Reasoning with Actions in Transaction Logic

Martín Rezk[1] and Michael Kifer[2]

[1] KRDB Research Center, Free University of Bozen-Bolzano, Bolzano, Italy
rezk@inf.unibz.it
[2] Department of Computer Science, Stony Brook University, NY 11794, U.S.A.
kifer@cs.stonybrook.edu

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

1 Introduction

Transaction Logic (*TR*) [5,7,8] was intended as a formalism for declarative specification of complex state-changing transactions in logic programming; and it has been used for planning, active databases, and as a declarative alternative to non-logical features in Prolog. The idea behind *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, [12] formalized a fragment of these regulations in Prolog, but could not formalize temporal, state-changing regulations. For instance, [12] had statements to express the fact that, to be compliant with the law, a DNA test requires a doctor's prescription and a patient's consent, but it was awkward to declaratively express the order in which these two independent actions are to be performed. The sequencing operator of *TR* enables these kinds of statements naturally.

Although *TR* was created to program state-changing transactions, [6] demonstrated that *TR* can do basic, yet interesting reasoning about actions. However, [6] was unable to develop a complete proof theory, and the fragment of *TR* studied there was not expressive enough for modeling many problems in the context of action languages (cf. Example 3). In this paper we continue that investigation and develop a full-fledged theory, *Transaction Logic with Partially Defined Actions* (TR^{PAD}), for reasoning about actions over states *in addition* to programming the actions. For instance, we can program an action "*do_dna*" that performs a DNA test if the patient gives an ok, but (assuming that the hospital was in compliance) if the test was administered we can also infer that the

patient must have given her prior consent. To carry out this kind of reasoning, we need to *extend TR* to express information about the states. For example, we need to be able to state that in a state $\mathbf{D}_2$ the patient consents to a DNA test or that executing the action *do_dna* in state $\mathbf{D}_1$ leads to state $\mathbf{D}_2$. 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. TR^{PAD} has a great deal of sophistication in action composition, enabling hypothetical, recursive, and non-deterministic actions. In particular, compared with other actions languages like [10,9,4,2,16,3], TR^{PAD} supports more general ways of describing actions and can be more selective in when and whether the fluents are subject to the laws of inertia. We will discuss problems that we can model and reason about, but that cannot be handled by the aforementioned action languages. A detailed study comparing TR^{PAD} with other formalisms for describing actions [14] was submitted to this conference.

Our contribution in this paper is four-fold: *(i)* extension of *TR* with *premise*-formulas, which make *TR* more suitable for specifying partial knowledge about actions; *(ii)* defining a subset of the resulting formalism, called 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 TR^{PAD}; *(iv)* a sound and complete reduction of the definite subset of TR^{PAD} to regular logic programming. This last contribution provides an easy way to implement and experiment with the formalism.

This paper is organized as follows. Section 2 presents the necessary background on Transaction Logic. Section 3 defines TR^{PAD}, and develops a sound and complete proof theory for it. Section 4 shows how to express the axioms of inertia in TR^{PAD} and illustrates the use of TR^{PAD} and its proof theory for complex reasoning tasks about actions. Section 5 introduces a reduction from TR^{PAD} to Horn logic programs and presents soundness and completeness results for this reduction. Section 6 concludes the paper. All proofs are given in the technical report [15].

2 Preliminaries

This section reviews the syntax and model theory of a subset of Transaction Logic, which we call TR^{-}, to the extent that is necessary for understanding the results of this paper. One of the important restrictions in TR^{-} is that it uses only the *explicit* negation $\mathbf{neg}$ (sometimes called "strong" negation [13]). This negation is a weaker form of classical negation, and it applies only to fluents, not actions. Another important restriction is that TR^{-} uses only relational database states—unlike the full *TR*, which allows arbitrary states and state transitions. In Section 5, these restrictions will enable us to reduce various subsets of interest of TR^{-} to ordinary logic programming.

Syntax. The alphabet of the language $\mathcal{L}_{TR}$ of TR^{-} consists of countably infinite sets of variables $\mathcal{V}$, function symbols $\mathcal{F}$, and predicates $\mathcal{P}$. The set of predicates $\mathcal{P}$ is further partitioned into two subsets, $\mathcal{P}_{fluents}$ and $\mathcal{P}_{actions}$. The former will be used to represent facts in database states and the latter transactions that change those states. Terms are defined as usual in first order logic. *TR* formulas are built as shown in Figure 1.

A literal whose predicate symbol is in $\mathcal{P}_{fluents}$ will be referred to as a ***fluent literal***. An atom whose predicate symbol is in $\mathcal{P}_{action}$ will be called a ***transactional literal*** or ***(trans)action atom***. Informally, the serial conjunction $\phi \otimes \psi$ is an action composed of an

$p(t_1 \ldots t_n)$	(*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*)
$\phi_1 \rightarrow \phi_2, \quad \phi_1 \leftarrow \phi_2$	(*implication*)
$\exists X.\phi_1(X), \quad \forall X.\phi_1(X)$	(*existential and universal quantification*)

Fig. 1. Transaction Logic Formulas

execution of ϕ followed by an execution of ψ. The hypothetical formula $\diamond\phi$ represents an action where ϕ is *hypothetically* tested whether it can be executed at the current state. However, no actual changes to the current state take place. Implications of the form $\psi \rightarrow \phi$ (or $\phi \leftarrow \psi$) can be understood as a statement that ϕ is an invocation sequence of a complex transaction and ψ is a definition of the actual course of action to be performed by that transaction. We will say that ϕ is a ***compound action*** if ϕ is an action atom. We will say that ϕ is a ***defined fluent*** if ϕ is a fluent literal.

A ***transaction*** is a statement of the form $?- \exists \bar{X} \phi$, where $\phi = l_1 \otimes \cdots \otimes l_k$ is a serial conjunction of literals (both fluent and action literals) and $\bar{X}$ is a list of all the variables that occur in ϕ. Transactions in TR^- are analogous to (and generalize) the notion of queries in ordinary logic programming. A ***transaction base*** is a set of transaction formulas of the form $head$ or $head \leftarrow body$, where *head* is a literal and *body* a serial conjunction of literals. The following example illustrates the above concepts. We will follow the usual logic programming convention whereby lowercase symbols represent constants, function, and predicate symbols, and the uppercase symbols represent variables that are universally quantified outside of the rules. The universal quantifiers are omitted, as usual.

Example 1 (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)).
In TR^{PAD}, this is expressed as follows:

(1) need_consent$(T) \leftarrow$ aids_t(T) (2) need_consent$(T) \leftarrow$ dna_t(T)

(3) *do_cmplnt_test*$(T, P, D) \leftarrow$ *rcv_consent*(P, T)consent$(P, T) \otimes$ *do_presc*$(D, T) \otimes$ $\otimes$ presc$(T, P, D) \otimes$ *do_t*(T, P, D)

In the rules above, *do_cmplnt_test*, *rcv_consent*, *do_presc* and *do_t* are actions, while need_consent, dna_t, aids_t, consent and presc are fluents. Rules (1) and (2) define the fluent need_consent. They consist exclusively of fluents so they are regular logic programming rules that do not cause state transitions. Moreover, serial conjunction of fluents is equivalent to the use of the classical conjunction, since fluents do not cause state transitions. Rules (1) and (2) formalize regulation (i). Rule (3) defines the *compound* action *do_cmplnt_test* which formalizes regulation (ii). The three actions in Rule (3) will be defined in Example 2. 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 [9,4,16].

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

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

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.

Model Theory. In TR^{-}, truth values of formulas are 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 another state. In doing so, the execution may pass through any number of intermediate states. A ***database state*** (or just a *state,* for short) $\mathbf{D}$ is a set of ***ground*** (i.e., variable-free) fluent literals. 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. We use the Herbrand semantics for 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***, which we denote by $\mathcal{U}$. 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. A central feature in the semantics of TR^{-} is the notion of execution *paths*, since *TR* formulas are evaluated over paths and *not* over states like in temporal logics. 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 $k \geq 1$. It is worth noting that 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.

Definition 1 (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 $\langle \boldsymbol{d}_1 \ \ldots \ \boldsymbol{d}_k \rangle$ is a path abstraction then $\langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $\mathbf{D}_i = \Delta(\boldsymbol{d}_i)$, is an execution path. We will also sometimes write $\mathcal{M}(\langle \boldsymbol{d}_1 \ \ldots \ \boldsymbol{d}_k \rangle)$ meaning $\mathcal{M}(\langle \Delta(\boldsymbol{d}_1) \ \ldots \ \Delta(\boldsymbol{d}_k) \rangle)$* □

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

state $\mathbf{D}_n$ (actions in *TR* can be non-deterministic). The following definition formalizes the idea that truth of TR^- formulas is defined on paths. As in classical logic, to define the truth value of quantified formulas we use the usual notion of variable assignment.

Definition 2 (Satisfaction). *Let $\mathcal{M}$ be a Herbrand path structure, π be a path, and let ν be a variable assignment.*
– **Base case:** *If p is a literal, then $\mathcal{M}, \pi \models_\nu p$ if and only if $\nu(p) \in \mathcal{M}(\pi)$.*
– **"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 $\mathcal{M}, \pi \models_\nu \psi$.*
– **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 $\mathcal{M}, \pi \models_\nu \phi$.*
– **Serial conjunction:** *$\mathcal{M}, \pi \models_\nu \phi \otimes \psi$, where $\pi = \langle \mathbf{D}_1 \ldots \mathbf{D}_k \rangle$, iff there exists a prefix subpath $\pi_1 = \langle \mathbf{D}_1 \ldots \mathbf{D}_i \rangle$ and a suffix subpath $\pi_2 = \langle \mathbf{D}_i \ldots \mathbf{D}_k \rangle$ (with π_2 starting where π_1 ends) such that $\mathcal{M}, \pi_1 \models_\nu \phi$ and $\mathcal{M}, \pi_2 \models_\nu \psi$. Such a pair of subpaths is called a* split *and we will be writing $\pi = \pi_1 \circ \pi_2$ to denote this.*
– **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.*
– **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 $\mathbf{D}$.*

Variable assignments are omitted for *sentences*, *i.e.*, formulas with no free variables, and from now on, we will deal with sentences only, unless explicitly stated otherwise. If $\mathcal{M}, \pi \models \phi$, we say that sentence ϕ is *satisfied* (or is *true*) on path π in structure $\mathcal{M}$.

Definition 3 (Model). *A Herbrand 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 Herbrand path structure is a **model of a set of formulas** if it is a model of every formula in the set.* □

A TR^- program consists of two distinct parts: a transaction base $\mathbf{P}$ and an initial database state $\mathbf{D}$. Recall that the database is a set of fluent literals 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 TR^- to the notion of execution.

Definition 4 (Executional entailment). *Let $\mathbf{P}$ be a transaction base, ϕ a transaction formula, and let $\boldsymbol{d_0} \ldots \boldsymbol{d_n}$ be a path abstraction. Then the following statement*

$$\mathbf{P}, \boldsymbol{d_0} \ldots \boldsymbol{d_n} \models \phi \tag{1}$$

is said to be true if and only if $\mathcal{M}, \langle \mathbf{D_0} \ldots \mathbf{D_n} \rangle \models \phi$, where $\mathbf{D}_i = \Delta_{\mathcal{M}}(\boldsymbol{d}_i)$, for every model $\mathcal{M}$ of $\mathbf{P}$. Related to this is the statement $\mathbf{P}, \boldsymbol{d_0}{---} \models \phi$, which is true iff there is a database sequence $\mathbf{D_0} \ldots \mathbf{D_n}$, where $\mathbf{D}_0 = \Delta_{\mathcal{M}}(\boldsymbol{d}_0)$, that makes (1) true. It says that a successful execution of transaction ϕ can change the database from state $\mathbf{D}_0$ to $\mathbf{D}_1 \ldots$ to $\mathbf{D}_n$. □

3 Partially Defined Actions and Incomplete Information

In this section we enhance TR^- to make it suitable for representing commonsense knowledge about the effects of actions in the presence of incomplete information. Our first step is to introduce *premise*-formulas, which are statements about action executions, that were not in the original Transaction Logic (and thus not in TR^-). Then we

propose a sublanguage of the resulting extended formalism. The new formalism, called TR^{PAD}, has a sound and complete proof theory, is much more expressive than the Horn fragment of *TR* studied in [5,7,8], and better lends itself to complex representational and reasoning tasks about actions.

TR^{PAD} consists of *serial-Horn* rules *partial action definitions* (PADs), and certain statements about action execution, which we call *premises*. Like TR^{-}, TR^{PAD} uses only relational states, i.e., they are simply sets of fluent literals.

A ***serial-Horn rule*** is a statement of the form:

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

where $n \geq 0$, b is a literal, and each b_i is a literal or a hypothetical serial conjunction (i.e., $\Diamond(serial\_conjunction)$). If the rule head is a fluent literal then we require that all the body literals are also fluents. We will refer to these last type of rules as *fluent rules*.

A ***partial action definition*** (a ***PAD*** , for short) is a statement of the form:

$$b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4 \tag{2}$$

where each b_i is a conjunction of fluent literals and α is an action atom (the serial conjunction $\otimes$ binds stronger than the implication). 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, (2) 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. It is worth noticing that neither the pre/postcondition nor the pre/effect are mandatory. Thus, any of them can be omitted. For instance, the PAD, alive_turkey $\otimes$ *shoot* $\otimes$ $\neg$alive_turkey $\rightarrow$ loaded, 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. Recall that since b_1, b_2, b_3, and b_4 are conjunctions of fluents, the serial and the classical conjunctions behave identically for them, since for fluents the semantics of TR^{-} guarantees that $f^1 \wedge ... \wedge f^n \equiv f^1 \otimes ... \otimes f^n$. Each individual conjunct in b_1 and b_4 will also be called a ***primitive precondition*** and ***primitive effect*** respectively. Observe that we distinguish two kinds of actions: ***partially defined actions*** (abbr., *pda*) and ***compound actions***. Partially defined actions can be defined by PADs only. In contrast, compound actions are defined only via serial-Horn rules. Note that *pda*s can appear in the rule bodies that define compound actions and, in this way, TR^{PAD} can be used to create larger action theories out of smaller ones in a modular way. A TR^{PAD} ***transaction base*** is a set of serial-Horn rules and PADs definitions.

One key addition that TR^{PAD} brings to *TR* is the notion of *premises*. In TR, state identifiers are not part of the language, since TR formulas never refer to such constants explicitly. In contrast, TR^{PAD} premises do explicitly use state identifiers, so these constants are part of the TR^{PAD} language.

Definition 5 (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).*[3] *A TR^{PAD} **specification** is a pair $(\mathbf{P}, \mathcal{S})$ where $\mathbf{P}$ is a TR^{PAD} transaction base, and $\mathcal{S}$ is a set of premises.* □

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

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 TR^{PAD} specifications.

Definition 6. (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 $\boldsymbol{d}_1 \overset{\alpha}{\rightsquigarrow} \boldsymbol{d}_2$ and $\mathcal{M}, \langle \Delta(\boldsymbol{d}_1)\Delta(\boldsymbol{d}_2)\rangle \models \alpha$.
– σ is a state-premise $\boldsymbol{d} \rhd f$ and $\mathcal{M}, \langle \Delta(\boldsymbol{d})\rangle \models f$.
*$\mathcal{M}$ is a **model** of a set of premises $\mathcal{S}$ if it satisfies every statement in $\mathcal{S}$.* □

Entailment is defined similarly to *TR*. That is, a specification $(\mathbf{P}, \mathcal{S})$ ***entails*** a formula ϕ on $\mathbf{d}_1 \ldots \mathbf{d}_n$, denoted $\mathbf{P}, \mathcal{S}, \mathbf{d_1} \ldots \mathbf{d_n} \models \phi$, if and only if for every model $\mathcal{M}$ of $\mathbf{P}$ and $\mathcal{S}$, we have $\mathcal{M}, \langle \mathbf{\Delta}(\mathbf{d_1}) \ldots \mathbf{\Delta}(\mathbf{d_n})\rangle \models \ \phi$.

3.1 Motivating Examples

We will now show how 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 to do the actual reasoning will be developed in subsequent sections.

Example 2 (Health Insurance, continued). Consider Example 1, and let us now present the three PADs that were left undefined. We also add the fluents dr, matching, and finished.

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

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 premises for the problem at hand are as follows:

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

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 problem, popular example in action languages, called the Turkey Hunting Problem, which is used in [9,4,16] among others.

Example 3 (The Turkey Shoot Problem [10]). 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, the pilgrim can only hunt during the day and after shooting the night falls.
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, daylight, and alive. The only action is *shoot*. The PADs and the set of premises are as follows:

$$\mathbf{P} = \left\{ \begin{array}{c} (\text{daylight} \wedge \\ \text{loaded}) \otimes \textit{shoot} \rightarrow \textit{shoot} \otimes \mathbf{neg}\ \text{alive} \end{array} \right. \quad \mathcal{S} = \left\{ \begin{array}{l} \mathbf{d}_1 \overset{\textit{shoot}}{\rightsquigarrow} \mathbf{d}_2 \\ \mathbf{d}_1 \rhd \text{alive} \\ \mathbf{d}_1 \rhd \text{daylight} \\ \mathbf{d}_2 \rhd \mathbf{neg}\ \text{alive} \\ \mathbf{d}_2 \rhd \mathbf{neg}\ \text{daylight} \end{array} \right.$$

The above premises state that a shooting occurs at some state $\mathbf{D}_1$ ($= \Delta(\mathbf{d}_1)$) and initially the turkey was alive and there was daylight. Following the shooting, the turkey was not alive and it was dark outside.

The PADs describe the effects of shooting. Our requirement is that the logic must be strong enough to prove that the gun was loaded initially: $\mathbf{P}, \mathbf{d}_1 \models \text{loaded}$.

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:

$\mathbf{D}_1 = \{\mathbf{neg}\ \text{loaded}, \text{alive}, \text{daylight}\}$ $\quad$ $\mathbf{D}_2 = \{\mathbf{neg}\ \text{loaded}, \mathbf{neg}\ \text{alive}, \mathbf{neg}\ \text{daylight}\}$

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 defined in Examples 2 and 3, 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, a subtle point in Example 3 is that daylight is not a direct effect of an action, so a simplistic law of inertia would conclude

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

Clearly, this is not the desired conclusion in this case. Thus, there are situations where assuming that things change only due to a direct effect of an action (and remain the same otherwise) is inappropriate. It is worth noting that the problem described in Examples 2 and 3 cannot be expressed in the action languages previously cited. For instance, the action language $\mathcal{A}$ [9], does not allow defined fluents, and neither $\mathcal{A}$ nor $\mathcal{AL}$ nor $\mathcal{AC}$ [9,4,16] support compound actions.

Note that in all previous examples we were using a restricted type of PADs of the form $b_1 \otimes \alpha \rightarrow \alpha \otimes b_2$. This restricted form is sufficient for most types of action specification, but inertia and related laws require a more general kind. For example, a rule suitable for expressing the inertia needed in Example 3 is

$$\mathbf{neg}\ \text{loaded} \otimes \textit{shoot} \otimes \mathbf{neg}\ \text{alive} \rightarrow \mathbf{neg}\ \text{alive} \otimes \textit{shoot}$$

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

3.2 Proof Theory

This section develops an inference system for proving statements about transaction execution. These statements, called ***sequents***, have the form $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_n \models \phi$, where

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

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

Applying transaction definitions:

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

Performing hypothetical operations:

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

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

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

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

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

Sequencing:

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

Fluent decomposition: Suppose ϕ and ψ are serial conjunctions of literals and hypotheticals. Then

$$\frac{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \phi \otimes \psi}{\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \phi \text{ and } \mathbf{P}, \mathcal{S}, \mathbf{d}_1 \vdash \psi}$$

Fig. 2. Inference System $\mathcal{F}$

$n \geq 1$ and ϕ is a serial-Horn goal and $(\mathbf{P}, \mathcal{S})$ a TR^{PAD} specification. Informally, such a sequent says that transaction ϕ can successfully execute starting at a state corresponding to $\mathbf{d}_1$, go through some intermediate states, and end up in a state denoted by $\mathbf{d}_n$. We refer to the inference system developed here as $\mathcal{F}$. It significantly generalizes the inference system for the serial-Horn fragment of TR^- presented in [7].

Definition 7. *(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 axioms and inference rules in Figure 3.2, where $\boldsymbol{d}$, $\boldsymbol{d}_1$, $\boldsymbol{d}_2 \ldots$ are state identifiers.* □

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

Theorem 1. ***(Soundness and Completeness)*** *For any serial goal ϕ and TR^{PAD} program $(\mathbf{P}, \mathcal{S})$, 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 $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_1 \ldots \boldsymbol{d}_n \vdash \phi$*

4 Axioms of Inertia

We now return to the problem of inertia discussed in Examples 2, and 3. Given a 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 this specification to be well-defined, we impose a restriction over ***interloping*** PADs—defined below. Observe that we do not impose this restriction on TR^{PAD} itself—only on the particular action theory presented in this section. For instance, the inference

system and the reduction to logic programming given in Section 5 do not rely on this assumption. Some other action languages (e.g., the $\mathcal{A}$-language of [9]) impose the same restriction. To capture the inertia laws in TR^{PAD} without the restriction over interloping PADs, one needs a more elaborate theory, to be presented in a follow-up paper. Two PADs are said to be ***interloping*** if they share a common primitive effect. For instance, the following PADs are interloping, as they share a fluent (loaded):

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

In this section, we will assume that 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 $\mathbf{neg}$ 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 3, it is a requirement that the frame axioms must be able to model a variety of different behaviors, depending on the problem at hand. In the following we define a general set of rules, $Frame(\mathbf{P})$, that encodes different aspects of the Frame Axiom. For instance, in Example 3 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.[4] 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 fluent literal h and each partially defined action α such that neither h nor $\mathbf{neg}\, h$ is a primitive effect of α

$$(\mathsf{inertial}(h) \wedge h) \otimes \alpha \rightarrow \alpha \otimes h \quad \in Frame(\mathbf{P})$$

Forward and Backward Disablement. Let g or $\mathbf{neg}\, g$ be literals and α a *pda*. Due to the absence of interloping PADs, 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 $Frame(\mathbf{P})$:

$$(\mathsf{inertial}(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes g \otimes \alpha \quad \rightarrow\ \alpha \otimes g$$
$$(\mathsf{inertial}(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \mathbf{neg}\, g \otimes \alpha \ \rightarrow\ \alpha \otimes \mathbf{neg}\, g$$

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

$$(\mathsf{inertial}(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \alpha \otimes g \quad \rightarrow g \otimes \alpha$$
$$(\mathsf{inertial}(g) \wedge \mathbf{neg}\, f_g \wedge \mathbf{neg}\, f_{\mathbf{neg}\, g}) \otimes \alpha \otimes \mathbf{neg}\, g \rightarrow \mathbf{neg}\, g \otimes \alpha$$

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 such that f is not a primitive effect of α

$$\mathsf{inertial}(f) \otimes \alpha \otimes \mathbf{neg}\, f \rightarrow \mathbf{neg}\, f \otimes \alpha \quad \in Frame(\mathbf{P})$$

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

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

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

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. Note that this axiom applies to both inertial and non-inertial fluents.

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 primitive precondition b_1^j

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

That is, if all but one primitive preconditions hold, but the effect of the action is false in the next state, we must conclude that the remaining precondition was false prior to the execution. Again, this axiom applies to both inertial and non-inertial fluents.

We now return to our examples and show how the above action theory handles the problems highlighted in Section 3.1. To preserve continuity, we will first solve the problem described in Example 3 and then we continue with Example 2.

Example 4 (Turkey Shoot, continued). The issue in Example 3 was the inability to prove that the gun was loaded initially: $\mathbf{P}, \mathbf{d}_1 \models$ loaded. This problem arose because TR^{PAD} was not sufficiently expressive to let us specify the rules of inertia. Fortunately, the axioms $Frame(\mathbf{P})$ do the trick. Figure 4 shows a proof for $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models$ loaded using the inference system $\mathcal{F}$. The relevant instance of the axioms in $Frame(\mathbf{P})$ is:

$$(a) \quad \text{alive} \otimes \textit{shoot} \otimes \mathbf{neg}\, \text{alive} \rightarrow \text{loaded} \otimes \textit{shoot} \quad (Causality)$$

We assume that all fluents are inertial in every state, except for daylight, which is not inertial in any state. Since daylight is not subject to the frame axioms, $\mathcal{S}$ does not contain $\mathbf{d}_2 \rhd \text{inertial}(\text{daylight})$ and thus we cannot infer $\mathbf{d}_1 \rhd$ daylight using the inertia axioms in $\mathcal{A}(\mathbf{P})$. The desired conclusion now follows from the soundness of $\mathcal{F}$ (Theorem 1). □

$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash$ alive	by the inference rule Premise
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathbf{neg}$ alive	again by Premise
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash$ *shoot*	by Premise
$\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash$ loaded	by the inference rule Forward Projection, the above instance (a) of the Causality axiom, and the previous three sequents

Fig. 3. Derivation of $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \models$ loaded.

Example 5 (Health Insurance, continued #2). The issue in Example 2 was to prove $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 --- \models$ *do_cmplnt_test*$(pr, m, s) \otimes$ negative(pr, m). We now show a proof for this statement using the inference system $\mathcal{F}$. We assume that all fluents are inertial in every state. The derivation is shown in Figure 4. 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*$(m, pr) \rightarrow$

$$\textit{rcv\_consent}(m, pr) \otimes \mathbf{neg}\ \mathsf{finished}(m, pr) \quad (Unrelatedness)$$

(b) $\mathsf{inertial}(\mathsf{dr(s)}) \otimes \mathsf{dr(s)} \otimes \textit{rcv\_consent}(m, pr) \rightarrow \textit{rcv\_consent}(m, pr) \otimes \mathsf{dr(s)} \quad (Unrelatedness)$

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

(1) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1 \vdash \mathsf{patient}(m) \otimes$ $\mathsf{need\_consent}(pr)$	by the inference rule Premise, App. tran. def. , and Sequencing
(2) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1\mathbf{d}_2 \vdash \textit{rcv\_consent}(m, pr) \otimes$ $\otimes \mathsf{consent}(m, pr)$	by the rule Premise, the previous sequent (1), Forward Projection, and Sequencing
(3) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2 \vdash \mathsf{dr}(s)$	by rule Premise and Forward projection using instance (b) of the Unrelateness axiom
(4) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_2\mathbf{d}_3 \vdash \textit{do\_presc}(pr, m, s) \otimes$ $\mathsf{presc}(pr, m, s)$	by sequent (3), rules Forward Projection and Sequencing
(5) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3 \vdash \mathbf{neg}\ \mathsf{finished}(m, pr)$ $\mathbf{neg}\ \mathsf{matching}(m, pr)$	by rule Premise, Forward Projection, and instance (a) of Unrelateness axiom
(6) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_3, \mathbf{d}_4 \vdash \otimes \textit{do\_t}(pr, m, s)$ $\otimes \mathsf{negative}(pr, m)$	by the above sequent (5) and rules Premise, Forward Projection, and Sequencing
(7) $\mathcal{A}(\mathbf{P}), \mathcal{S}, \mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \vdash$ $\textit{do\_cmplnt\_test}(pr, m, s) \otimes \mathsf{negative}(pr, m)$	by the rule Applying Transaction Definitions and the sequents (2),(4),(6)

Fig. 4. Derivation of $\mathbf{P}, \mathcal{S}, \mathbf{d}_1 \ldots \mathbf{d}_4 \models \textit{do\_cmplnt\_test} \otimes \mathsf{negative}(pr, m)$.

5 Reducing Relational TR_{D}^{PAD} to Logic Programming

In this section we present a reduction for a large fragment of TR^{PAD} to sorted Horn logic programming and state the theorems of soundness and completeness.

The subset in question, which we call *definite* TR^{PAD} , TR_{D}^{PAD}, has only three restrictions: it 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, $\mathbf{d}_1 \overset{shoot}{\rightsquigarrow} \mathbf{d}_2$ and $\mathbf{d}_3 \overset{load}{\rightsquigarrow} \mathbf{d}_2$. A set of run-premises is ***non-deterministic*** if it has a pair of run-premises for the same *pda*s and the same initial state but different final states. Note that this restriction concerns *pda*s only: *compound* actions defined via serial-Horn rules *can* be non-deterministic and TR_{D}^{PAD} has no problem dealing 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 $\mathbf{d}_0, \mathbf{d}_1, \mathbf{d}_2, \ldots$ and *pda*s $\alpha_0, \alpha_1, \alpha_2, \ldots$. 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: $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \quad \mathbf{d}_2 \overset{\beta}{\rightsquigarrow} \mathbf{d}_1$. Given a language $\mathcal{L}_{TR}$ of Transaction Logic, the corresponding language $\mathcal{L}_{LP}$ of the target logic programing reduction of TR_{D}^{PAD} 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. Recall that in Transaction Logic fluents can be viewed as "trivial" actions that do not change the current state. We will see that the same holds for the LP reduction.

$\mathcal{L}_{LP}$ has two distinguished predicates, $Holds$ and $Execute$; and four distinguished function symbols, $Result$, $\mathbf{neg}$, $\diamond$, $\otimes$. 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 $\mathbf{neg}$, $\diamond$, $\otimes$ 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. In addition, $\mathcal{L}_{LP}$ has a unique `state`-constant $s_{\mathbf{d}}$ for each state identifier $\mathbf{d}$ in TR_D^{PAD}.

For each n-ary predicate symbol $p \in \mathcal{P}_{fluent}$ (or $\mathcal{P}_{action}$) and each n-ary function symbol $f \in \mathcal{F}$ in $\mathcal{L}_{TR}$, $\mathcal{L}_{LP}$ has an n-ary function symbol p and f (with the same names as in $\mathcal{L}_{TR}$). To simplify the language, we will use the following conventions about variables: S, S_1, S_2, ... denote `state`-variables, while A, A_1, A_2, etc., will denote `action`-variables. We will also rely on 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, $Holds(f_1 \wedge f_2, s) \equiv Holds(f_1, s) \wedge Holds(f_n, s)$. The reduction $\Gamma(\mathbf{P}, \mathcal{S})$ of a TR_D^{PAD} specification $(\mathbf{P}, \mathcal{S})$ is defined by a set of rules and facts that resemble the inference rules and axioms of Section 3.2. Due to space limitation, we provide only the intuition of the reduction. Full details can be found in [15].

First we define a mapping $\mathbf{db2st}_{\mathcal{S}}$ from the set of database states to the set of `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 TR_D^{PAD} state identifier $\mathbf{d}$ and α is a pda.
- $db2st_{\mathcal{S}}(\mathbf{d}) = Result(\alpha, s)$, if $\mathcal{S}$ has a *run*-premise of the form $\mathbf{d}_0 \overset{\alpha}{\rightsquigarrow} \mathbf{d}$, and $db2st_{\mathcal{S}}(\mathbf{d}_0) = s$.

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

$\Gamma(\mathbf{P}, \mathcal{S})$ 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. For instance, for each *state*-premise $\mathbf{d} \rhd f \in \mathcal{S}$ and every state $s \in db2st_{\mathcal{S}}(\mathbf{d})$, $\Gamma(\mathbf{P}, \mathcal{S})$ contains the fact $Holds(f, s)$. Observe that whenever we can derive $\mathbf{P}, \mathcal{S}, \mathbf{d} \vdash f$ using the Premise inference rule in $\mathcal{F}$, there is a state $s = db2st_{\mathcal{S}}(\mathbf{d})$ such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, s)$. Analogously, for each *run*-premise $\mathbf{d}_1 \overset{\alpha}{\rightsquigarrow} \mathbf{d}_2 \in \mathcal{S}$ there is a state $s = db2st_{\mathcal{S}}(\mathbf{d}_1)$ such that $\Gamma(\mathbf{P}, \mathcal{S})$ contains the fact $Execute(\alpha, s, Result(\alpha, s))$. This similarity is reflected in other rules. For instance, for the *Hypothetical* and *Sequencing* inference rules, the corresponding LP rules in $\Gamma(\mathbf{P}, \mathcal{S})$ are:

$$Execute(\diamond A, S, S) \leftarrow Execute(A, S, S_1) \quad \text{(Hypothetical)}$$
$$Execute(A_1 \otimes A_2, S_1, S_2) \leftarrow Execute(A_1, S_1, S), Execute(A_2, S, S_2) \quad \text{(Sequencing)}$$

The soundness theorem uses the following partial function from `state`-terms to database states. Let $st2db$ be the partial function defined 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 $\mathbf{d}_0$. Otherwise, $st2db(s_{\mathbf{d}})$ is undefined.

[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.

– $st2db(Result(\alpha, s)) = \mathbf{d}$, 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 2 (Soundness). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a TR_D^{PAD} program $(\mathbf{P}, \mathcal{S})$. Suppose $\Gamma(\mathbf{P}, \mathcal{S}) \models Execute(\alpha, s_1, s_2)$, where s_1 and s_2 are ground* `state`*-terms 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 $\mathcal{L}_{TR^{PAD}}$, such that $\Gamma(\mathbf{P}, \mathcal{S}) \models Holds(f, s)$.

Theorem 3 (Completeness). *Let $\Gamma(\mathbf{P}, \mathcal{S})$ be an LP-reduction of a 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:*

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

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

6 Conclusions and Future Work

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 [11].

This work continues the line of research started in [6], 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, TR^{PAD} supports more general ways of describing actions than other related formalisms [10,9,4,2,16,1,3], including non-determinism, recursion, and hypothetical suppositions. Uniquely among these formalisms it supports powerful ways of action composition. Nevertheless, TR^{PAD} does not subsume other works on the subject, as it cannot perform certain reasoning tasks that are possible with formalisms such as [3,1,16]. A detailed study of the relationship of our approach to other languages for actions was submitted to this conference [14].

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

Acknowledgments. We thank Mariano Rodriguez-Muro, Tamara Rezk, and the anonymous reviewers for useful comments and feedback. M. Rezk was partially supported by the European Commission under the project OntoRule. M. Kifer was partially supported by the NSF grant 0964196.

References

1. C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proceedings of the 13th international joint conference on Artifical intelligence - Volume 2*, pages 866–871, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.
2. C. Baral and M. Gelfond. *Reasoning agents in dynamic domains*, pages 257–279. Kluwer Academic Publishers, Norwell, MA, USA, 2000.
3. C. Baral and M. Gelfond. Reasoning about intended actions. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 2*, pages 689–694. AAAI Press, 2005.
4. C. Baral, M. Gelfond, and A. Provetti. Representing actions: Laws, observations and hypotheses. *Journal of Logic Programming*, 1997.
5. A.J. Bonner and M. Kifer. Transaction logic programming. In *Int'l Conference on Logic Programming*, pages 257–282, Budapest, Hungary, June 1993. MIT Press.
6. A.J. Bonner and M. Kifer. Applications of transaction logic to knowledge representation. In *Proceedings of the International Conference on Temporal Logic*, number 827 in Lecture Notes in Artificial Inteligence, pages 67–81, Bonn, Germany, July 1994. Springer-Verlag.
7. A.J. Bonner and M. Kifer. Transaction logic programming (or a logic of declarative and procedural knowledge). Technical Report CSRI-323, University of Toronto, November 1995. `http://www.cs.sunysb.edu/~kifer/TechReports/transaction-logic.pdf` .
8. A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.
9. M. Gelfond and V. Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.
10. S. Hanks and D. McDermott. Nonmonotonic logic and temporal projection. *Artif. Intell.*, 33(3):379–412, 1987.
11. M. Kifer. FLORA-2: An object-oriented knowledge base language. The FLORA-2 Web Site. http://flora.sourceforge.net.
12. Peifung E. Lam, John C. Mitchell, and Sharada Sundaram. 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, pages 73–85, Berlin, Heidelberg, 2009. Springer-Verlag.
13. D. Pearce and G. Wagner. Logic programming with strong negation. In *Proceedings of the international workshop on Extensions of logic programming*, pages 311–326, New York, NY, USA, 1991. Springer-Verlag New York, Inc.
14. M. Rezk and M. Kifer. Representing $\mathcal{L}_1$ Domain Descriptions as Partially Defined Actions of Transaction Logic. in preparation, 2011.
15. Martin Rezk and Michael Kifer. Reasoning with actions in transaction logic, 2011. Available from `http://www.inf.unibz.it/~mrezk/techreportPAD.pdf`.
16. H. Turner. Representing actions in default logic: A situation calculus approach. In *In Proceedings of the Symposium in honor of Michael Gelfond's 50th birthday (also in Common Sense 96*, 1996.