \mathbf{D}, \mathbf{D}'_1, \ldots, \mathbf{D}'_n \vdash \beta$ and $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, \ldots, \mathbf{D}_m \vdash \gamma$ derive $\mathbf{P}, \mathbf{D}, \mathbf{D}_1, \ldots, \mathbf{D}_m \vdash \Diamond\beta \otimes \gamma$.

TR_3 (query): Suppose f is a fluent such that $f \in \mathbf{D}_1$. Then from $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash f \otimes \beta$.

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 $\mathbf{P}, \mathbf{D}_2 \ldots \mathbf{D}_n \vdash \beta$ derive $\mathbf{P}, \mathbf{D}_1\mathbf{D}_2 \ldots \mathbf{D}_n \vdash u \otimes \beta$.

Observe that we only need to consider states which are "reachable" from **D**. We say that $\mathbf{D}_1$ is reachable from **D**, if there a serial conjunction of elementary actions ϕ such that

$$\mathbf{P}, \mathbf{D}, \ldots, \mathbf{D}_1 \vdash \phi \tag{28}$$

Let $\mathbf{D}_1$ be reachable from **D**. We will now prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \mathbf{D}_1, \ldots, \mathbf{D}_n \vdash h \tag{29}$$

using the aforementioned inference rules and the axiom.

Base case: $N = 1$. In that case, (29) 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 (29) can be derived by the aforementioned 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 (29) 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 (29) 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:(29) 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: (17) 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. □

D Proofs for the Reduction of $\mathcal{TR}^{PAD}$ to Logic Programming

In this appendix we prove soundness and completeness of the reduction of $\mathcal{TR}^{PAD}_D$ to sorted Horn logic programming developed in Sect. 5.

Lemma 4 *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}^{PAD}_D$ 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 19.

Inductive step: Suppose $db2st_{\mathcal{S}}(\mathbf{d}) = s$, and $s = Result(\alpha, s_0)$ for some partially defined action 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 $\mathbf{d} = st2db(s)$. □

Lemma 5 *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a 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 4, 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 $\mathcal{TR}^{PAD}_D$ to logic programming.

Lemma 6 *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a PAD specification $(\mathbf{P}, \mathcal{S})$. Suppose that, for some ground `state`-terms s_1 and s_2 and a partially defined action α*

- *$\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$*
- *$st2db(s_1)$, $st2db(s_2)$ are defined, and*
- *$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1\boldsymbol{d}_2 \models \alpha$, where $\boldsymbol{d}_1 = st2db(s_1)$ and $\boldsymbol{d}_2 = st2db(s_2)$*

Then

- $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \models \boldsymbol{d}(s_1)$
- $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_2 \models \boldsymbol{d}(s_2)$

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 5.

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 $\Gamma(\mathbf{P}, \mathbf{D})$.

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:

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

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 PDA $b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4$ is satisfied, the definition of satisfaction in TR implies that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models b_3$. Finally, since f is a conjunct in b_3, we conclude that $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \models f$. □

Proposition 5 *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a $\mathcal{TR}_D^{PAD}$ specification $(\mathbf{P}, \mathcal{S})$. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\phi, s_1, s_2)$ where s_1 and s_2 are ground `state`-terms. Then*

- *$st2db(s_1)$ and $st2db(s_2)$ are defined and*
- *$\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_2 \models \phi$, where $\boldsymbol{d}_1 = st2db(s_1)$, $\boldsymbol{d}_2 = st2db(s_2)$*

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 4 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 for which $Execute(\alpha, s_1, s_2)$ is derivable in less than k derivation steps using the axioms in $\Gamma(\mathbf{P}, \mathbf{D})$.

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 $\Gamma(\mathbf{P}, \mathbf{D})$:

$$Execute(\phi, s_1, s_2) \leftarrow Execute(\psi, s_1, s_2)$$

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 was 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 $\Gamma(\mathbf{P}, \mathbf{D})$:

$$Execute(\beta \otimes \gamma, s_1, s_2) \leftarrow Execute(\beta, s_1, s), Execute(\gamma, s, s_2)$$

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 $\Gamma(\mathbf{P}, \mathbf{D})$:

$$Execute(\phi, s_1, s_1) \leftarrow Execute(g, s_1, s_1)$$

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 was 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 $\Gamma(\mathbf{P}, \mathbf{D})$:

$$Execute(\Diamond\psi, s_1, s_1) \leftarrow Execute(\psi, s_1, s)$$

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 5.

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 $PAD\ \alpha$. 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:

$$\begin{aligned} Holds(b_3, s_1) \leftarrow\ & Execute(\alpha, S, Result(\alpha, s_1)), \\ & Execute(b_1, s_1, s_1), \\ & Execute(b_2, Result(\alpha, s_1), \\ & \quad Result(\alpha, s_1)) \end{aligned}$$

Recall from Sect. 5 that the aforementioned 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

$$\begin{aligned} Holds(\phi, s_1) \leftarrow\ & Execute(\alpha, s_1, Result(\alpha, s_1)), \\ & Execute(b_1, s_1, s_1), \\ & Execute(b_2, Result(\alpha, s_1), \\ & \quad Result(\alpha, s_1)) \end{aligned}$$

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 $st2db(Result(\alpha, s_1))$ are defined and
- $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d} \models \alpha$, where $\mathbf{d}_1 = st2db(s_1)$ and $\mathbf{d} = st2db(Result(\alpha, s_1))$

The claim now follows from Lemma 6. □

Theorem 15 (Soundness) *Let* $\Gamma(\mathbf{P}, \mathcal{S})$ *be an LP-reduction of a* 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* $\boldsymbol{d}_1, \ldots, \boldsymbol{d}_2$ *(in* $\mathcal{L}_{TR}$*) such that the following holds:*

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

Proof Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$. Claims 1 and 2 follow directly from Proposition 5. Claim 3 follows from Claim 2 and Lemma 6. □

Theorem 16 (Completeness) *Let* $\Gamma(\mathbf{P}, \mathbf{D})$ *be an LP-reduction of a* $\mathcal{TR}_D^{PAD}$ *specification* $(\mathbf{P}, \mathcal{S})$. *Suppose* $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \models \phi$. *Then the following holds:*

- *If* $n = 1$, *and there is a* `state`*-term* s_1 *such that* $db2st_{\mathcal{S}}(\boldsymbol{d}_1) = s_1$, *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* $db2st_{\mathcal{S}}(\boldsymbol{d}_1) = s_1$ *and* $db2st_{\mathcal{S}}(\boldsymbol{d}_n) = s_2$, *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 Sect. 4.1.

We will prove the theorem by induction on the number N of steps needed to derive

$$\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \vdash \phi \tag{30}$$

using the inference system $\mathcal{F}$ from Sect. 4.1. We will be referring to the inference rules using the same enumeration than the one used in Definition 22.

Base Case: $N = 1$. In that case, (30) 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 (30) was derived using the no-op axiom $\mathbf{P}, \mathbf{d}_1 \vdash ()$. Then the following must be true:
  - the sequent (30) has the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash ()$
  - ϕ is 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 (30) 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 (30) was derived using a *state*-premise $\mathbf{d}_1 \rhd \phi \in \mathcal{S}$. Then the following statements are true:
    - the sequent (30) 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 (30) was derived using a *run*-premise $\mathbf{d}_1 \overset{\phi}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$ and (30) 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 (30) 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 (30) 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 (30) was derived because $\phi = a \otimes \psi$, $a \leftarrow \eta \in \mathbf{P}$, and $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \vdash \eta \otimes \psi$ 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 (30) was derived because $\phi = \Diamond\beta \otimes \gamma$, and both $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}' \vdash \beta$ and $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_2 \vdash \gamma$ 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 (30) 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 (30) 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 (30) 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. □

References

1. Anicic D, Fodor P, Stühmer R, Stojanovic N (2009) An approach for data-driven logic-based complex event processing. In: The 3rd ACM international conference on distributed event-based systems (DEBS)
2. Apt KR, van Emden MH (1982) Contributions to the theory of logic programming. J ACM 29:841–862. doi:10.1145/322326.322339
3. Baral C, Gelfond M (1993) Representing concurrent actions in extended logic programming. In: Proceedings of the 13th international joint conference on artifical intelligence, vol 2. Morgan Kaufmann, San Francisco, pp 866–871. http://portal.acm.org/citation.cfm?id=1624140.1624145
4. Baral C, Gelfond M (2000) Reasoning agents in dynamic domains. Kluwer, Norwell, pp 257–279. http://portal.acm.org/citation.cfm?id=566344.566364
5. Baral C, Gelfond M (2005) Reasoning about intended actions. In: Proceedings of the 20th national conference on artificial intelligence, vol 2. AAAI Press, pp 689–694. http://portal.acm.org/citation.cfm?id=1619410.1619443
6. Baral C, Gelfond M, Provetti A (1997) Representing actions: laws, observations and hypotheses. J Logic Program 31:201–244
7. Bonner A, Kifer M (1993) Transaction logic programming. In: International conference on logic programming. MIT Press, Budapest, pp 257–282
8. Bonner A, Kifer M (1994) Applications of transaction logic to knowledge representation. In: Proceedings of the international conference on temporal logic. Lecture notes in artificial intelligence, vol 827. Springer, Bonn, pp 67–81
9. Bonner A, Kifer M (1995) Transaction logic programming (or a logic of declarative and procedural knowledge). Technical report CSRI-323, University of Toronto. http://www.cs.sunysb.edu/~kifer/TechReports/transaction-logic.pdf
10. Bonner A, Kifer M (1998) A logic for programming database transactions. In: Chomicki J, Saake G (eds) Logics for databases and information systems, chap 5. Kluwer, pp 117–166
11. Bonner AJ, Kifer M (1994) Applications of transaction logic to knowledge representation. In: ICTL, pp 67–81
12. de Bruijn J, Rezk M (2009) A logic based approach to the static analysis of production systems. In: RR, pp 254–268
13. Chen W, Kifer M, Warren D (1993) HiLog: a foundation for higher-order logic programming. J Logic Program 15(3):187–230
14. Damásio CV, Alferes JJ, Leite Ja (2010) Declarative semantics for the rule interchange format production rule dialect. In: Proceedings of the 9th international semantic web conference on the semantic web (ISWC'10) volume, part I. Springer, Berlin, pp 798–813. http://portal.acm.org/citation.cfm?id=1940281.1940332
15. Davulcu H, Kifer M, Ramakrishnan CR, Ramakrishnan IV (1998) Logic based modeling and analysis of workflows. In: PODS, pp 25–33
16. Davulcu H, Kifer M, Ramakrishnan IV (2004) Ctr-s: a logic for specifying contracts in semantic web services. In: WWW, pp 144–153
17. Emerson EA (1995) Temporal and modal logic. In: Handbook of theoretical computer science. Elsevier, pp 995–1072
18. Enderton H (2001) A mathematical introduction to logic. Academic Press
19. Gelfond M, Lifschitz V (1993) Representing action and change by logic programs. J Logic Program 17:301–322
20. Giunchiglia E, Lifschitz V (1998) An action language based on causal explanation: preliminary report. In: Proc AAAI-98. AAAI Press, pp 623–630
21. Hanks S, McDermott D (1987) Nonmonotonic logic and temporal projection. Artif Intell 33(3):379–412. doi:10.1016/0004-3702(87)90043-9
22. Inclezan D (2009) Modular action language ALM. In: Proceedings of the 25th international conference on logic programming, ICLP '09, pp 542–543. doi:10.1007/978-3-642-02846-5_55, URL:http://dx.doi.org/10.1007/978-3-642-02846-5_55
23. John M (1983) Situations, actions, and causal laws. Tech. Rep. Memo 2. Stanford Artificial Intelligence Project, Stanford University
24. Kifer M (2007) FLORA-2: an object-oriented knowledge base language. The FLORA-2 Web Site. http://flora.sourceforge.net
25. Kowalski R, Sergot M (1986) A logic-based calculus of events. New Gen Comput 4:67–95. doi:10.1007/BF03037383. http://dl.acm.org/citation.cfm?id=10030.10034
26. Lam PE, Mitchell JC, Sundaram S (2009) A formalization of HIPAA for a medical messaging system. In: Proceedings of the 6th

international conference on trust, privacy and security in digital business, TrustBus '09, pp 73–85. doi:10.1007/978-3-642-03748-1_8
27. Levesque HJ, Reiter R, Lespérance Y, Lin F, Scherl RB (1997) Golog: a logic programming language for dynamic domains. J Log Program 31(1-3):59–83
28. Lloyd J (1987) Foundations of logic programming, 2nd edn. Springer, Berlin
29. Pearce D, Wagner G (1991) Logic programming with strong negation. In: Proceedings of the international workshop on extensions of logic programming. Springer, New York, pp 311–326. http://portal.acm.org/citation.cfm?id=111360.111371
30. Raymond R (1991) The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression. In: Artificial intelligence and mathematical theory of computation. Academic Press, San Diego, pp 359–380. http://dl.acm.org/citation.cfm?id=132218.132239
31. Rezk M, Kifer M (2011) On the equivalence between the L1 action language and partial actions in transaction logic. In: Rudolph S, Gutierrez C (eds) Web reasoning and rule systems. Lecture notes in computer science, vol 6902. Springer, Berlin, pp 185–200. doi:10.1007/978-3-642-23580-1_14
32. Rezk M, Kifer M (2011b) Reasoning with actions in transaction logic. In: Rudolph S, Gutierrez C (eds) Web reasoning and rule systems. Lecture notes in computer science, vol 6902. Springer, Berlin, pp 201–216
33. Rezk M, Kifer M (2012) Formalizing production systems with rule-based ontologies. In: Seventh international symposium on foundations of information and knowledge systems (FOIKS). Lecture notes in computer science, vol 7153. Springer, Kiel
34. Rezk M, Nutt W (2011) Combining production systems and ontologies. In: The fifth international conference on web reasoning and rule systems RR'11. Springer, Berlin, pp 287–293
35. Roman D, Kifer M (2007) Reasoning about the behavior of semantic web services with concurrent transaction logic. In: VLDB, pp 627–638
36. Roman D, Kifer M (2008) Semantic web service choreography: contracting and enactment. In: International semantic web conference, pp 550–566
37. Roman D, Kifer M, Fensel D (2008) Wsmo choreography: from abstract state machines to concurrent transaction logic. In: ESWC, pp 659–673
38. Thielscher M (2005) Flux: a logic programming method for reasoning agents. TPLP 5(4–5):533–565
39. Thielscher M (2005) Reasoning robots: the art and science of programming robotic agents. Applied logic series. Springer, Berlin
40. Turner H (1996) Representing actions in default logic: a situation calculus approach. In: Proceedings of the symposium in honor of Michael Gelfond's 50th birthday
41. van Emden M, Kowalski R (1976) The semantics of predicate logic as a programming language. J ACM 23(4):733–742
42. Yang G, Kifer M, Zhao C (2003) FLORA-2: a rule-based knowledge representation and inference infrastructure for the Semantic Web. In: International conference on ontologies, databases and applications of semantics (ODBASE-2003). Lecture notes in computer science, vol 2888. Springer, Berlin, pp 671–688