Formalizing Production Systems with Rule-Based Ontologies

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 11794-4400, U.S.A.
kifer@cs.stonybrook.edu

Abstract. In this paper we proposed a new semantics for the combination of production systems with *arbitrary DL* ontologies. Unlike previous approaches, the semantics presented here allow looping rules and can handle inconsistencies produced by the interaction of the rule actions and the ontology. We also define a sound embedding of such semantics, restricted to rule-based *DL* Ontologies, into Transaction Logic with partial action definitions ($\mathcal{TR}^{PAD}$). This reduction adds a declarative semantics to the combination. To model production systems in $\mathcal{TR}^{PAD}$, we extend $\mathcal{TR}^{PAD}$ with default negation and define the well-founded semantics for it.

Keywords: Transaction Logic, Well-founded semantics, Ontologies, Production Systems, Knowledge Representation.

1 Introduction

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

Such systems consist of a set of *production rules* that rely on forward chaining reasoning to update the underlying database, called *working memory*. Traditionally, PS have had only operational semantics, where satisfaction of rule conditions is checked using pattern matching, and rule actions produce assertion and deletions of facts from the working memory. PS syntax and semantics have been standardized as W3C's Production Rule Dialect of the Rule Interchange Format (RIF-PRD) [14]. The RIF-PRD specification has a number of limitations, however. First, it omits certain important primitives that are found in many commercial production systems such as IBM's *JRules*[19]. The *FOR*-loop and the *while*-loop constructs are examples of such an omission. Second, RIF-PRD still does not integrate with ontologies [3,18].

To illustrate the need for ontology integration, consider a set PS that keeps a number of clinical databases that are compliant with the health insurance regulations. The clinical record of each patient together with other data must be accessible by all the clinics in the network. This needs a shared vocabulary

T. Lukasiewicz and A. Sali (Eds.): FoIKS 2012, LNCS 7153, pp. 332–351, 2012.

that, in this case, is defined in a shared *DL* ontology. However, each PS can have extra concepts outside the ontology, which are meant for local use only. The following production rules state that *(i)* if a doctor D requests a DNA test T to be performed for patient P, then the system records that P is taking the test T; *(ii)* if a patient gets cured, then she cannot be unhealthy; and *(iii)* if a patient receives medicine, then she gets cured.

r_1: `Forall` D, P, T: `if` $\mathsf{requested}(D, P, T) \wedge \mathsf{dnaT}(T)$ `then` *Assert*$(\mathsf{takesT}(P, T))$
r_2: `For` P: $\mathsf{cured}(P)$ `do` *Retract*$(\mathbf{neg}\ \mathsf{healthy}(P))$
r_3: `Forall` P: `if` $\mathsf{rcv\_meds(P)}$ `then` *Assert*$(\mathsf{cured}(P))$

The *DL* ontology that defines the shared concepts and implements different constraints is as follows

$$\mathsf{flu} \sqsubseteq \mathbf{neg}\ \mathsf{healthy} \qquad \mathsf{dnaT} \sqsubseteq \mathbf{neg}\ \mathsf{virusT} \qquad \exists \mathsf{takesT}.\mathbf{neg}\ \mathsf{virusT} \sqsubseteq \mathsf{healthy}$$

The *DL* axioms say that a patient with a flu is not a healthy patient, that DNA tests do not search for viruses, and that if a person is taking a test not related with any virus disease, then we can conclude that she is healthy. Here we are using explicit negation **neg** [22] to say that patients taking the DNA test should not be considered unhealthy. This type of negation is the preferred way of adding explicit negative information in rule-based knowledge representation; it is weaker than classical negation, does not add complexity to the logic, and makes knowledge representation more natural. The `Forall` construct in r_1 should not be confused with the `For` construct in r_2. The former is just a way RIF-PRD declares variables used in the body of a rule. The latter is a FOR-loop extension found in commercial systems, but not in RIF-PRD.

The complexity of the regulations in our example makes it difficult to determine whether executing a production rule leaves the database in a compliant state. Suppose we have the following database $\mathsf{WM}_0 = \{\mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathsf{flu}(\text{Laura}), \mathsf{dnaT}(\text{pcr}), \mathsf{rcv\_meds(Laura)}\}$.

This example raises several question: *(i)* Suppose that we execute r_3 with P instantiated with *Laura*. How do we interpret the retraction executed by r_2, (with P again instantiated with *Laura*) given that $\mathbf{neg}\ \mathsf{healthy}(\text{Laura})$ is inferred by the ontology, *(ii)* how to interpret the rule conditions of r_1 and r_2 given the open world semantics of *DL*, and *(iii)* how do we treat the inconsistency that results after execution of rule r_1 in WM_0? (Observe that in the state resulting from execution of r_1 in WM_0 we can infer $\mathsf{healthy}(\text{Laura})$ and $\mathbf{neg}\ \mathsf{healthy}(\text{Laura})$.)

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

Our contribution in this paper is three-fold: *(i)* a new semantics for production systems augmented with DL ontologies that includes looping-rules, and can handle inconsistency; *(ii)* a sound embedding of the combination of PS and rule-based ontologies into *Transaction Logic with partially defined actions* (abbr., $\mathcal{TR}^{PAD}$) [25], which provides a model-theoretic semantics to the combination; *(iii)* an extension of $\mathcal{TR}^{PAD}$ with default negation under a variant of the well-founded semantics [29] for $\mathcal{TR}^{PAD}$.

Transaction logic [5,6,7] was chosen because it provides a natural platform that satisfies most of the requirements to model the combination of PS and ontologies. It was designed from the outset as a formalism for declarative specification of complex state-changing transactions in logic programming, and it has been successfully used for planning [6], knowledge representation [8], active databases [6], event processing [1], workflow management and Semantic Web services [11,12,27,28], and as a declarative alternative to non-logical features in Prolog, like the *assert* and *retract* operators [7], which are also present in production systems.

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

This paper is organized as follows. Section 2 briefly surveys previous results on the combination of PS and ontologies, and on the reduction of PS to formalisms with denotational semantics. Section 3 presents the necessary background on first order logic and description logic. Section 4 introduces an operational semantics for production systems augmented with DL ontologies. Section 5 augment $\mathcal{TR}^{PAD}$ with default negations, and provides a well-founded semantics for such extension. Section 6 provides a reduction from the semantics proposed here to $\mathcal{TR}^{PAD}$ and presents soundness results for this reduction. Section 7 concludes the paper. Proofs of the main results and further details are found in [24].

2 Related Work

In this section we compare our approach with other literature on the declarative semantics for production systems and on the operational and declarative semantics for the combination of PS and ontologies. The work described in [26,13] provides an operational and model-theoretic semantics to the combination of PS and ontologies. The model-theoretic semantics is given by an embedding of PS into fix-point logic. However, they cannot handle looping rules, their semantics cannot handle inconsistencies, their interpretation of retraction of *DL* facts is not intuitive since a fact can remain true after being deleted, and their reduction to a declarative formalism is considerably more complex than the one presented here. In [21,31], the goal is to devise languages for unifying some aspects of active rules, logic rules, and production systems. They do not deal with considerably more complex standard languages such as production systems augmented with ontologies and looping rules. In particular, [21,31] do not show how to embed production systems into those languages, although they provide some examples showing how typical production rules can be expressed in their language. In [23] the authors only allow a very restricted type of production systems: stratified PS. Such PS are much weaker that the ones formalized here, and again, they do not consider ontologies. In addition, they do not tackle the problem of the integration with ontologies. In [10,4], the authors reduce the semantics of PS to logic programming (LP). Their reduction is considerably more complex and less

compact than ours—it results in an infinite number of rules. In addition, they use stable models semantics which has much higher computational complexity than the well founded semantics used here. Given the complexity of such a reduction, the proposed integration with LP ontologies is not ideal, since the ontology needs to be transformed with state arguments and auxiliary predicates. In addition, neither of them allow looping rules. Finally, [20] presents a new formalism that combines some aspects of logic rules and production rules. However, negation in rule conditions[1] and looping rules are disallowed. Furthermore, their embedding into Horn Logic is less clear and compact than our embedding in $\mathcal{TR}^{PAD}$.

3 Preliminaries

In this Section we briefly review the basic notions from Description Logic (*DL*) that we will use throughout the paper. Details can be found in [3].

Description Logic is a family of knowledge representation formalisms that provide a syntax and a model-theoretic semantics for a compact representation of information. The alphabet of a *DL* language $\mathcal{L}$ includes a countably infinite disjoint sets of variables $\mathcal{V}$, constant symbols $\mathcal{C}$, and unary and binary predicate symbols (concepts and roles respectively) $\mathcal{P}$. $\mathcal{L}$ includes the logical connectives $\sqcup, \sqcap, \neg, \forall, \exists, \sqsubseteq$.

A *DL* knowledge base has two parts: the *TBox*, with terminological knowledge, which consists of a number of concept axioms, and the *ABox*, which consists of assertions about actual individuals. Concept axioms in the TBox are of the form $C \sqsubseteq D$, meaning the extension of C is a subset of the extension of D. Concepts and TBox axioms can be understood as formulas of first-order logic with one free variable and closed universal formulas respectively. Therefore, the semantics of *DL* can be given by its translation to FOL. Details can be found in [3]. To integrate ontologies with production systems, we will later start using Herbrand domains and the unique name assumption (UNA). UNA is a commonly made assumption in *DL*s literature as well [2].

Therefore, we will use the Herbrand semantics from the outset. As is well-known, this semantics is equivalent to the general one for universal clausal form.

The semantics defines *semantic structures*. The domain of a Herbrand semantic structure is called the Herbrand universe $\mathcal{U}$; in our restricted case it is just the set of all constants $\mathcal{C}$ in the language $\mathcal{L}$. The Herbrand base $\mathcal{B}$ is a set of all ground literals in the language.

Definition 1 (Semantic Structure). *A* semantic structure $\mathcal{I}$ *is a triple* $\langle \mathcal{U}, \mathbf{B}, \sigma \rangle$*, where*

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

[1] The authors informally claim that negation could be added, but they do not provide formal details.

The definitions of satisfaction and entailment are as usual.

In Section 6, we will use *DL*s that can be embedded into Logic Programming (LP). In particular, [17] defines a class of *DL*s called ***Datalog-rewritable*** DL*s*. This class is interesting in our setting because reasoning with Datalog-rewritable *DL*s can be reduced to reasoning with Datalog programs. Due to space limitation, we will omit the details of these *DL*s. Complete definitions and the relationship with OWL can be found in [17].

4 Production Systems Augmented with Ontologies

In this section we propose a new semantics for the combination of production systems and *arbitrary DL* ontologies. This approach follows the outline of [26], but includes looping rules, it can handle inconsistencies produced by the system, and it gives a more intuitive semantics to the retraction of *DL* facts.

4.1 Syntax

The alphabet of a language $\mathcal{L}_{\mathsf{PS}}$ for a production system is defined the same way as in the case of *DL* except that now the set of all predicates $\mathcal{P}$ is partitioned into two countably infinite subsets, $\mathcal{P}_{PS}$ and $\mathcal{P}_{DL}$. The latter will be used to represent predicates occurring in the ontology. A term is either a variable or a constant symbol and, to avoid unnecessary distractions, we will leave out the various additional forms allowed in RIF, such as frames and the RIF membership and subclass relations ($o\#t$, $t\#\#s$). However, they can easily be added without increasing the complexity of the problem. A **atomic formula** is a statement of the form $p(t_1 \dots t_n)$, where $p \in \mathcal{P}$. A literal is either an atom, a formula of the form $\mathbf{neg}\, f$ where f is a $\mathcal{P}_{DL}$-atom, or a formula of the form $\neg f$ where f is a $\mathcal{P}_{PS}$-atom.

A **condition formula** has one of the following forms: a literal l, $\phi_1 \wedge \phi_2$ or $\phi_1 \vee \phi_2$ where ϕ_1 and ϕ_2 are condition formulas. Observe that all the rule conditions in our example are condition formulas. An **atomic action** is a statement that has one the following forms:

- ***assert***(l): Adds the literal l to the working memory
- ***retract***(l) : $\begin{cases} \text{if } p \in \mathcal{P}_{PS} & \text{Removes the } atom^2 l \text{ from the working memory} \\ \text{if } p \in \mathcal{P}_{DL} & \text{Enforces the literal } l \text{ to be false in the working memory} \end{cases}$ □

Beside these elementary actions, RIF also provides actions to change or delete objects and properties. Such actions can be treated similarly to `FOR`-rules below or as sequences of simpler actions, so we leave them out as well.

Definition 2 (Production System Augmented with Ontology). *A production system augmented with ontology (abbr.,* production system*, or* PS*) is a tuple* $\mathsf{PS} = (\mathcal{T}, \mathsf{L}, R)$ *such that*

[2] Negative literals with predicate symbols in $\mathcal{P}_{PS}$ cannot occur in the working memories. See Definition 3.

– $\mathcal{T}$ *is a* DL *ontology (TBox) whose predicates belong to* $\mathcal{P}_{\text{DL}}$*;*
– L *is a set of rule labels, and*
– R *is a set of rules, which are statements of one of the following forms*[3]

$$\texttt{IF-THEN Rule:} \quad r\texttt{: Forall } \boldsymbol{x}\texttt{: if } \phi_r(\boldsymbol{x}) \texttt{ then } \psi_r(\boldsymbol{x}) \tag{1}$$

$$\texttt{FOR Rule:} \quad r\texttt{: For } \boldsymbol{x}\texttt{: } \phi_r(\boldsymbol{x}) \texttt{ do } \psi_r(\boldsymbol{x}) \tag{2}$$

where (i) $r \in \mathsf{L}$ *is the above rule's label, (ii)* ϕ_r *is a condition formula in* $\mathcal{L}$ *with free variables* $\boldsymbol{x}$*, and (iii)* $\psi_r(\boldsymbol{x})$ *is a sequence of atomic actions with free variables contained in* $\boldsymbol{x}$*.* □

4.2 Operational Semantics

We now turn to the operational semantics of the combination of PS with ontologies. In a PS, two different constants represent two different domain elements, which is called the *unique name assumption*. In addition, production systems assume that each constant symbol is also a symbol in the domain of discourse, i.e., they are dealing with Herbrand domains. It is also worth noting that the semantics presented in this section does not depend on the specifics of the DL associated with production systems.

Definition 3 (Working Memory). *A* ***working memory****,* WM*, for a PS language* $\mathcal{L}$ *is a disjoint union* $\mathsf{WM} = \mathsf{WM}_{PS} \uplus \mathsf{WM}_{\text{DL}}$ *where* WM_{PS} *is a set of ground atoms that use predicate symbols from* $\mathcal{P}_{PS}$ *and* WM_{DL} *is a set of ground literals that use predicate symbols from* $\mathcal{P}_{\text{DL}}$*.* □

Definition 4 ($\mathcal{T}$-structure). *Let* $\mathcal{T}$ *be a* DL *TBox. A* **$\mathcal{T}$-structure***,* $\mathcal{I}$*, for a PS language* $\mathcal{L}_{\mathsf{PS}}$ *has the form*

$$\mathcal{I} = (\mathsf{WM}_{PS} \uplus \mathsf{WM}_{\text{DL}} \uplus \mathbf{E}_{\text{DL}}, \sigma)$$

where $\mathsf{WM} = \mathsf{WM}_{PS} \uplus \mathsf{WM}_{\text{DL}}$ *is a working memory,* $\mathbf{E}_{\text{DL}}$ *is a set of* $\mathcal{P}_{\text{DL}}$*-literals,* σ *is a variable assignment, and* $(\mathsf{WM}_{\text{DL}} \uplus \mathbf{E}_{\text{DL}}, \sigma)$ *is a model of* $\mathcal{T}$*.* □

We say that (WM, σ), where WM is a working memory, is a ***prestructure***.

Example 1. In our running example, the two disjoint sets composing the initial working memory WM_0 are as follows:

$$\mathsf{WM}_{0\,PS} = \{\mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}),\ \mathsf{rcv\_meds(Laura)}\}.$$
$$\mathsf{WM}_{0\,DL} = \{\mathsf{flu}(\text{Laura}), \mathsf{dnaT}(\text{pcr})\}$$

In addition, we can build up a $\mathcal{T}$-structure, $\mathcal{I}$, by pairing any arbitrary assignment σ with WM_0 together with $\{\mathbf{neg}\,\mathsf{healthy}(\text{Laura})\}$. That is, $\mathcal{I} = (\mathsf{WM}_0 \uplus \{\mathbf{neg}\,\mathsf{healthy}(\text{Laura})\}, \sigma)$. □

[3] To avoid a misunderstanding, recall that the `Forall` construct is just a RIF-PRD syntax for declaring variables; it does not indicate a loop. In contrast, the `For-do` construct specifies a loop; it is found only in commercial PS systems, like JRules.

Definition 5 (Satisfaction). *A $\mathcal{T}$-structure $\mathcal{I} = (\mathsf{WM}_{PS} \uplus \mathsf{WM}_{\mathrm{DL}} \uplus \mathbf{E}_{\mathrm{DL}}, \sigma)$* satisfies *a literal l, denoted $\mathcal{I} \models l$, iff*

- *if l is a $\mathcal{P}_{\mathsf{PS}}$-atom then $l^{\mathcal{I}} \in \mathsf{WM}_{PS}$*
- *if l is a $\mathcal{P}_{\mathrm{DL}}$-literal then $\mathsf{WM}_{\mathrm{DL}} \uplus \mathbf{E}_{\mathrm{DL}} \models l^{\mathcal{I}}$*

*If ϕ is a formula of the form $\neg\phi_1$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ then we define $\mathcal{I} \models \phi$ as usual in FOL. A formula ϕ **holds** in a prestructure (WM, σ) relative to an ontology $\mathcal{T}$, denoted $\mathcal{T}, (\mathsf{WM}, \sigma) \models \phi$, iff $\mathcal{I} \models \phi$ for every $\mathcal{T}$-structure of the form $\mathcal{I} = (\mathsf{WM} \uplus \mathbf{E}_{\mathrm{DL}}, \sigma)$ (That is, WM and σ are fixed but the $\mathbf{E}_{\mathrm{DL}}$ varies.)* □

Example 2. Consider again the initial working memory WM_0 from our running example, and let (WM_0, σ) be a prestructure. Observe that *(i)* the formula $\neg\mathsf{requested}(\mathrm{Smith}, \mathrm{Laura}, \mathrm{pcr})$ holds in (WM_0, σ) but *(ii)* $\neg\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr})$ does not. We can conclude *(i)* because $\mathsf{requested}(\mathrm{Smith}, \mathrm{Laura}, \mathrm{pcr})$ is a $\mathcal{P}_{PS}$ atom and it does not belong to WM_0. On the other hand, *(ii)* follows since $\neg\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr})$ is a $\mathcal{P}_{DL}$ atom and there is a $\mathcal{T}$-structure with working memory WM_0 and assignment σ that satisfies $\neg\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr})$. Note that $\mathbf{neg}\,\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr})$ does not hold in (WM_0, σ) either. □

A prestructure is $\mathcal{T}$-***consistent*** if there is a $\mathcal{T}$-structure with the same working memory and variable assignment, i.e., $(\mathsf{WM} \uplus \mathbf{E}_{DL}, \sigma)$, that does not entail f and $\mathbf{neg}\, f$ for any atom f. A working memory is $\mathcal{T}$-consistent if it is part of a $\mathcal{T}$-consistent prestructure.

Definition 6 (Atomic Transition). *Let (WM, σ) be a prestructure, t_1, t_2 be terms, and α be an action. We say that there is an α-**transition** from (WM, σ) to (WM', σ), denoted $(\mathsf{WM}, \sigma) \overset{\alpha}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$, iff*

- *if $\alpha = \boldsymbol{assert}(p(t_1, t_2))$ then $\mathsf{WM}' = (\mathsf{WM} \cup \{p(t_1^\sigma, t_2^\sigma)\}) \setminus \{\mathbf{neg}\, p(t_1^\sigma, t_2^\sigma)\}$*
- *if $\alpha = \boldsymbol{retract}(p(t_1, t_2)])$ then* $\begin{cases} \text{if } p \in \mathcal{P}_{PS} & \mathsf{WM}' = \mathsf{WM} \setminus \{p(t_1^\sigma, t_2^\sigma)\} \\ \text{if } p \in \mathcal{P}_{\mathrm{DL}} & \mathsf{WM}' = (\mathsf{WM} \cup \\ & \{\mathbf{neg}\, p(t_1^\sigma, t_2^\sigma)\}) \setminus \{p(t_1^\sigma, t_2^\sigma)\} \end{cases}$

where t^σ is $\sigma(t)$ if t is a variable and it is t if t is a constant. □

In the remainder, we will write $(\mathsf{WM}_0, \sigma) \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$, to denote the sequence of transitions:

$$(\mathsf{WM}_0, \sigma) \overset{\alpha_1}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma) \overset{\alpha_2}{\twoheadrightarrow} (\mathsf{WM}_2, \sigma) \overset{\alpha_3}{\twoheadrightarrow} \ldots \overset{\alpha_{n-1}}{\twoheadrightarrow} (\mathsf{WM}_{n-1}, \sigma) \overset{\alpha_n}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

If, for some σ and $n \geq 1$, there is a transition $(\mathsf{WM}_0, \sigma) \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$ between prestructures then we will also write $\mathsf{WM}_0 \overset{\alpha_1 \ldots \alpha_n}{\twoheadrightarrow} \mathsf{WM}'$.

Suppose that there is a transition of the form $\mathsf{WM} \overset{\alpha}{\twoheadrightarrow} \hat{\mathsf{WM}}$ and $\hat{\mathsf{WM}}$ is not $\mathcal{T}$-consistent. The $\mathcal{T}$-**consistent result** of applying α to WM is the intersection of all the maximal subsets of $\hat{\mathsf{WM}}$ that contain $\hat{\mathsf{WM}} \setminus \mathsf{WM}$. This approach is known in the belief revision literature as *When in Doubt Throw it Out* (WIDTIO) [30]. This form of belief revision is in line with traditional ontologies and it has been also used in the context of evolution of *DL* knowledge bases [9].

Example 3. Suppose we execute r_1 in WM_0. We obtain the inconsistent working memory $\mathsf{WM}_1 = \{\mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathsf{takeT}(\text{Laura}, \text{pcr}), \mathsf{flu}(\text{Laura}), \mathsf{dnaT}(\text{pcr}), \mathsf{rcv\_meds}(\text{Laura})\}$. We have two maximal consistent subsets of WM_1:

- $\mathsf{WM}'_1 = \{\mathsf{takeT}(\text{Laura}, \text{pcr}), \mathsf{dnaT}(\text{pcr}), \mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathsf{rcv\_meds}(\text{Laura})\}$.
- $\mathsf{WM}'_1 = \{\mathsf{takeT}(\text{Laura}, \text{pcr}), \mathsf{flu}(\text{Laura}), \mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathsf{rcv\_meds}(\text{Laura})\}$.

Thus, the consistent result is:

$$\mathsf{WM}_1^{cons} = \{\mathsf{takeT}(\text{Laura}, \text{pcr}), \mathsf{requested}(\text{Smith}, \text{Laura}, \text{pcr}), \mathsf{rcv\_meds}(\text{Laura})\}\ \square$$

A **consistent transition**, denoted $(\mathsf{WM}_0, \sigma) \overset{\alpha}{\twoheadrightarrow} (\mathsf{WM}_1^{cons}, \sigma)$, is a transition where the result of applying α in (WM_0, σ) is replaced the $\mathcal{T}$**-consistent result** of that action.

The following definition formalizes the *conflict resolution strategy* for a given rule r.

Definition 7 (Fireability). *We say that a rule r is* ***fireable*** *in a prestructure (WM_0, σ) if and only if:*

- `IF-THEN`*: r is of the form (1), $\phi_r(\sigma(\boldsymbol{x}))$ holds in WM_0, and there is a $\mathcal{T}$-consistent transition of the form*

$$(\mathsf{WM}_0, \sigma) \overset{\psi_r(\sigma(\boldsymbol{x}))}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma)$$

- `FOR`*: r is of the form (2) and there are prestructures $(\mathsf{WM}_0, \sigma_0), (\mathsf{WM}_1, \sigma_0), (\mathsf{WM}_1, \sigma_1) \ldots (\mathsf{WM}_n, \sigma_{n-1})$ such that there are $\mathcal{T}$-consistent transitions of the form*

$$\begin{array}{c} (\mathsf{WM}_0, \sigma_0) \overset{\psi_r(\sigma_0(\boldsymbol{x}))}{\twoheadrightarrow} (\mathsf{WM}_1, \sigma_0) \\ (\mathsf{WM}_1, \sigma_1) \overset{\psi_r(\sigma_1(\boldsymbol{x}))}{\twoheadrightarrow} (\mathsf{WM}_2, \sigma_1) \\ \vdots \\ (\mathsf{WM}_{n-1}, \sigma_{n-1}) \overset{\psi_r(\sigma_{n-1}(\boldsymbol{x}))}{\twoheadrightarrow} (\mathsf{WM}_n, \sigma_{n-1}) \end{array} \tag{3}$$

  where the following conditions hold:

  1. Looping*: r's condition holds in each prestructure $(\mathsf{WM}_i, \sigma_i)$ ($0 \leq i \leq n-1$)*
  2. No repetitions*: For every pair of assignments σ_i, σ_j ($j \neq i$ and $0 \leq j, i \leq n-1$) we have that $\sigma_i \neq \sigma_j$. That is, assignments cannot be re-used in the same rule execution.*
  3. Termination*: There is no assignment σ such that it produces a $\mathcal{T}$-consistent transition from WM_n, and (WM_n, σ) satisfies r's condition.*

In both cases above we say that r **causes transition** *from WM_0 to WM_n and denote it as $\mathsf{WM}_0 \overset{r}{\hookrightarrow} \mathsf{WM}_n$.* $\square$

Recall that a PS applies rules in three steps: (1) pattern matching, (2) conflict resolution, (3) rule execution, and then it loops back to (1). So far we have

described only the steps (1) and (3). The next series of definitions describes Step (2) and show how looping is modeled in the semantics. This semantics does not depend on any particular conflict resolution strategy so, for concreteness, in Step (2) we will simply randomly choose a fireable rule from the conflict resolution set.[4] Some other works [4,13] use the same strategy.

The ***transition graph***, $\mathcal{T}_{\mathsf{PS}}$, of a production system is a directed labeled graph, whose set of nodes is the set of all working memories. There is an edge between two nodes WM and WM', labeled with α, σ for some action α and variable assignment σ, iff $(\mathsf{WM}, \sigma) \overset{\alpha}{\twoheadrightarrow} (\mathsf{WM}', \sigma)$. We will use $\mathcal{P}_{\mathsf{WM}}$ to denote the set of all paths (sequences of WMs) in the graph $\mathcal{T}_{\mathsf{PS}}$ starting at WM.

Definition 8 (Run). *A path π in $\mathcal{P}_{\mathsf{WM}_0}$ is a* **run** *$\mathcal{R}$ for a production system PS iff π can be split in paths $\pi_1, \ldots, \pi_n$ and there are rules $r_1 \ldots r_n$ such that for each $i = 1 \ldots n$, $WM_{i,start} \overset{r_i}{\hookrightarrow} WM_{i,end}$, where $WM_{i,start}$ is the first element in π_i and $WM_{i,end}$ is its last. Note that this implies that every π_i is a $\mathcal{T}$-consistent transition.* □

5 Extending $\mathcal{TR}^{PAD}$ with Default Negation

Transaction Logic with Partially Defined Actions [25], $\mathcal{TR}^{PAD}$, is a logic for programming actions and reasoning about them. In this section, we extend $\mathcal{TR}^{PAD}$ with *default* negation (a.k.a. negation as failure). Default negation allows a logic system to conclude the negation of any atom that the system unsuccessfully finishes exploring all possible proofs.

The alphabet of the language $\mathcal{L}_{\mathcal{TR}}$ of $\mathcal{TR}^{PAD}$ is defined the same way as in the *DL* case except that now the set of all 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 for transactions that change those states. Querying a fluent can be viewed as an action that does not change the underlying database state. We also add new symbols, $\rhd$ and $\overset{a}{\rightsquigarrow}$, where a is an atom whose predicate symbol is in $\mathcal{P}_{actions}$. Terms are defined as usual in first order logic. States are referred to with the help of special constants called ***state identifiers***; these will be usually denoted by boldface lowercase letters $\mathbf{d}$, $\mathbf{d}_1$, $\mathbf{d}_2$.

The symbol **neg** will be used to represent the explicit negation and **not** will be used for the *default* negation. These two symbols are applicable to fluents only. A fluent literal is either an atomic fluent or it has one of the following negated forms: $\mathbf{neg}\, f$, $\mathbf{not}\, f$, $\mathbf{not}\ \mathbf{neg}\, f$, where f is an atomic fluent. Literals that do not mention **not** are said to be **not**-free.

Note that in the ontologies one can have both **neg**- and $\neg$-literals, while $\mathcal{TR}^{PAD}$ uses **neg**- and **not**-literals instead. This is because logic programming rules cannot use classical negation, while ontologies do not use default negation.

Like the original Transaction Logic, $\mathcal{TR}^{PAD}$ contains logical connectives from the standard FOL $(\wedge, \vee, \forall, \exists,)$ plus two additional logical connectives: the ***serial conjunction***, $\otimes$, and the modal operator $\Diamond$ for *hypothetical* execution.

[4] Recall that the conflict resolution set contains all the rules that can be fired on a given working memory.

Informally, a serial conjunction of the form $\phi \otimes \psi$ represents 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 state changes take place. For instance, the first part of the following formula

$$\Diamond(\mathit{insert}(\mathsf{infection}) \otimes \mathit{bill\_insurance} \otimes \mathsf{has\_paid}) \otimes \mathit{insert}(\mathsf{takesT})$$

is a hypothetical test to verify that the patient's insurance company will pay in case of an infection after the blood test. The actual blood test is only performed if the hypothetical test succeeds. In particular, we will use hypothetical executions to check—before firing a rule—that executing the action associated with such a rule will not produce an inconsistent state. In this paper we will assume that hypothetical formulas contain only serial conjunctions of literals.

$\mathcal{TR}^{PAD}$ 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 is a **not**-free literal, $c_1, \ldots, c_n$ are *literals* (fluents or actions), f is a **not**-free fluent literal, b_1, b_2 are conjunctions of *fluent* literals or hypotheticals (**not**-literals are ok), b_3, b_4 are conjunctions of **not**-free *fluent* literals, $\mathbf{d}_0, \mathbf{d}_1 \ldots$ are identifiers, and a is an action atom. These actions will be used to encode the *assert* and *retract* actions in production rules, as well as the laws of inertia (a.k.a. frame axioms). Observe that in this paper we address the issue of how to express frame axioms, not the larger issue of the frame problem, which aims to encode one general principle of inertia, rather than developing particular frame axioms that are suitable in specific applications.

Rules		**Premises**	
(i) $c \leftarrow c_1 \otimes \cdots \otimes c_n$	(a serial-Horn rule)	(iii) $\mathbf{d}_0 \rhd f$	(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 the literal c, which can be viewed as a calling sequence for a complex transaction and $c_1 \otimes \cdots \otimes c_n$ can be viewed as 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. In that case we call c a ***defined fluent*** and the rule itself 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 $\mathit{r\_1} \leftarrow \mathsf{requested}(D,P,T) \otimes \mathsf{dnaT}(T) \otimes \mathit{insert}(\mathsf{takesT}(P,T))$ defines a compound action *r_1*. This action behaves in the same way as the rule r_1 in our running example. 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. For instance, the PAD $(\mathsf{healthy}(P) \otimes \mathit{insert}(\mathsf{dnaT}(T))) \rightarrow (\mathit{insert}(\mathsf{dnaT}(T)) \otimes \mathsf{healthy}(P))$. states that if a patient is healthy, she remains so after adding a DNA type in the database. This is a simplified version of an inertial law in $\mathcal{TR}^{PAD}$. 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 *r_1* above) and, in this way, $\mathcal{TR}^{PAD}$ can create larger action theories by composing 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, to represent the initial database in our example, we can use the state premises

$$\mathbf{d}_0 \rhd \mathsf{dnaT}(\mathrm{pcr}) \qquad \mathbf{d}_0 \rhd \mathsf{requested}(\mathrm{Smith}, \mathrm{Laura}, \mathrm{pcr}) \qquad \mathbf{d}_0 \rhd \mathsf{flu}(\mathrm{Laura})$$

The run-premise $\mathbf{d}_0 \overset{\mathit{insert}(\mathsf{takeT}(\boldsymbol{t}))}{\rightsquigarrow} \mathbf{d}_1$ says that executing the *pda* $\mathit{insert}(\mathsf{takeT}(\boldsymbol{t}))$ in the state associated with $\mathbf{d}_0$ leads to the state represented by $\mathbf{d}_1$.

A ***transaction*** is a statement of the form $\mathtt{?-}_{\mathbf{d}_0} \exists \bar{X} \phi$, 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 $\mathcal{TR}$ generalize the notion of queries in ordinary logic programming. For instance, $\mathtt{?-}_{\mathbf{d}_0} \mathsf{flu}(\mathrm{Laura}) \otimes \mathit{r\_1}$ is a transaction that first checks if the patient has a flu in the initial state $\mathbf{d}_0$; if so, the compound action *r_1* is executed. If the execution of the transaction cannot proceed the already executed actions are undone and the underlying database state remains unchanged. This property is known as *atomicity* of transactions in databases. A $\mathcal{TR}^{PAD}$ ***transaction base*** is a set of serial-Horn rules. A $\mathcal{TR}^{PAD}$ ***action base*** is a set of PADs. A $\mathcal{TR}^{PAD}$ **specification** is a tuple $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ where $\mathcal{E}$ is a $\mathcal{TR}^{PAD}$ action base, $\mathbf{P}$ is a $\mathcal{TR}^{PAD}$ transaction base, and $\mathcal{S}$ is a set of premises.

Semantics. Now we define a well-founded semantics [29] for $\mathcal{TR}^{PAD}$, which, to the best of our knowledge, has never been done before. This semantics uses three truth values, $\mathbf{u}$, $\mathbf{t}$ and $\mathbf{f}$, which stand for *true*, *false*, and *undefined* and are ordered as follows: $\mathbf{f} < \mathbf{u} < \mathbf{t}$. In addition, we will use the following operator $\sim$: $\sim \mathbf{t} = \mathbf{f}$, $\sim \mathbf{f} = \mathbf{t}$, $\sim \mathbf{u} = \mathbf{u}$. A **database state D** (or just a *state,* for short) is a set of ***ground*** (i.e., variable-free) fluent literals. In the language, states are referred to via state identifier constants, which were introduced earlier. The mapping between state identifiers and states is determined by *path structures*, defined next.

Definition 9 (Three-valued Partial Herbrand Interpretation). *A **partial Herbrand interpretation** is a mapping $\mathcal{H} : \mathcal{B} \mapsto \{\boldsymbol{f}, \boldsymbol{u}, \boldsymbol{t}\}$ that assigns a truth value, $\boldsymbol{f}, \boldsymbol{u}$, or $\boldsymbol{t}$, to every formula ϕ in $\mathcal{B}$.* □

A central feature in $\mathcal{TR}$ is the notion of *execution paths*, since $\mathcal{TR}$ formulas are evaluated over paths and *not* over states like in temporal logics [15].

Definition 10 (Three-valued Herbrand Path Structure). *A **Herbrand path structure** is a mapping $\mathbf{I}$ that assigns a partial Herbrand interpretation to every path. That is, for any path π, $\mathbf{I}(\pi)$ is an interpretation. So, for instance, $\mathbf{I}(\pi)(f)$ is a truth value for any literal f. This mapping must satisfy the restriction that for each ground base fluent f and database state $\mathbf{D}$:*

$$\mathbf{I}(\langle \mathbf{D} \rangle)(f) = \boldsymbol{t} \;\; \textit{if } f \in \mathbf{D} \qquad \textit{and} \qquad \mathbf{I}(\langle \mathbf{D} \rangle)(\mathbf{neg}\, f) = \boldsymbol{t} \;\; \textit{if } \mathbf{neg}\, f \in \mathbf{D}$$

where $\langle \mathbf{D} \rangle$ *is a path that contains only one state,* $\mathbf{D}$*.*
In addition, $\mathbf{I}$ *includes a mapping of the form:*

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

which associates states (i.e., sets of atomic formulas) to state identifiers. We will usually omit the subscript. □

Intuitively, Herbrand path structures in $\mathcal{TR}$ play a role similar to transition functions in temporal logics by providing a link between states and actions.

An ***execution path*** of length k, or a ***k-path***, is a finite sequence of states, $\pi = \langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $k \geq 1$. A ***path abstraction*** is a finite sequence of state identifiers. If $\langle \mathbf{d}_1 \ \ldots \ \mathbf{d}_k \rangle$ is a path abstraction then $\langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $\mathbf{D}_i = \Delta(\mathbf{d}_i)$, is an execution path. We will also sometimes write $\mathcal{M}(\langle \mathbf{d}_1 \ \ldots \ \mathbf{d}_k \rangle)$ meaning $\mathcal{M}(\langle \Delta(\mathbf{d}_1) \ \ldots \ \Delta(\mathbf{d}_k) \rangle)$. A ***split*** of a path π is a pair of subpaths, π_1 and π_2, such that $\pi_1 = \langle \mathbf{D}_1 \ldots \mathbf{D}_i \rangle$ and $\pi_2 = \langle \mathbf{D}_i \ldots \mathbf{D}_k \rangle$ for some i $(1 \leq i \leq k)$. In this case, we write $\pi = \pi_1 \circ \pi_2$.

In the remainder of this section we will consider ground rules and PADs only. We can make this assumption without loosing generality because all the variables in a rule are considered to be universally quantified.

The following definition formalizes the idea that truth of $\mathcal{TR}$ formulas is defined on paths.

Definition 11 (Satisfaction). *Let* $\mathbf{I}$ *be a Herbrand path structure,* π *be a path,* f *a ground* **not** *-free literal, and* G, G_1, G_2 *ground serial goals. We define* ***truth valuations*** *with respect to the path structure* $\mathbf{I}$ *as follows:*

- $\mathbf{I}(\pi)(f)$ *was already defined as part of the definition of Herbrand path structures.*[5]
- $\mathbf{I}(\pi)(\phi \otimes \psi) = max\{min(\mathbf{I}(\pi_1)(\phi), \mathbf{I}(\pi_2)(\psi)) \mid \pi = \pi_1 \circ \pi_2\}$
- $\mathbf{I}(\pi)(G_1 \wedge G_2) = min(\mathbf{I}(\pi)(G_1), \mathbf{I}(\pi)(G_2))$
- $\mathbf{I}(\pi)(\mathbf{not}\ \phi) = \sim \mathbf{I}(\pi)(\phi)$[6]
- $\mathbf{I}(\pi)(\diamond\phi) = \begin{cases} max\{\mathbf{I}(\pi')(\phi) \mid \pi' \textit{ is a path that starts at } \mathbf{D}\} & \textit{if } \pi = \langle \mathbf{D} \rangle \\ \boldsymbol{f} & \textit{otherwise} \end{cases}$
- $\mathbf{I}(\pi)(f \leftarrow G) = \boldsymbol{t}$ *iff* $\mathbf{I}(\pi)(f) \geq \mathbf{I}(\pi)(G)$
- $\mathbf{I}(\pi)(b_1 \otimes \alpha \otimes b_2 \rightarrow b_3 \otimes \alpha \otimes b_4) = \boldsymbol{t}$ *iff* π *has the form* $\langle \mathbf{D}_1, \mathbf{D}_2 \rangle$, $\mathbf{I}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle)(\alpha) = \boldsymbol{t}$*, and the following holds:*

$$\begin{aligned} &min\{min\{\mathbf{I}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b_1\}, min\{\mathbf{I}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b_2\}\} \\ \leq\ &min\{min\{\mathbf{I}(\langle \mathbf{D}_1 \rangle)(f) \mid f \in b_3\}, min\{\mathbf{I}(\langle \mathbf{D}_2 \rangle)(f) \mid f \in b_4\}\} \end{aligned}$$

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

In addition, we assume that the language includes the distinguished propositional constants $\mathbf{t}^\pi$, and $\mathbf{u}^\pi$ for each $\mathcal{TR}$ path π. Observe that since there is an infinite number of paths, there is an infinite number of such constants. Informally, $\mathbf{t}^\pi$ ($\mathbf{u}^\pi$) is a proposition that has the truth value $\mathbf{t}$(respectively $\mathbf{u}$) only on the path π, and it is false on every other path.

[5] Here `max` is taken over a finite set of truth values $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$.
[6] Recall that $\sim \mathbf{t} = \mathbf{f}$, $\sim \mathbf{f} = \mathbf{t}$, $\sim \mathbf{u} = \mathbf{u}$.

Definition 12 (Model). *A path structure,* $\mathbf{I}$*, is a* ***model of a formula*** ϕ *if* $\mathbf{I}, \pi \models \phi$ *for every path* π*. In this case we write* $\mathbf{I} \models \phi$*. A path structure,* $\mathbf{I}$*, is a model of a set of formulas if it is a model of every formula in the set. A path structure,* $\mathbf{I}$*, is a* ***model of a premise-statement*** σ *iff:*

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

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

Example 4. Consider our running example. Let us present the specification $\Lambda = (\mathcal{E}, \mathbf{P}, \mathcal{S})$, that intuitively encodes the ontology and part of the PS. A complete encoding will be described in Section 6. Assume that $\mathcal{S}$ contains the premises already introduced in the previous section. The transaction base $\mathbf{P}$ contains the following rules encoding the ontology

$$\mathbf{neg}\ \mathsf{virusT}(T) \leftarrow \mathsf{dnaT}(T) \qquad \mathsf{healthy}(P) \leftarrow \mathsf{takesT}(P,T), \mathbf{neg}\ \mathsf{virusT}(T)$$
$$\mathbf{neg}\ \mathsf{healthy}(P) \leftarrow \mathsf{flu}(P)$$

The action base $\mathcal{E}$ contains two PADs encoding the (simplified) inertia laws, and the definition of the action $\mathit{insert}_{\mathsf{takeT}}$.

$$\mathsf{dnaT}(P) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{not}\ \mathsf{inconsistent} \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{dnaT}(P)$$
$$\mathsf{flu}(P) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{not}\ \mathsf{inconsistent} \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{flu}(P)$$
$$\mathsf{rcv\_meds}(P) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{not}\ \mathsf{inconsistent} \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{rcv\_meds}(P)$$
$$\mathit{insert}_{\mathsf{takeT}}(P,T) \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{takeT}(P,T)$$

From the premises and the rules in Λ, we can see that any path structure $\mathcal{I}$ that models Λ satisfies

$$\mathcal{I}(\mathbf{d}_0)(\mathsf{dnaT}(\mathrm{pcr})) = \mathbf{t} \qquad \mathcal{I}(\mathbf{d}_0\mathbf{d}_1)(\mathit{insert}_{\mathsf{takeT}}(\mathrm{Laura}, \mathrm{pcr})) = \mathbf{t}$$
$$\mathcal{I}(\mathbf{d}_0)(\mathsf{flu}(\mathrm{Laura})) = \mathbf{t} \qquad \mathcal{I}(\mathbf{d}_0)(\mathsf{rcv\_meds}(\mathrm{Laura})) = \mathbf{t}$$

Now take an interpretation, $\mathcal{I}_1$, such that $\mathcal{I}_1(\mathbf{d}_1)(\mathsf{inconsistent}) = \mathbf{f}$. From the PADs in $\mathcal{E}$ instantiated with pcr, Laura, and Smith, we can conclude that:

$$\mathcal{I}_1(\mathbf{d}_1)(\mathsf{dnaT}(\mathrm{pcr})) = \mathbf{t} \qquad \mathcal{I}_1(\mathbf{d}_1)(\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr})) = \mathbf{t}$$
$$\mathcal{I}_1(\mathbf{d}_1)(\mathsf{flu}(\mathrm{Laura})) = \mathbf{t} \qquad \mathcal{I}(\mathbf{d}_1)(\mathsf{rcv\_meds}(\mathrm{Laura})) = \mathbf{t}$$

and from the rules in the ontology it follows that $\mathcal{I}_1(\mathbf{d}_1)(\mathsf{healthy}) = \mathbf{t}$ and $\mathcal{I}_1(\mathbf{d}_1)(\mathbf{neg}\ \mathsf{healthy}) = \mathbf{t}$. Thus, $\mathbf{d}_1$ is inconsistent in $\mathcal{I}_1$. □

In classical logic programming based on three-valued models, given two Herbrand *partial* interpretations $\mathbf{N}_1$ and $\mathbf{N}_2$, we say that $\mathbf{N}_1 \leq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$. In addition, we say that $\mathbf{N}_1 \preceq^c \mathbf{N}_2$ iff all **not**-free literals that are true in $\mathbf{N}_1$ are true in $\mathbf{N}_2$ and all **not**-literals that are true in $\mathbf{N}_2$ are true in $\mathbf{N}_1$.

Definition 13 (Order on Path Structures). *Let* $\mathbf{M}_1$ *and* $\mathbf{M}_2$ *be two Herbrand path structures, then:*

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

These two orderings are considerably different. The *truth ordering* minimize the *amount of truth*, by minimizing the atoms that are true and maximizing the atoms that are false on each path. In contrast, the *information ordering* minimizes the amount of information by minimizing both the atoms that are true and false in each path.

Example 5. Consider a path structure $\mathcal{I}_2$ for the specification Λ in Example 4 that coincides with $\mathcal{I}_1$ in the path $\langle \mathbf{d}_0 \rangle$ but differs in $\langle \mathbf{d}_1 \rangle$ as follows:

$$\mathcal{I}_2(\mathbf{d}_1)(\mathsf{dnaT}(\mathrm{pcr})) = \mathbf{u} \qquad \mathcal{I}_2(\mathbf{d}_1)(\mathsf{inconsistent}) = \mathbf{u}$$
$$\mathcal{I}_2(\mathbf{d}_1)(\mathsf{flu}(\mathrm{Laura})) = \mathbf{u} \qquad \mathcal{I}_2(\mathbf{d}_1)(\mathsf{takeT}(\mathrm{Laura}, \mathrm{pcr}) = \mathbf{t}$$

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

Definition 14 (Least Model). *A model* $\mathbf{M}$ *of a specification* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *is* ***minimal*** *with respect to* $\preceq$ *iff for any other model,* $\mathbf{N}$*, of* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$*, if* $\mathbf{N} \preceq \mathbf{M}$ *then* $\mathbf{N} = \mathbf{M}$*. The* ***least*** *model of* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$*, denoted LPM*$(\mathcal{E}, \mathbf{P}, \mathcal{S})$*, is a minimal model that is unique.* □

The definition of *quotient* is key to the notion of well-founded $\mathcal{TR}^{PAD}$ models. It is modeled after [16] with appropriate extensions for PADs.

By $\mathcal{TR}^{PAD}$**-quotient** of an specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ **modulo** a path structure $\mathbf{I}$ we mean a new specification, $\frac{(\mathcal{E},\mathbf{P},\mathcal{S})}{\mathbf{I}}$, which is obtained from $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ by first *(i)* replacing every literal of the form $\mathbf{not}\, b$ in $\mathbf{P} \cup \mathcal{E}$ with $\mathbf{t}^\pi$ or $\mathbf{u}^\pi$ for every path π such that $\mathbf{not}\, b$ is true (respectively undefined) in $\mathbf{I}(\pi)$, and then *(ii)* removing all the remaining rules and PADs that have a literal of the form $\mathbf{not}\, b$ in the body that is false in $\mathbf{I}(\pi)$.

Next, we give a constructive definition of *well-founded models* for $\mathcal{TR}^{PAD}$ specifications in terms of a consequence operator. As in the classical case, the consequence operator, Γ, for a $\mathcal{TR}^{PAD}$ specification is defined as:

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

Now, suppose $\mathbf{I}_\emptyset$ is the path structure that maps each path π to the empty Herbrand interpretation in which all atoms are undefined. The ordinal powers of the consequence operator Γ are then defined inductively as follows:

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

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

Lemma 1. *(see [24]) The operator Γ is monotonic with respect to the information order relation $\leq$ when $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ is fixed. That is,*

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

Definition 15 (Well-founded Model). *The **well-founded** model of a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, written $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$, is defined as a limit of the sequence $\{\Gamma^{\uparrow n}(\mathbf{I}_\emptyset)\}$.*
$WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is, indeed, a model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$, as shown in [24]. □

Example 6. Consider the specification in Example 4 together with the following rule defining the fluent inconsistent.

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

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

$$\mathsf{dnaT}(P) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{u}^\pi \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{dnaT}(P)$$
$$\mathsf{flu}(P) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{u}^\pi \rightarrow \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{flu}(P)$$

Since $\frac{\Lambda}{\mathbf{I}_\emptyset}$ is **not**-free, it has a minimal model [24] $\Gamma^{\uparrow 1}(\mathbf{I}_\emptyset) = \mathcal{I}_1$. It follows from the construction of $\mathcal{I}_1$ that $\mathcal{I}_1(\langle \mathbf{d}_1 \rangle)(\mathsf{inconsistent}) = \mathbf{u}$. It is not hard to see that in the WFM of Λ, inconsistent is also undefined in $\mathcal{I}_1(\langle \mathbf{d}_1 \rangle)$. This is because the frame axioms are preventing the inconsistency from occurring, but it is still detected. Without the rules encoding the ontology, inconsistent would be false in $WFM(\langle \mathbf{d}_1 \rangle)$. □

Theorem 1. *(see [24]) $WFM((\mathcal{E}, \mathbf{P}, \mathcal{S}))$ is the least model of $(\mathcal{E}, \mathbf{P}, \mathcal{S})$.*

6 Declarative Semantics

In this section we present the reduction of production systems augmented with Datalog-rewritable ontologies to $\mathcal{TR}^{PAD}$. Given an alphabet $\mathcal{L}_{\mathsf{PS}}$ for a production system PS, the corresponding language $\mathcal{L}_{\mathcal{TR}}$ of the target $\mathcal{TR}^{PAD}$ formulation will consist of symbols for rule labels, constants, and predicates. In addition, $\mathcal{L}_{\mathcal{TR}}$ has the following symbols: *(i)* the *pdas* *add_used* and *clean_used*, and for every predicate $p \in \mathcal{L}_{\mathsf{PS}}$, *Ins_p*, and *del_p*; *(ii)* the compound action *act*; *(iii)* the defined fluent inconsistent, and for every rule label r the defined fluent fireable_r; *(iv)* the fluents inertial and used. Intuitively, the *pdas* *Ins_p*, and *del_p* above represent the actions ***assert*** and ***retract***. The *pdas* *add_used* and *clean_used*, and the fluent used, are used to keep track of the assignments that has already been used to instantiate a FOR-production rule. The compound action *act* represents a generic production rule. The defined fluent fireable_r is true if the condition of the rule r holds and the action produces no inconsistencies. The defined fluent inconsistent is true, if there is an inconsistency in the state. The fluent inertial is used to distinguish inertial from non-inertial fluents.

Let $\psi = \alpha_1 \ldots \alpha_n$ be a sequence of atomic actions and $\phi = f_1 \wedge \cdots \wedge f_n \wedge l_1 \ldots l_m$ be a conjunction of atoms (f_i) and negative literals (l_j). We use $\widehat{\psi}$ and $\widehat{\phi}$ to denote

the $\mathcal{TR}$ -serial conjunctions $\widehat{\psi} = \widehat{\alpha_1} \otimes \cdots \otimes \widehat{\alpha_n}$ and $\widehat{\phi} = f_1 \wedge \cdots \wedge f_m \wedge \sim l_1 \wedge \cdots \wedge \sim l_m$, where

$$\widehat{\alpha_i} = \begin{cases} \mathit{Ins\_p}(\boldsymbol{t}) & \text{if } \alpha_i = \boldsymbol{assert}(p(\boldsymbol{t})) \\ \mathit{del\_p}(\boldsymbol{t}) & \text{if } \alpha_i = \boldsymbol{retract}(p(\boldsymbol{t})) \end{cases}$$

$$\sim l_j = \begin{cases} \mathbf{not}\, f(\boldsymbol{c}) & \text{if } l_j = \neg f(\boldsymbol{c}) \wedge f \in \mathcal{P}_{PS} \\ \mathbf{neg}\, f(\boldsymbol{c}) & \text{if } l_j = \mathbf{neg}\, f(\boldsymbol{c}) \wedge f \in \mathcal{P}_{DL} \end{cases}$$

In the following, let $\mathsf{PS} = (\mathcal{T}, \mathsf{L}, R)$ be a production system. For simplicity we assume that conditions in production rules are conjunctions of fluent literals. In addition, we assume we have an ***initial working memory***, WM_0, that represents the knowledge we have about the initial state of the system.

The reduction, Λ_{PS}, of a $\mathsf{PS}, \mathsf{WM}_0$ to $\mathcal{TR}^{PAD}$ is a $\mathcal{TR}^{PAD}$ specification $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ composed of the following PADs ($\mathcal{E}$), rules for defined fluents ($\mathbf{P}$) and premises ($\mathcal{S}$). From now on we assume that the ontology $\mathcal{T}$ is *Datalog-rewritable*.

1. **Ontology $\mathcal{T}$: $\mathbf{P}$** contains all the rules from the Datalog rendering of $\mathcal{T}$.
2. **Initial Database:** The premises below characterize the content of the initial working memory WM_0. For each atomic literal $p(t_1, \ldots, t_n)$ in WM_0

$$\mathbf{d}_0 \rhd p(t_1, \ldots, t_n) \in \mathcal{S} \qquad \mathbf{d}_0 \rhd \mathsf{inertial}(p(t_1, \ldots, t_n)) \in \mathcal{S}$$

   We could have written this as $\mathsf{inertial}(p, t_1, \ldots, t_n)$ to avoid the appearance of being second order or that the use of function symbols here is essential.
3. **Frame Axioms:** The following frame axioms encode the laws of inertia. They also take care of the actual "removal" of $\mathcal{L}_{\mathsf{PS}}$ atoms from the working memory, and the cleaning of the used assignments. Let p be a fluent predicate and α an action such that either $\alpha = \mathit{add\_used}$, or $\alpha = \mathit{clean\_used}$ or α involves assertion or retraction of an atom with predicate symbol q, where $p \neq q$. For any such pair (p, α), $\mathcal{E}$ has the following PAD except when $\alpha = \mathit{clean\_used}$ and $p = \mathsf{used}$:

$$(\mathsf{inertial}(p(\boldsymbol{X})) \wedge p(\boldsymbol{X})) \otimes \alpha(\boldsymbol{Y}) \otimes \mathbf{not}\, \mathsf{inconsistent} \rightarrow \alpha(\boldsymbol{Y}) \otimes (p(\boldsymbol{X}) \wedge \mathsf{inertial}(p(\boldsymbol{X})))$$

   Therefore, the application of the action *clean_used* leads to a state that does not contain any atom with predicate used. If α inserts an atom with predicate symbol q, the frame axiom needs an additional condition of the form $\boldsymbol{X} \neq \boldsymbol{Y}$ on the left side of the above PAD.
4. **Actions:** The following rules encode ***assert*** and ***retract*** in $\mathcal{TR}^{PAD}$:
   - *Insert:* For each predicate $p \in \mathcal{L}_{\mathsf{PS}}$ (whether in $\mathcal{P}_{DL}$ or $\mathcal{P}_{PS}$):

$$\left\{ \begin{array}{l} \mathit{Ins\_p}(t_1, \ldots, t_n) \rightarrow \mathit{Ins\_p}(t_1, \ldots, t_n) \otimes \\ \qquad\qquad (p(t_1, \ldots, t_n) \wedge \mathsf{inertial}(p(t_1, \ldots, t_n))) \end{array} \right\} \in \mathcal{E}$$

   - *Retract:* For each predicate $p \in \mathcal{P}_{DL}$,

$$\left\{ \begin{array}{l} \mathit{del\_p}(t_1, \ldots, t_n) \rightarrow \mathit{del\_p}(t_1, \ldots, t_n) \otimes \\ \qquad (\mathbf{neg}\, p(t_1, \ldots, t_n) \wedge \mathsf{inertial}(\mathbf{neg}\, p(t_1, \ldots, t_n))) \end{array} \right\} \in \mathcal{E}$$

Recall that the effect of the *pda* $\mathit{del\_p}$ for PS atoms is given by the interaction with the frame axioms. For instance, if applying $\mathit{del}_{dnaT}(pcr)$ in $\mathbf{d}_1$ results in a state $\mathbf{d}_2$, it holds that $\mathbf{d}_2$ is equal to $\mathbf{d}_1$ except for $\mathsf{dnaT}(pcr)$, which is not carried to $\mathbf{d}_2$ by the frame axioms. This is equivalent to remove $\mathsf{dnaT}(pcr)$ from $\mathbf{d}_2$.

5. **Production rules:** The following rules encode the production rules.
   - For each `IF-THEN`-rule of the form "r: `Forall` $\boldsymbol{x}$: `if` $\phi_r(\boldsymbol{x})$ `then` $\psi_r(\boldsymbol{x})$"

$$r \leftarrow \mathsf{fireable\_r}(\boldsymbol{X}) \otimes \widehat{\psi}_r(\boldsymbol{X}) \in \mathbf{P}$$

   - For each `FOR`-rule of the form "r: `For` $\boldsymbol{x}$: $\phi_r(\boldsymbol{x})$ `do` $\psi_r(\boldsymbol{x})$"

$$\left.\begin{array}{l} r \leftarrow \mathsf{fireable\_r}(\boldsymbol{X}) \otimes \widehat{\psi}_i(\boldsymbol{X}) \otimes \mathit{add\_used}(\boldsymbol{X}) \otimes \mathit{loop\_r} \\ \mathit{loop\_r} \leftarrow r \\ \mathit{loop\_r} \leftarrow (\mathbf{not}\ \exists \boldsymbol{X} : \mathsf{fireable\_r}(\boldsymbol{X})) \otimes \mathit{clean\_used} \end{array}\right\} \in \mathbf{P}$$

   where $\mathbf{not}\ \exists \boldsymbol{X} : \mathsf{fireable\_r}(\boldsymbol{X})$ above is a shorthand for $\mathbf{not}\ \mathsf{p}'$ such that p' is a new predicate defined as $\mathsf{p}' \leftarrow \mathsf{fireable\_r}(\boldsymbol{X})$.

6. **Auxiliary Actions and Premises**
   - *Run-Premises*: Then for each pda α and a sequence ξ of actions Ins, del, $\mathit{add\_used}$, or $\mathit{clean\_used}$, the set of premises $\mathcal{S}$ contains the following run-premise:

$$\mathbf{d}_\xi \overset{a}{\rightsquigarrow} \mathbf{d}_{\xi,a}$$

     For example, $\mathbf{d}_{0,\mathit{Ins\_p}(c)} \overset{\mathit{Ins}_q(d)}{\rightsquigarrow} \mathbf{d}_{0,\mathit{Ins\_p}(c),\mathit{Ins}_q(d)}$.
   - *Inconsistency*: For each predicate $p \in \mathcal{L}_{\mathsf{PS}}$, $\mathbf{P}$ contains a rule of the form:

$$\mathsf{inconsistent} \leftarrow p(\boldsymbol{X}),\ \mathbf{neg}\, p(\boldsymbol{X})$$

   - *Adding* used *assignments:*

$$\left\{ \mathit{add\_used}(\boldsymbol{Y}) \rightarrow \mathit{add\_used}(\boldsymbol{Y}) \otimes \mathsf{used}(\boldsymbol{X}) \wedge \mathit{inertial}(\mathsf{used}(\boldsymbol{X})) \right\} \in \mathcal{E}$$

   - *Fireability.* The following rules are in $\mathbf{P}$:

$$\left.\begin{array}{l} \text{If } r \text{ is an} \\ \texttt{IF-THEN} \text{ rule} \end{array}\right\} \left\{ \begin{array}{l} \mathsf{fireable\_r}(\boldsymbol{X}) \leftarrow \widehat{\phi}_r(\boldsymbol{X}) \wedge \\ \qquad (\Diamond \widehat{\psi}_r(\boldsymbol{X}) \otimes \mathbf{not}\ \mathsf{inconsistent}) \end{array}\right.$$

$$\left.\begin{array}{l} \text{If } r \text{ is a} \\ \texttt{FOR} \text{ rule} \end{array}\right\} \left\{ \begin{array}{l} \mathsf{fireable\_r}(\boldsymbol{X}) \leftarrow \widehat{\phi}_r(\boldsymbol{X}) \wedge \mathbf{not}\ \mathsf{used}(\boldsymbol{X}) \wedge \\ \qquad (\Diamond \widehat{\psi}_r(\boldsymbol{X}) \otimes \mathbf{not}\ \mathsf{inconsistent}) \end{array}\right.$$

   - *Random choice of action*: For each rule label $r_i \in L$

$$\mathit{act} \leftarrow r_i \ \in \mathbf{P}$$

To run k rules of the production system we use the transaction:[7]

$$?- \underbrace{\mathit{act} \otimes \cdots \otimes \mathit{act}}_{k}$$

[7] Here we could also use recursion to represent runs of arbitrary length.

Theorem 2 (Soundness [24]). *Let* $(\mathcal{E}, \mathbf{P}, \mathcal{S})$ *be the* $\mathcal{TR}^{PAD}$ *embedding of a PS configuration. Suppose*

$$\mathcal{E}, \mathbf{P}, \mathcal{S}, \boldsymbol{d}_0 \ldots \boldsymbol{d}_k \models \underbrace{\mathit{act} \otimes \cdots \otimes \mathit{act}}_{m}$$

Then there are working memories $\mathsf{WM}_1 \ldots \mathsf{WM}_m$*, and rules* $r_1 \ldots r_m$ *such that*

$$\begin{array}{c} \mathsf{WM}_0 \overset{r_1}{\hookrightarrow} \mathsf{WM}_1 \\ \vdots \\ \mathsf{WM}_{m-1} \overset{r_m}{\hookrightarrow} \mathsf{WM}_m \end{array}$$

Example 7. In the previous examples, we worked with a simplified version of the frame axioms. Let us now show the complete definition of one of them:

$$\begin{aligned} \mathit{inertial}(\mathsf{dnaT}(T)) \wedge \mathsf{dnaT}(T) \otimes \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathbf{not}\ \mathsf{inconsistent} \rightarrow \\ \mathit{insert}_{\mathsf{takeT}}(P,T) \otimes \mathsf{dnaT}(T) \wedge \mathit{inertial}(\mathsf{dnaT}(T)) \end{aligned}$$

Now we are ready to define the `FOR` rule r_2 and the defined fluent $\mathsf{fireable\_}r_2$

$$\begin{aligned} r_2 &\leftarrow \mathsf{fireable\_}r_2(P,T) \otimes \mathit{del}_{\mathbf{neg}\,\mathsf{healthy}}(P) \otimes \mathit{add\_used}(P,T) \otimes \mathit{loop\_r} \\ \mathit{loop\_r} &\leftarrow r_2 \\ \mathit{loop\_r} &\leftarrow (\mathbf{not}\ \exists P,T\colon \mathsf{fireable\_}r_2(P,T)) \otimes \mathit{clean\_used} \end{aligned}$$

$$\begin{aligned} \mathsf{fireable\_}r_2(P,T) \leftarrow \mathsf{takesT}(P,T) \wedge \mathsf{dnaT}(T) \wedge \mathbf{not}\ \mathsf{used}(\boldsymbol{X})) \wedge \\ (\Diamond \mathit{del}_{\mathbf{neg}\,\mathsf{healthy}}(P) \otimes \mathbf{not}\ \mathsf{inconsistent}) \end{aligned}$$

Now we come back to our query, and check: $\texttt{?-}_{\mathbf{d}_0}\mathsf{flu}(\mathrm{Laura}) \otimes \mathit{r\_1}$.

In this case we would detect an inconsistency produced by *r_1*. By observing the construction of the well founded model, we can track back the conflict. □

Recall that beside detecting inconsistencies, $\mathcal{TR}^{PAD}$ allow to execute inconsistency free rules and obtain runs as in the original production system.

7 Conclusions

In this paper we proposed a new semantics for the combination of production systems with *arbitrary DL* ontologies. Unlike previous approaches [26,13,10,4,21,31], the semantics presented here supports extensions, like the *FOR*-loops or *while*-loops, that are not included in RIF-PRD, but are found in commercial production systems such as IBM's *JRules*[19]. In addition, our approach can handle inconsistencies produced by the interaction of rule actions and the ontology.

We also defined a sound embedding of such semantics, restricted to rule-based *DL* ontologies, into Transaction Logic with partial action definitions ($\mathcal{TR}^{PAD}$). This reduction gives a declarative semantics to the combination, and is considerably simpler and compact that other approaches, including [26,21,31,10,20].

To model production systems in $\mathcal{TR}^{PAD}$, we extended $\mathcal{TR}^{PAD}$ with default negation and defined the well-founded semantics [29] for it. It is worth noting

that this $\mathcal{TR}^{PAD}$ embedding can be used as an implementation vehicle for the combination of PS and rule-based ontologies.

Acknowledgments. We thank the anonymous reviewers for useful feedback. M. Rezk and M. Kifer were partially supported by the European Commission under the project OntoRule. M. Kifer was also partially supported by the NSF grant 0964196.

References

1. Anicic, D., Fodor, P., Stühmer, R., Stojanovic, N.: An approach for data-driven logic-based complex event processing. In: The 3rd ACM International Conference on Distributed Event-Based Systems, DEBS (2009)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The dl-lite family and relations. Journal of Artificial Intelligence Research (JAIR) 36, 1–69 (2009)
3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press (2003)
4. Baral, C., Lobo, J.: Characterizing production systems using logic programming and situation calculus (1995), `http://www.public.asu.edu/~cbaral/papers/char-prod-systems.ps`
5. Bonner, A., Kifer, M.: Transaction logic programming. In: Proc. of International Conference on Logic Programming (ICLP), Budapest, Hungary, pp. 257–282. MIT Press (June 1993)
6. Bonner, A., 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`
7. 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 (March 1998)
8. Bonner, A.J., Kifer, M.: Applications of Transaction Logic to Knowledge Representation. In: Gabbay, D.M., Ohlbach, H.J. (eds.) ICTL 1994. LNCS, vol. 827, pp. 67–81. Springer, Heidelberg (1994)
9. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of $DL-Lite$ Knowledge Bases. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 112–128. Springer, Heidelberg (2010)
10. Damásio, C.V., Alferes, J.J., Leite, J.: Declarative Semantics for the Rule Interchange Format Production Rule Dialect. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 798–813. Springer, Heidelberg (2010)
11. Davulcu, H., Kifer, M., Ramakrishnan, C.R., Ramakrishnan, I.V.: Logic based modeling and analysis of workflows. In: PODS, pp. 25–33 (1998)
12. Davulcu, H., Kifer, M., Ramakrishnan, I.V.: Ctr-s: a logic for specifying contracts in semantic web services. In: WWW, pp. 144–153 (2004)
13. de Bruijn, J., Rezk, M.: A Logic Based Approach to the Static Analysis of Production Systems. In: Polleres, A., Swift, T. (eds.) RR 2009. LNCS, vol. 5837, pp. 254–268. Springer, Heidelberg (2009)

14. de Sainte Marie, C., Hallmark, G., Paschke, A.: Rif production rule dialect (2010), http://www.w3.org/TR/rif-prd/
15. Emerson, E.A.: Temporal and modal logic. In: Handbook of Theoretical Computer Science, pp. 995–1072. Elsevier (1995)
16. Fodor, P., Kifer, M.: Transaction logic with defaults and argumentation theories. In: ICLP (Technical Communications), pp. 162–174 (2011)
17. Heymans, S., Eiter, T., Xiao, G.: Tractable reasoning with dl-programs over datalog-rewritable description logics. In: European Conference on Artificial Intelligence, pp. 35–40 (2010)
18. Horrocks, I.: Ontologies and the semantic web. Commun. ACM 51, 58–67 (2008)
19. I. JRules, http://www.ibm.com/software/integration/business-rule-management/jrules-family/
20. Kowalski, R., Sadri, F.: Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents. In: Polleres, A., Swift, T. (eds.) RR 2009. LNCS, vol. 5837, pp. 1–23. Springer, Heidelberg (2009)
21. Lausen, G., Ludäscher, B., May, W.: On Active Deductive Databases: The Statelog Approach. In: Kifer, M., Voronkov, A., Freitag, B., Decker, H. (eds.) Dagstuhl Seminar 1997, DYNAMICS 1997, and ILPS-WS 1997. LNCS, vol. 1472, pp. 69–106. Springer, Heidelberg (1998)
22. Pearce, D., Wagner, G.: Logic Programming with Strong Negation. In: Eriksson, L.-H., Hallnäs, L., Schroeder-Heister, P. (eds.) ELP 1991. LNCS, vol. 596, pp. 311–326. Springer, Heidelberg (1992)
23. Raschid, L.: A semantics for a class of stratified production system programs. J. Log. Program. 21(1), 31–57 (1994)
24. Rezk, M., Kifer, M.: Formalizing production systems with rule-based ontolgies (2011), http://www.inf.unibz.it/~mrezk/techreportTRPS.pdf
25. Rezk, M., Kifer, M.: Reasoning with Actions in Transaction Logic. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 201–216. Springer, Heidelberg (2011)
26. Rezk, M., Nutt, W.: Combining Production Systems and Ontologies. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 287–293. Springer, Heidelberg (2011)
27. Roman, D., Kifer, M.: Reasoning about the behavior of semantic web services with concurrent transaction logic. In: VLDB, pp. 627–638 (2007)
28. Roman, D., Kifer, M.: Semantic Web Service Choreography: Contracting and Enactment. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 550–566. Springer, Heidelberg (2008)
29. Van Gelder, A., Ross, K., Schlipf, J.: The well-founded semantics for general logic programs. Journal of the ACM 38(3), 620–650 (1991)
30. Winslett, M.: Updating logical databases. Cambridge University Press, New York (1990)
31. Zaniolo, C.: A unified semantics for active and deductive databases. In: Workshop on Rules In Database Systems (RIDS 1993), pp. 271–287. Springer, Heidelberg (1993)