On the Equivalence between the $\mathcal{L}_1$ Action Language and Partial 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. Transaction Logic with Partially Defined Actions (TR^{PAD}) is an expressive formalism for reasoning about the effects of actions and for declarative specification of state-changing transactions. The action language $\mathcal{L}_1$ is a well-known formalism to describe changing domains and for reasoning about actions. The purpose of this paper is to compare these two formalisms and identify their similarities and points of divergence in order to better understand their modeling and reasoning capabilities. We provide a sound reduction of a large fragment of $\mathcal{L}_1$ to TR^{PAD}, and show that this reduction is complete with respect to the LP embedding of $\mathcal{L}_1$. We also explore how action planning is modeled in both languages and discuss the relationship to other languages for representing actions.

1 Introduction

Designing agents that can reason about actions has been a long-standing target in AI. 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 $\mathcal{A}$[5], $\mathcal{L}_1$[1], $\mathcal{C}$[6], $\mathcal{ALM}$[7]. Unfortunately, most of such languages have their weak points along with the strong ones, and neither is sufficient as a logical foundation for agents. Another area where action theories are important is Semantic Web Services, since it is necessary to reason about the effects of actions in order to discover, contract, and enact such services automatically [2,8].

Recently, another powerful language for actions, based on partially defined actions in Transaction Logic (abbr., TR^{PAD}), was proposed [12]. This language is based on a very different logical paradigm than the aforesaid languages, and it is an interesting challenge to understand the relative expressive power of these languages.

In this paper we identify and compare the modeling and reasoning capabilities of TR^{PAD} and $\mathcal{L}_1$. We chose $\mathcal{L}_1$ because it is a powerful language that can serve as a good representative of the family of action languages mentioned earlier. However, we also briefly discuss the relation between TR^{PAD} and action languages $\mathcal{C}$ and $\mathcal{ALM}$.

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

Example 1 (Health Insurance). The problem is to encode the following set of health insurance regulations. (i) For vaccination to be *compliant*, doctors must require that patients obtain authorization from their health insurers *prior* to vaccination. (ii) To obtain authorization, the patient must first visit a doctor (or be a doctor). (iii) Vaccinating a *healthy* patient makes her *immune* and *healthy*. (iv) A patient who has a *flu* is not healthy, but (v) flu can be treated with *antivirals*. In addition, we know that (vi) there is a patient, John, who has a *flu*, is not *immune* and (vii) is a *doctor*. We want to find a legal sequence of actions (a plan) to make John immune and healthy. □

We show how limitations of each language can be worked around to represent the above problem. Then we venture into the domain of action planning and discuss how it is done in each language. We show that certain planning goals, those that require intermediate conditions in order to construct legal plans, cannot be easily expressed in $\mathcal{L}_1$ and that they are very natural in TR^{PAD}. It is interesting to note that the above problem was discussed in [9] where a fragment of these health regulations was formalized in Prolog. However, that approach had troubles dealing with temporal and state-changing regulations, and it could not reason about the effect of actions. For instance, that approach had difficulty finding law-compliant sequences of actions that could achieve goals of the type described in the example.

This paper is organized as follows. Section 2 presents the necessary background on TR^{PAD} and $\mathcal{L}_1$. Section 3 illustrates similarities and differences between the two formalisms by means of non-trivial examples. Section 4 studies a fragment of $\mathcal{L}_1$ and reduces it to TR^{PAD}. We show that the reduction is sound with respect to the $\mathcal{L}_1$ semantics and complete with respect to the logic programming embedding of $\mathcal{L}_1$. Section 5 discusses planning problems in both formalisms and Section 6 compares TR^{PAD} with other popular action languages: $\mathcal{ALM}$ and $\mathcal{C}$. Section 7 concludes the paper.

2 Preliminaries

This section provides a brief introduction to TR^{PAD} — Transaction Logic with Partially Defined Actions. Details can be found in [12].[1]

2.1 Transaction Logic with Partially Defined Actions

TR^{PAD} [12] is a logic for programming actions and reasoning about them. It is an extension of a Horn dialect of Transaction Logic [3,4]. Like TR, TR^{PAD} contains logical connectives from the standard FOL ($\wedge, \vee, \forall, \exists$,) plus two additional logical connectives: the *serial conjunction*, $\otimes$, and the modal operator for hypothetical execution, $\Diamond$. 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. A hypothetical formula, $\Diamond\phi$, represents an action where ϕ is *tested* whether it can be executed at the current state, but no actual changes to the current state take place. For instance, the first part of the following formula $\Diamond(\mathit{insert}(\mathsf{vaccinated} \wedge \mathsf{allergic}) \otimes \mathit{bill\_insurance} \otimes \mathsf{has\_paid}) \otimes \mathit{vaccinate}$ 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

[1] A short version has been submitted to this conference.

will assume that hypothetical formulas contain only serial conjunctions of literals. The alphabet of the language $\mathcal{L}_{TR}$ of TR^{PAD} consists of countably infinite sets of variables $\mathcal{V}$, function symbols $\mathcal{F}$ (including 0-ary symbols, i.e., *constants*), predicates $\mathcal{P}$, and an infinite set of special constants called ***state identifiers***. These latter constants will be used to denote database states and we will be using boldface lowercase letters $\mathbf{d}$, $\mathbf{d}_1$, $\mathbf{d}_2$, to represent them. Finite sequences of such identifiers, e.g., $\mathbf{d}_1$, ..., $\mathbf{d}_n$, are called ***path abstractions***. Terms are defined as usual in first order logic. The set of predicates $\mathcal{P}$ is further partitioned into $\mathcal{P}_{fluents}$ and $\mathcal{P}_{actions}$. The former set contain predicates that represent facts in database states and the latter contains predicates for transactions that change those states. Fluents can be viewed as actions that do not change the underlying database state. In addition, TR^{PAD} has a form of negation that is called *explicit negation* (or *strong* negation) [10], denoted $\mathbf{neg}$. This negation is a weaker form of classical negation, and it applies only to fluents, not actions.

TR^{PAD} consists of *serial-Horn* rules, *partial action definitions* (PADs), and certain statements about states and actions, which we call *premises*. The syntax for all this is shown below, where $c, c_1, \ldots$ are *literals* (fluents or actions), f is a fluent literal, $b_1, b_2 \ldots$ are conjunctions of literals or hypotheticals, $\mathbf{d}, \mathbf{d}_1 \ldots$ are database state identifiers, and a is an action atom.

Rules		**Premises**	
(i) $c \leftarrow c_1 \otimes \cdots \otimes c_n$	(a serial-Horn rule)	(iii) $\mathbf{d} \rhd f$	(a **state**-premise)
(ii) $b_1 \otimes a \otimes b_2 \rightarrow b_3 \otimes a \otimes b_4$	(a PAD)	(iv) $\mathbf{d}_1 \overset{a}{\rightsquigarrow} \mathbf{d}_2$	(a **run**-premise)

The serial-Horn rule (i) is a statement that defines c. The literal c is a calling sequence for a complex transaction and $c_1 \otimes \cdots \otimes c_n$ is a definition for the actual course of action to be performed by that transaction. If c is a fluent literal then we require that c_1, ..., c_n are also fluents. We then call c a ***defined fluent*** and the rule a *fluent rule*. Fluent rules are equivalent to regular Horn rules in logic programming. If c is an action, we will say that c is a ***compound action***, as it is defined by a rule. For instance, the serial-Horn rule *vaccinate_legally* $\leftarrow$ *request_authorization* $\otimes$ authorized $\otimes$ *vaccinate* defines a compound action *vaccinate_legally*. This action first requests authorization and, if granted, performs the actual vaccination. The PAD (ii) means that if we know that b_1 holds before executing action a and b_2 holds after, we can conclude that b_3 must have held before executing a and b_4 must hold as a result of a. (The serial conjunctions b_1, b_2, b_3, b_4 are mandatory.) For instance, the PAD, healthy$\otimes$*vaccinate* $\longrightarrow$ *vaccinate*$\otimes$immune, states that if a patient is healthy before being vaccinated, we can conclude the person will be immune after receiving the vaccine. Note that the serial conjunction $\otimes$ binds stronger than the implication, so the above statement should be interpreted as: (healthy $\otimes$ *vaccinate*) $\longrightarrow$ (*vaccinate* $\otimes$ immune)). To sum up, we distinguish two kinds of actions: ***partially defined actions*** (abbr., *pda*) and ***compound actions***. Partially defined actions cannot be defined by rules—they are defined by PAD statements only. In contrast, compound actions are defined via serial-Horn rules but not by PADs. Note that *pda*s can appear in the bodies of serial-Horn rules that define compound actions (see *vaccinate_legally* above) and, in this way, TR^{PAD} can compose larger action theories out of smaller ones in a modular way.

Premises are statements about the initial and the final database states (state premises) and about possible state transitions caused by partially defined actions (run-premises).

For example, $\mathbf{d}_1 \rhd \mathsf{healthy}$ says that the patient was healthy in state represented by $\mathbf{d}_1$, while $\mathbf{d}_1 \overset{vaccinate}{\leadsto} \mathbf{d}_2$ states that executing the PAD action *vaccinate* in state represented by $\mathbf{d}_1$ leads to the state associated with $\mathbf{d}_2$. A ***transaction*** is a statement of the form $?\text{-}\ \exists \bar{X}\phi$, 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* generalize the notion of queries in ordinary logic programming. For instance, $?\text{-}\ \mathsf{healthy} \otimes$ *vaccinate_legally* is a transaction that first checks if the patient is healthy; if so, the compound action *vaccinate_legally* is executed. Note that if the execution of the transaction *vaccinate_legally* cannot proceed (say, because authorization was not obtained), the already executed actions are undone and the underlying database state remains unchanged. A TR^{PAD} ***transaction base*** is a set of serial-Horn rules and PADs. 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.

Semantics. The semantics of TR^{PAD} is Herbrand, as usual in logic programming. The ***Herbrand universe*** $\mathcal{U}$ is a set of all *ground* (i.e., variable-free) terms in the language and the ***Herbrand base*** $\mathcal{B}$ is a set of all ground literals in the language. A database state, $\mathbf{D}$, is a set of fluent literals. The semantics defines *path structures*, which generalize the usual first-order semantic structures. The key idea in *TR* is that formulas are evaluated over paths and *not* over states like in classical or temporal logics. For instance, suppose that executing *vaccinate* in a state $\mathbf{D}_1$ leads to $\mathbf{D}_2$, and we know that the patient is immune in $\mathbf{D}_2$. We would not say that the formula $\phi =$ *vaccinate* $\otimes\, \mathsf{immune}$ is true in the state $\mathbf{D}_1$, as in temporal logics; instead, we say that ϕ is true on the path $\langle \mathbf{D}_1, \mathbf{D}_2 \rangle$. 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^{PAD} 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****, $\mathcal{M} = (\mathbf{M}, \Delta)$, is a pair of mappings such that:* $\mathbf{M}$ *assigns a subset of* $\mathcal{B}$ *to every path, and* Δ *assigns a database state to every state identifier.* $\mathbf{M}$ *must satisfy the following condition for every state* $\mathbf{D}$: $\mathbf{D} \subseteq \mathbf{M}(\langle \mathbf{D} \rangle)$. $\square$

Intuitively, Herbrand path structures in *TR* play a role similar to transition functions in temporal logics such as LTL or μ-Calculus: they are relations between states and actions. However, while transition functions take a state and an action and return a set of next-states, a Herbrand path structure takes paths of the form $\langle \mathbf{D}_1 \ldots \mathbf{D}_n \rangle$ and return sets of actions that are executable over those paths, i.e., starting at $\mathbf{D}_1$ there is an execution that leads to $\mathbf{D}_n$ (actions in *TR* can be non-deterministic). For instance, if we take a Herbrand path structure, $\mathcal{M} = (\mathbf{M}, \Delta)$, that satisfies the specifications in our example, we could conclude *vaccinate* $\in \mathcal{M}(\langle \Delta(\mathbf{d}_1), \Delta(\mathbf{d}_2) \rangle)$ and $\mathsf{immune} \in \mathcal{M}(\langle \Delta(\mathbf{d}_2) \rangle)$.

The following definition formalizes the idea that truth of *TR* (and TR^{PAD}) formulas is defined on paths.

Definition 2 (Satisfaction). *Let $\mathcal{M} = (\mathbf{M}, \Delta)$ be a Herbrand path structure, π be a path, and let ν be a variable assignment $\mathcal{V} \longrightarrow \mathcal{U}$.*

- **Base case:** *If p is a literal then $\mathcal{M}, \pi \models_\nu p$ if and only if $\nu(p) \in \mathbf{M}(\pi)$.*
- **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.*
- **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 $\mathbf{D}$.*
- **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 $\mathcal{M}, \pi \models_\nu \phi$.*
- ***Conjunction, disjunction, quantification*** *are defined similarly to FOL. For instance, $\mathcal{M}, \pi \models_\nu \phi \wedge \psi$ iff $\mathcal{M}, \pi \models_\nu \phi$ and $\mathcal{M}, \pi \models_\nu \psi$ (see [12]).*

If $\mathcal{M}, \pi \models \phi$, then we say that ϕ is satisfied (or is true) on path π in structure $\mathcal{M}$. □

Since we work with propositions in the examples, we omit the variable assignments.

Definition 3 (Model). *A Herbrand path structure, $\mathcal{M} = (\mathbf{M}, \Delta)$, 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 Herbrand path structure, $\mathcal{M}$, is a* ***model of a premise statement*** *σ iff:*
– σ is a run-premise of the form $\boldsymbol{d}_1 \overset{\alpha}{\rightsquigarrow} \boldsymbol{d}_2$ and $\mathcal{M}, \langle \Delta(\boldsymbol{d}_1)\Delta(\boldsymbol{d}_2) \rangle \models \alpha$; or
– 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 specification $(\mathbf{P}, \mathcal{S})$ if it satisfies every formula in $\mathbf{P}$ and every premise in $\mathcal{S}$.* □

Executional entailment relates the semantics of TR^{PAD} to the notion of execution.

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

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

is said to be true if and only if $\mathcal{M}, \langle \mathbf{\Delta}(\boldsymbol{d}_0) \ldots \mathbf{\Delta}(\boldsymbol{d}_n) \rangle \models \phi$ for every model $\mathcal{M}$ of $(\mathbf{P}, \mathcal{S})$. We say that $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_0 --- \models \phi$, is true iff there is a database sequence $\mathbf{D_0} \ldots \mathbf{D_n}$ that makes (1) true with $\Delta(\boldsymbol{d}_0) = \mathbf{D}_0$. □

For reasoning about actions, one often needs the *frame axioms* (a.k.a. inertia axioms). These axioms say that things stay the same unless there is an explicitly stated reason for a change. To incorporate the frame axioms in the semantics, transaction bases in TR^{PAD} specifications are augmented with ***action theories***. An action theory for a given TR^{PAD} transaction base $\mathbf{P}$ is a set $\mathcal{A}(\mathbf{P})$, which contains $\mathbf{P}$ and and a set of PADs that represent the frame axioms. The exact form of these axioms is given in [12]. Action theories in TR^{PAD} can be selective as to when and whether the fluents are subject to laws of inertia. This is specified using the predicate inertial, which determines which facts are supposed to be inertial in which state. For instance, consider the action *vaccinate* shown above. One of the frame axioms for immune would be: $\mathsf{inertial(immune)} \otimes \mathbf{neg}\, \mathsf{healthy} \otimes \mathbf{neg}\, \mathsf{immune} \otimes \mathit{vaccinate} \rightarrow \mathit{vaccinate} \otimes \mathbf{neg}\, \mathsf{immune}$. This

PAD says that if a patient is not immune and is unhealthy, vaccination will not have the expected effect and the patient will remain not immune. One restriction on PADs is that they cannot have *interloping* actions. Two PADs are said to be ***interloping*** if they share a common primitive effect. TR^{PAD} itself does not need this restriction, but to define the frame axioms it does. Note that the above frame axiom for immune, would not be sound in presence of interloping actions. To see this, suppose that we have the following interloping PADs for *vaccinate*:

$$\mathsf{healthy} \otimes \mathit{vaccinate} \longrightarrow \mathit{vaccinate} \otimes \mathsf{immune}$$
$$\mathbf{neg}\,\mathsf{healthy} \otimes \mathit{vaccinate} \longrightarrow \mathit{vaccinate} \otimes \mathsf{immune}$$

These axioms say that regardless of the patient's health the vaccine will be effective. Clearly, the above frame axiom for **neg** immune and these rules contradict each other. Due to the lack of space, we will not provide details about how this restriction is reflected in the frame axioms. The readers are referred to [12]. In this paper, we will assume that TR^{PAD} specifications do *not* contain interloping PADs.

Reasoning. TR^{PAD} has a sound and complete proof theory and much of TR^{PAD} can be reduced to regular logic programming. This reduction will be used to compare the reasoning capabilities of TR^{PAD} and $\mathcal{L}_1$. The readers are referred to [12].

2.2 Action Language $\mathcal{L}_1$

This section reviews the basics of the action language $\mathcal{L}_1$ [1]. 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 set **S** contains two special situations: s_0, which is called the **initial situation**, and s_C, called the **current situation** (which is also the last one). The language $\mathcal{L}_1$ contains two kinds of propositions: *causal laws* and *facts*. In the following table, each $f, f_1 \dots f_n$ is a fluent literal, each s_i is a situation, a is an action and α is a sequence of actions.

Causal laws	**Atomic Facts**	
	(2) α **occurs_at** s	(occurrence fact)
(1) a **causes** f **if** $f_1 \dots f_n$ (causal law)	(3) f **at** s	(fluent fact)
	(4) s_1 **precedes** s_2	(precedence fact)

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

It is worth noting that with disjunction one can express *possible states* of the world and non-determinism. For instance, we could say that in the initial state the patient is either healthy or has a flu: healthy **at** $s_0 \lor$ flu **at** s_0, this is not expressible in TR^{PAD}. One can also state that in the initial state, either the patient is vaccinated or the doctor asks for an authorization *vaccinate* **occurs_at** $s_0 \lor$ *request_authorization* **occurs_at** s_0. With *conjunctions* of occurrence-facts we could express concurrency of action executions as in *vaccinate* **occurs_at** $s_0 \wedge$ *request_authorization* **occurs_at** s_0. However, the semantics (cf. Definition 6) does not support concurrent execution of actions at the same state. Unfortunately, the boost of expressivity coming from the propositional combination of atomic facts cannot be exploited for reasoning. This is because the reduction of $\mathcal{L}_1$ to LP works only when such combinations are disallowed.

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

Definition 5 (Situation Assignment). *A **situation assignment** of $\mathcal{D}$, $sit2act$, is a partial function from situations to sequences of actions such that:*

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

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

Definition 6 (Action Interpretation). *An **action interpretation** of $\mathcal{D}$, $act2st$, is a partial function from sequences of actions to states such that:*

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

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

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

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

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

Definition 7. ***(Model)** A **model** M of $\mathcal{L}_1$, is a pair $(act2st, sit2act)$, where*

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

The **actual path** of a model M for a domain description $\mathcal{D}$ is $sit2act(s_{\mathbf{C}})$.[2] Intuitively, it represents the unique sequence of actions defined by M and consistent with $\mathcal{D}$.

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

Definition 8. ***(Satisfaction)*** *For any model* $M = (act2st, sit2act)$

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

Since fluent facts can be expressed as hypothesis, we focus on just this kind of statements. $\mathcal{L}_1$ semantics has a minimality condition on the situation assignments of $\mathbf{S}$ that formalizes the informal assumption that an action is executed only if it is required by the domain description. Further details can be found in [1]. Definition 9 describes the set of acceptable conclusions obtainable from a domain description $\mathcal{D}$.

Definition 9. ***(Entailment)*** *A domain description* $\mathcal{D}$ ***entails*** *a query* q *(written as* $\mathcal{D} \models q$*) iff* q *is true in all minimal models of* $\mathcal{D}$*. We will say that the* ***answer*** *given by* $\mathcal{D}$ *to a query* q *is* yes *if* $\mathcal{D} \models q$, no *if* $\mathcal{D} \models \neg q$, unknown *otherwise.* □

Reasoning. $\mathcal{L}_1$ does not come with a proof system that allows reasoning, unlike TR^{PAD}. Thus, it is necessary to reduce $\mathcal{L}_1$ domain descriptions to some other formalism that supports reasoning. In [1], a reduction, of *simple* domain descriptions to logic programming is given. Intuitively, a domain description $\mathcal{D}$ is simple if (i) the situations $s_0 \dots s_k$ in $\mathcal{D}$ are linearly ordered with respect to **precedes** ; (ii) $\mathcal{D}$ has a unique actual path of the form $[a_1 \dots a_{k-1}]$ where each a_i occurs in s_i, and s_k is the current situation $s_{\mathbf{C}}$; (iii) $\mathcal{D}$ is consistent; (iv) all facts in $\mathcal{D}$ are atomic; and (v) $\mathcal{D}$ does not contain inconsistent causal laws.

3 Motivating Examples

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

Example 2 (Health Insurance (cont'd)). Consider the US health insurance regulations scenario of Example 1. The specification $\mathcal{T}_H = (\mathbf{P}, \mathcal{S})$ in Figure 1 shows a TR^{PAD} representation of that scenario. In the rules in Figure 1, *vaccinate_legally*, *request_authorization*,

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

vaccinate and *take_antivir* are actions, while healthy, flu, visited_dr, doctor, immune and authorized are fluents. Note that the two PADs defining *request_authorization* are interloping, which is discussed below. For clarity, each statement is numbered with the corresponding regulation number from Example 1.

$$\mathbf{P} = \begin{cases} (i) & \textit{vaccinate\_legally} \leftarrow \textit{request\_authorization} \otimes \text{authorized} \otimes \textit{vaccinate} \\ (ii) & \text{doctor} \otimes \textit{request\_authorization} \rightarrow \textit{request\_authorization} \otimes \text{authorized} \\ (ii) & \text{visited\_dr} \otimes \textit{request\_authorization} \rightarrow \textit{request\_authorization} \otimes \text{authorized} \\ (iii) & \text{healthy} \otimes \textit{vaccinate} \rightarrow \textit{vaccinate} \otimes \text{immune} \otimes \text{healthy} \\ (iv) & \mathbf{neg}\,\text{healthy} \leftarrow \text{flu} \\ (v) & \textit{take\_antivir} \rightarrow \textit{take\_antivir} \otimes \text{healthy} \otimes \mathbf{neg}\,\text{flu} \end{cases}$$
$$\mathcal{S} = \begin{cases} (vi) & \mathbf{d}_1 \rhd \text{flu} \\ (vi) & \mathbf{d}_1 \rhd \mathbf{neg}\,\text{immune} \\ (vii) & \mathbf{d}_1 \rhd \text{doctor} \end{cases}$$

Fig. 1. TR^{PAD} formalization of the health care scenario

The corresponding domain description $\mathcal{D}_H$ for the language $\mathcal{L}_1$ is shown in Figure 2. Since fluent rules are not allowed in $\mathcal{L}_1$, we manually encoded the consequence of regulation *(iv)* in the initial state. □

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

Fig. 2. $\mathcal{L}_1$ formalization of the health care scenario

Example 3 (Health Insurance (cont'd)). Consider Example 2 extended with the following additional information.

(viii) executing *take_antivir* in the initial state leads to state 1.
(ix) executing *vaccinate* in state 1 leads to state 2.

This additional information is shown in Figure 3 for both languages. In TR^{PAD}, we can use the inference system to derive $\mathbf{P}, \mathcal{S}, \mathbf{d}_0 \text{---} \models$ *take_antivir* $\otimes$ *vaccinate* $\otimes$ healthy, meaning that the patient becomes healthy as a result. In $\mathcal{L}_1$, the reduction of $\mathcal{D}_H$ to logic programming, $\Pi_{\mathcal{D}_H}$, can be used to establish the same thing (albeit in different notation): $\mathcal{D}_H \models \text{healthy}\ \mathtt{after}\ [\textit{take\_antivir}, \textit{vaccinate}]\ \mathtt{at}\ s_0$. □

Discussion. The formalizations of Example 1 in $\mathcal{L}_1$ and TR^{PAD} are not equivalent, however. For example, $\mathcal{L}_1$ does not allow compound actions, making the representation of

$$\mathcal{T}_H = \begin{cases} (viii) - \mathbf{d}_0 \overset{\textit{take\_antivir}}{\rightsquigarrow} \mathbf{d}_1 \\ (ix) \quad - \mathbf{d}_1 \overset{\textit{vaccinate}}{\rightsquigarrow} \mathbf{d}_2 \end{cases} \quad \mathcal{D}_H = \begin{cases} (viii) - \textit{take\_antivir}\ \mathbf{occurs\_at}\ s_0 \\ (viii) - s_0\ \mathbf{precedes}\ s_1 \\ (ix) \quad - \textit{vaccinate}\ \mathbf{occurs\_at}\ s_0 \\ (ix) \quad - s_1\ \mathbf{precedes}\ s_2 \end{cases}$$

Fig. 3. Describing states and executions in TR^{PAD} and $\mathcal{L}_1$

$$\mathcal{D}_H = \begin{cases} (i) - \textit{vaccinate\_legally} \leftarrow \textit{request\_authorization}_1 \otimes \text{authorized} \otimes \textit{vaccinate} \\ (i) - \textit{vaccinate\_legally} \leftarrow \textit{request\_authorization}_2 \otimes \text{authorized} \otimes \textit{vaccinate} \\ (ii) - \text{doctor} \otimes \textit{request\_authorization}_1 \rightarrow \textit{request\_authorization}_1 \otimes \text{authorized} \\ (ii) - \text{visited\_dr} \otimes \textit{request\_authorization}_2 \rightarrow \textit{request\_authorization}_2 \otimes \text{authorized} \end{cases}$$

Fig. 4. Replacing interloping actions in TR^{PAD}

the problem a little harder and less modular, precluding the possibility for expressing dependencies between fluents. On the other hand, TR^{PAD} does not support interloping actions in the definition of *request_authorization*. This constraint can be circumvented using the transformation in Figure 4, but this comes at the expense of readability.

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

4 Representing $\mathcal{L}_1^{\mathrm{A}}$ in TR^{PAD}

In this section we define a reduction for the *accurate* fragment of $\mathcal{L}_1$, denoted $\mathcal{L}_1^A$ to TR_D^{PAD}, and provide a soundness proof. The *accurate* fragment of $\mathcal{L}_1$ is defined as $\mathcal{L}_1$ except that it allows only *simple* domain descriptions and disallows interloping causal laws. Although the reduction presented here is not complete with respect to $\mathcal{L}_1^A$, we show that it is complete with respect to the LP reduction of $\mathcal{L}_1$ developed in [1]. Let $\mathcal{D}$ be a simple $\mathcal{L}_1^A$ domain description. Given an alphabet $\mathcal{L}_\mathcal{D}$ of $\mathcal{D}$, the corresponding language $\mathcal{L}_{TR}$ of the target TR^{PAD} formulation will consist of symbols for actions and fluents literals from $\mathcal{L}_\mathcal{D}$, except that the symbol $\neg$ in $\mathcal{L}_1^{\mathrm{A}}$, is replaced with **neg** in $\mathcal{L}_{TR}$. In the remainder of the section, let $\mathcal{D}$ be a simple domain description in $\mathcal{L}_1^{\mathrm{A}}$. Since simple domain descriptions have an explicit linear order over the situations and actions, we can disregard the **precedes** facts from the reduction, as they become redundant. Thus, we only translate the remaining laws and facts. The reduction $\Lambda(\mathcal{D}) = (\mathbf{P}, \mathcal{S})$ of $\mathcal{D}$ is defined in Figure 5. For the reduction, we map each situation s_i in $\mathcal{D}$ to the state identifier $\mathbf{d}_i$. In addition, we postulate that every fluent in $\Lambda(\mathcal{D})$ is inertial in every database state, and we include in the reduction the action theory of $\mathbf{P}$. Recall that the action theory $\mathcal{A}(\mathbf{P})$ of a transaction base $\mathbf{P}$ consists of $\mathbf{P}$

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

Fig. 5. Reduction for the accurate fragment of $\mathcal{L}_1$ to $\mathcal{T}\mathcal{R}_D^{\scriptscriptstyle PAD}$

and the frame axioms. Note that the translation of occurrence-facts of $\mathcal{L}_1$ takes care of both the facts of the form a_i **occurs_at** s_i and s_i **precedes** a_{i+1} since, by definition, these two types of formulas are encoded in the order of actions in an explicit actual path. Recall that the query language in $\mathcal{L}_1$ consists of hypotheses of the form $q = f$ **after** $[a_1, a_2, \ldots, a_n]$ **at** s_i. Therefore, to check soundness we restrict our query language to statements of the form: $a_1 \otimes a_2 \otimes \cdots \otimes a_n \otimes f$.

Theorem 1. *(Soundness) Let $\mathcal{D}$ be a simple domain description. Let $\Lambda(\mathcal{D}) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{T}\mathcal{R}^{\scriptscriptstyle PAD}$ reduction of $\mathcal{D}$, and α be a serial conjunction of actions. Suppose that $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_i \ldots \boldsymbol{d}_n \models \alpha \otimes f$. Then $\mathcal{D} \models f$* **after** *α* **at** *s_i.*

The converse of Theorem 1, *completeness*, does not hold. However, $\Lambda(\mathcal{D})$ is complete with respect to the reduction of $\mathcal{L}_1^{\mathrm{A}}$ to logic programming, $\Pi_{\mathcal{D}}$, developed in [1] (which, we remind, is also incomplete). We will write $\Pi_{\mathcal{D}} \models true\_after(f, \alpha, s_i)$ if $\mathcal{D}$ entails the fact that f holds after executing α in s_i.

Theorem 2. *(Completeness relative to $\Pi_{\mathcal{D}}$) Let $\mathcal{D}$ be a simple domain description. Let $\Lambda(\mathcal{D}) = (\mathbf{P}, \mathcal{S})$ be the $\mathcal{T}\mathcal{R}_D^{\scriptscriptstyle PAD}$ reduction of $\mathcal{D}$. Suppose $\Pi_{\mathcal{D}} \models true\_after(f, \alpha, s_i)$. Then $\mathbf{P}, \mathcal{S}, \boldsymbol{d}_i \models \alpha \otimes f$.*

The proofs for the soundness and completeness theorems can be found in [11].

5 Planning: $\mathcal{L}_1$ vs $\mathcal{T}\mathcal{R}^{\scriptscriptstyle PAD}$

We now turn to the problem of planning agents' actions and compare the capabilities of $\mathcal{T}\mathcal{R}^{\scriptscriptstyle PAD}$ and $\mathcal{L}_1$. With a slight abuse of the language we will often refer to state identifiers just as states. In planning, one starts with an initial state and action specifications and seeks to find a sequence of actions that lead to a state with certain desirable properties. We will show how the two languages approach the corresponding modeling and reasoning tasks.

Planning in $\mathcal{L}_1$. In [1], the initial state and the goal are represented by fluent facts, and the action descriptions by causal laws. However, $\mathcal{L}_1$ is not expressive enough to encode a *planning strategy*, since it cannot express recursion which is needed to search over the space of all possible plans. To cope with this problem, $\mathcal{L}_1$ is embedded in LP and planning strategies are then expressed in that larger setting.

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

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

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

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

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

Planning in TR^{PAD}. In [3], it was shown that planning strategies can be represented directly as *TR* rules and transactions, and plans could then be found by simply executing suitable transactions. Here we extend that formulation to model *conditional* planning. We also make use of premise statements and PADs to support more complex planning problems, including planning in the presence of incomplete information. The purpose here is to demonstrate that planning is possible, not to find the shortest plan. Additional techniques found in [3], including "script-based planning" and "locking," can also be ported to TR^{PAD}. Suppose we have a planning problem consisting of *(i)* an initial state $\mathbf{d}_0$, *(ii)* a set of actions $\mathbf{A}$ consisting of n elementary actions defined as PADs and m compound actions, a set of PADs $\mathbf{P}_{PAD}$ and rules $\mathbf{P}_{rule}$ for the actions in $\mathbf{A}$. and *(iii)* a planning goal goal. Then, the TR^{PAD} representation of a planner consists of:

1. A set of *state*-premises that model the knowledge about the initial state $\mathbf{d}_0$,
2. A new set of actions $\mathbf{A}' = \mathbf{A} \cup \mathbf{A}^s$, where $\mathbf{A}^s = \{a^s \mid a \text{ is a } pda \text{ in } \mathbf{A}\}$. The new actions a^s are *compound* (not *pdas*, although they are created out of *pdas*).
   The new set of PADs, $\mathbf{P'}_{PAD}$, has a PAD of the form $b_1 \otimes a \otimes b_2 \to b_3 \otimes a \otimes b_4 \otimes \mathsf{succeeded}_a$ for each PAD of the form $b_1 \otimes a \otimes b_2 \to b_3 \otimes a \otimes b_4$ in $\mathbf{P}_{PAD}$. Intuitively, $\mathsf{succeeded}_a$ is used to test that the *pda* a was executed successfully.
   The set of rules $\mathbf{P'}_{rule}$ contains a rule r' for every rule r in $\mathbf{P}_{rule}$, where r' is an exact copy of r except that each *pda* $a \in \mathbf{A}$ that occurs in the body of r is replaced in r' with the corresponding compound action $a^s \in \mathbf{A}^s$. Also, for each pda $a \in \mathbf{A}$ and the corresponding new compound action $a^s \in \mathbf{A}^s$, $\mathbf{P'}_{rule}$ has a new rule of the form $a^s \leftarrow a \otimes \mathsf{succeeded}_a$.
3. $\mathbf{A}'$ has two additional compound actions: *plan* and *act*. Intuitively, *act* is a generic action and *plan* represents sequences of such actions. $\mathbf{P'}_{rule}$ includes additional rules that define *plan* and *act*, as shown in the middle portion of Figure 6.
4. A set of *run*-premises that encodes the possible executions of the *pdas*, described in the bottom part of the figure.
5. A transaction of the form *plan* $\otimes$ goal, whose execution is expected to produce the requisite plans.

The key features of TR^{PAD} that enable this sort of general representation of planning are: *(i)* Premises and PADs are used to describe the content of the initial state, the frame axioms, and the effects of the actions; *(ii)* Compound Actions, which allow combining simpler actions into complex ones in a modular way; *(iii)* Recursion allows the inference system to use the generate-and-test method for plans; *(iv)* Non-determinism allows

Object	TR^{PAD} **Formulas**	**Notes**
Initial State	$\mathbf{d}_0 \rhd f$	for each fluent $f{\in}\mathbf{d}_0$
PADs + some rules	$b_1 \otimes a \otimes b_2 \rightarrow$ $b_3 \otimes a \otimes b_4 \otimes \mathsf{succeeded}_a$ $a^s \leftarrow a \otimes \mathsf{succeeded}_a$	for each PAD in $\mathcal{A}$ of the form $b_1 \otimes a \otimes b_2 \rightarrow$ $b_3 \otimes a \otimes b_4$
Planner rules	*plan* $\leftarrow$ *act* $\otimes$ *plan* *act* $\leftarrow a_1^s$ $\ldots$ *act* $\leftarrow a_n^s$ *act* $\leftarrow c_1$ $\ldots$ *act* $\leftarrow c_m$ *act* $\leftarrow$ *skip*	where $c_1 \ldots c_m \in \mathcal{A}$ are compound actions where each pda a is replaced by a^s; and *skip* is an action that does not cause a state change.
Possible Executions	$\mathbf{d}_r \overset{a}{\rightsquigarrow} \mathbf{d}_{r'}$	for each pda $a \in \mathcal{A}$ and a sequence r consisting of the indices $\{1 \ldots n\}$
Planning goal goal	$?-$ *plan* $\otimes$ goal	As a transaction

Fig. 6. Encoding Planning in TR^{PAD}

actions to be executed in different ways and, together with recursion, supports exploration of the search space of all possible plans. Finally, the last two features also enable one to define rules that produce various heuristic-directed searches available in various advanced planning strategies.

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

Example 4 (Planning). Consider the specification $\mathcal{T}_H$ and the domain description $\mathcal{D}_H$ for the health care scenario of Example 2. Recall that our goal is to find a *legal* plan that makes the patient John (who is a flu-afflicted doctor and one who lacks immunity) into an immune and healthy person. That is, our planning goal is $g_1 = \mathsf{immune} \wedge \mathsf{healthy}$. We will examine the behavior of the planners in both formalisms.

The case of $\mathcal{L}_1$: The LP program would start checking if $\mathcal{D}_H \models g_1$ **`after`** $[\,]$. Since the goal is not satisfied in $\mathcal{D}_H$ it would increase N and try sequences of length 1. The planner will find the plan [*take_antivir*, *vaccinate*]. when $N = 2$. However, this plan is not compliant with the law as required. This "illegal" plan was found because the goal does not represent the meaning of *legal* plans. Unfortunately, the query language of $\mathcal{L}_1$ is not expressive enough to deal with the requirement that the patient must obtain an authorization *before* vaccination. One could try the goal $g_1 \wedge \mathsf{authorized}$, but it is easy to see that this goal can lead to illegal plans as well. A solution could be to remove *vaccinate* from the action description to avoid the bad plans but this weakens the domain description and might block other desired inferences.

The case of TR^{PAD}: As described earlier in Figure 6, we need to transform the specification into the following (we show only the main parts, due to space limitation):

$$\begin{array}{l}
\mathsf{healthy} \otimes \mathit{vaccinate} \longrightarrow \mathit{vaccinate} \otimes \mathsf{immune} \otimes \mathsf{healthy} \otimes \mathsf{succeed}_{\text{vaccinate}} \\
\mathit{vaccinate}^s \longleftarrow \mathit{vaccinate} \otimes \mathsf{succeed}_{\text{vaccinate}} \\
\mathit{vaccinate\_legally} \longleftarrow \mathit{request\_authorization}^s \otimes \mathsf{authorized} \otimes \mathit{vaccinate}^s \\
\mathit{plan} \longleftarrow \mathit{act} \otimes \mathit{plan} \\
\mathit{act} \longleftarrow \mathit{vaccinate}^s \\
\mathbf{d}_{[\,]} \overset{vaccinate}{\rightsquigarrow} \mathbf{d}_{[1]}
\end{array}$$

Since we saw that the goal g_1 may lead to bad plans, we will modify the goal to specify that the patient can be vaccinated only after getting an authorization. This can be expressed as follows: $g_2 = \mathit{plan} \otimes (\mathbf{neg}\, \mathsf{immune} \wedge \mathsf{authorized}) \otimes \mathit{plan} \otimes (\mathsf{immune} \wedge \mathsf{healthy})$. The goal states that the planner must first try to obtain an authorization while the patient is still not immunized. Having achieved that, the planner will go on and plan for immunizing the patient and making her healthy. Note the ability of TR^{PAD} to specify intermediate conditions that the planner must achieve, not just the final goal. This is not possible in $\mathcal{L}_1$ without complicated encoding. The TR^{PAD} planner will construct a desired plan while proving the goal g_2 from the above specification at the initial state $\mathbf{d}_0$. Figure 7 illustrates how this works. The resulting plan will be *take_antivir* $\otimes$ *request_authorization* $\otimes$ *vaccinate*, which is equivalent to *take_antivir* $\otimes$ *vaccinate_legally*. □

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

Fig. 7. Planner execution in TR^{PAD}

6 Comparison with Other Action Languages

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

The $\mathcal{ALM}$ language [7]. This action language introduces the following features that $\mathcal{L}_1$ lacks: defined fluents, modular definition of actions, sorts, executability conditions and a form of concurrency. Although in $\mathcal{ALM}$ one can describe the effects and hierarchies of actions, and define fluents based on other fluents, one cannot (i) express the execution of actions like occurrence facts in $\mathcal{L}_1$ and *run*-premises in TR^{PAD} do, or (ii) assert information about the states, like fluent facts in $\mathcal{L}_1$ and *state*-premises in TR^{PAD} do. Recursion is disallowed for actions, but it is allowed for fluents.

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

The $\mathcal{C}$ language [6]**.** This language is based on the theory of causal explanation. That is, everything that is true in a state must be caused. This implies that the frame axioms are not part of the semantics but are expressed as axioms. In that sense, $\mathcal{TR}^{PAD}$ is closer to $\mathcal{C}$ than to $\mathcal{L}_1$. The language $\mathcal{C}$ is the simplest among the formalisms mentioned so far. It only allows causal laws and fluent definitions of the form: **caused** F **if** G and **causes** F **if** G **after** H, where F, G, H are propositional formulas, and only H can contain actions. Note that H may contain more than one action, which leads to concurrency in causal laws. Although causal laws can contain disjunctions in the rule conditions and effects, which is disallowed in PADs, in the propositional case disjunction can be modeled in $\mathcal{TR}^{PAD}$ by splitting rules. In this way, $\mathcal{TR}^{PAD}$ can model non-concurrent $\mathcal{C}$ domain description. In addition, [6] 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 $\mathcal{TR}^{PAD}$.

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

7 Conclusion

In this paper we explored and compared two expressive formalisms for reasoning about actions: $\mathcal{TR}^{PAD}$ and $\mathcal{L}_1$. We have shown that these formalisms have different capabilities and neither subsumes the other. Nevertheless, we established that a large subset of $\mathcal{L}_1$ can be soundly represented in $\mathcal{TR}^{PAD}$ and that the LP reduction of that subset is logically equivalent to the corresponding subset of $\mathcal{TR}^{PAD}$. We also compared the two logics in the domain of action planning and showed that $\mathcal{TR}^{PAD}$ has a significant advantage when it comes to planning under constraints. Finally, we briefly compared $\mathcal{TR}^{PAD}$ with two other action languages, $\mathcal{ALM}$ and $\mathcal{C}$.

We are planning to extend it with default negation, which will eliminate some of the restrictions such as the inability to handle interloping PADs. This extension will also make $\mathcal{TR}^{PAD}$ into a production rules style language and will provide formal basis for a large subset of that paradigm.

Acknowledgments. We thank 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. Baral, C., Gelfond, M., Provetti, A.: Representing actions: Laws, observations and hypotheses. Journal of Logic Programming (1997)
2. Berardi, D., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., Su, J., Tabet, S.: SWSL: Semantic Web Services Language. Technical report, Semantic Web Services Initiative (April 2005), `http://www.w3.org/Submission/SWSF-SWSL/`

3. Bonner, A.J., Kifer, M.: 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`
4. Bonner, A.J., Kifer, M.: A logic for programming database transactions. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems. ch.5, pp. 117–166. Kluwer Academic Publishers, Dordrecht (March 1998)
5. Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. Journal of Logic Programming 17, 301–322 (1993)
6. Giunchiglia, E., Lifschitz, V.: An action language based on causal explanation: Preliminary report. In: Proc. AAAI 1998, pp. 623–630. AAAI Press, Menlo Park (1998)
7. Inclezan, D.: Modular action language ALM. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 542–543. Springer, Heidelberg (2009)
8. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: ISWC 2004 Semantic Web Services Workshop. CEUR Workshop Proceedings (November 2004)
9. Lam, P.E., Mitchell, J.C., Sundaram, S.: A formalization of HIPAA for a medical messaging system. In: Fischer-Hübner, S., Lambrinoudakis, C., Pernul, G. (eds.) TrustBus 2009. LNCS, vol. 5695, pp. 73–85. Springer, Heidelberg (2009)
10. Pearce, D., Wagner, G.: Logic programming with strong negation. In: Proceedings of the International Workshop on Extensions of Logic Programming, pp. 311–326. Springer-Verlag New York, Inc., New York (1991)
11. Rezk, M., Kifer, M.: On the equivalence between the L1 action language and partial actions in transaction logic (2011), `http://www.inf.unibz.it/~mrezk/techreportTRL1.pdf`
12. Rezk, M., Kifer, M.: Reasoning with actions in transaction logic (2011), `http://www.inf.unibz.it/~mrezk/techreportPAD.pdf`