Combining Production Systems and Ontologies

Martín Rezk and Werner Nutt

KRDB Research Centre, Faculty of Computer Science
Free University of Bozen-Bolzano, Italy
{rezk,nutt}@inf.unibz.it

Abstract. Production systems are an established paradigm in knowledge representation, while ontologies are widely used to model and reason about the domain of an application. Description logics, underlying for instance the Web ontology language OWL, are a well-studied formalism to express ontologies. In this work we combine production systems (PS) and Description Logics (DL) in such a way that allows one to express both, facts and rules, using an ontology language.

We explore the space of design options for combining the traditional closed world semantics of PS with the open world semantics of DL and propose a generic semantics for such combination. We show how to encode our semantics in a fixpoint extension of first-order logic. We show that in special cases (monotonic and light PS) checking properties of the system such as termination is decidable.

1 Introduction

Production systems (PS) are one of the oldest knowledge representation paradigms in Artificial Intelligence, and are still widely used today.

We consider PSs that consist of *(i)* a set of rules r of the form

$$\textbf{if } \phi_r \textbf{ then } \psi_r \tag{1}$$

(ii) a set of ground facts, called *working memory*, which contains the current state of knowledge, and *(iii)* a rule interpreter, which executes the rules and makes changes in the working memory, based on the actions in the rules. The condition ϕ_r is a FOL formula, and the action $\psi_r = +a_1, \ldots, +a_k, {\sim}b_1, \ldots, {\sim}b_l$ where each $+a_i$ and ${\sim}b_j$ stand for asserting and retracting an atomic fact (atom) to/from the working memory. These rules syntactically correspond to the fragment of the RIF Production Rule Dialect[1] that does not include the *forall* construct, *modify* actions, external functions, etc. Semantically it deviates from RIF on the fact that we assume that all the actions are applied simultaneously. Given a working memory, the rule interpreter applies the rules in three steps: (1) *pattern matching*—typically using the RETE algorithm [5]—(2) *conflict resolution*–the interpreter chooses zero or one pair among the rules whose condition is satisfied according to its strategy—and (3) *rule execution*. The formal semantics for a PS can be found in [3].

PSs do not provide a way to express knowledge about the domain, and the relations among terms in the PS vocabulary. Moreover, they cannot handle incomplete information. Description Logic (DL) ontologies [1] are a standard way to achieve that. In this work we consider standard DLs without nominals. For concreteness, we will work with

[1] http://www.w3.org/TR/rif-prd/

$\mathcal{ALC}$. Observe that such combination is particularly relevant when different PSs, with different policies and vocabularies need to work as one. This can happen, for instance, when two or more companies fuse together.

The integration of two knowledge representation languages with such different semantics requires a solid theoretical foundation in order to understand the implications of the combination—both semantical and operational—on a deep level. In this paper we bridge the gap between the semantics of production rules and ontologies.

Example 1 ***(Running Example).*** A research institute has staff members ($\texttt{Staff}$), and visitors ($\texttt{Visitor}$). This institute has a system that enforces a set of regulations over the database (the working memory WM_0 shown below) through a set of rules:

1. If a user of the system ($\texttt{Usr}$) quits ($\texttt{Qt}$) his/her position in the institute, then he/she is removed from the database:

   r_1: **if** $\texttt{Usr}(x) \wedge \texttt{Qt}(x)$ **then** $\sim \texttt{Staff}(x)$

2. If a user is banned ($\texttt{Bnd}$) or is not allowed ($\texttt{Allowed}$) to use the system, then s/he should no longer be a user:

   r_2: **if** $\texttt{Staff}(x) \wedge (\texttt{Bnd}(x) \vee \neg\texttt{Allowed}(x))$ **then** $\sim \texttt{Usr}(x)$

Our institute is going to join a network of research institutes. To set up a common vocabulary, and to agree on the definitions of the shared terms and some basic regulation, they need to incorporate the following ontology ($\mathcal{ALC}$ TBox) into their system:
(a) $\texttt{Staff} \sqcup \texttt{Visitor} \equiv \texttt{Usr}$ (b) $\texttt{Staff} \sqcap \texttt{Visitor} \equiv \bot$ (c) $\texttt{Allowed} \sqsubseteq \texttt{Usr}$
Observe that the new ontology does not cover the whole vocabulary of the system, but only part of it. The portion of the working memory whose vocabulary is covered by the ontology is physically distributed over different institutions, the rest remains in our local database. Our initial working memory is $\mathsf{WM}_0 = \{\texttt{Staff}(\text{Kim}), \texttt{Qt}(\text{Kim})\}$. Before being able to run this system, we need to solve the following issues

– *How do we check if a rule condition holds in* WM_0*?* In traditional PS semantics, WM_0 is viewed as a unique model in which we can check the satisfaction of formulas. However, under $\mathcal{ALC}$ semantics, WM_0 would be seen as theory (ABox) that together with the TBox has a possibly infinite set of models, and thus entailment is needed.
– *How do we interpret the retraction of an atom as stated in rules 1 and 2?* In traditional PS semantics, to retract an atom is equivalent to changing the truth value of a fact from true to false since there is a unique model. It is a simple operation that is achieved by removing the fact from the working memory. In $\mathcal{ALC}$ (as in other DLs) to retract a fact that is entailed by the knowledge base or to enforce the knowledge base to entail a fact to be false, is a complex problem that cannot always be solved [6,4].
– *When do we execute a rule?* Traditionally, rules are fired only if they change the working memory. However, now two syntactically different working memories can be semantically equivalent, for instance, the working memory WM_0 in our running example and $\mathsf{WM}_0' = \{\texttt{Staff}(\text{Kim}), \texttt{Usr}(\text{Kim}), \texttt{Qt}(\text{Kim})\}$.

We solve the issues highlighted above by presenting a semantics, named *POS*, that takes an hybrid approach to checking satisfaction of rule conditions, and a traditional approach to rule application and execution.

Our contribution with this work is three-fold: *(i)* we discuss the design options of combining production systems and DL ontologies, together with the problems and advantages that arise from the different options; *(ii)* we define a syntax for PSs augmented with DL ontologies and an operational semantics, called POS; *(iii)* we embed POS into Fixed-Point Logic (FPL), giving in this way a model-theoretic semantics to the combination and study how, in some restricted cases, the static analysis of production rules, and the FPL embedding can be used to check properties (like termination).

2 Design Space

In this section we discuss the design options for the semantics through our running example. Due to the lack of space, we will not go over all the options, but just the most relevant ones. Details can be found in [7].

Rule Conditions: Traditional PSs evaluate rule conditions over the unique model represented by the working memory. In our running example, this would mean that Kim is not allowed to use the system, since $\texttt{Allowed}(\text{Kim})$ is not in WM_0. Thus, we could fire Rule 2 removing Kim from the users list. On the contrary, in DL semantics, the absence of knowledge does not imply any conclusion. In fact, in our example, this is desirable for the portion of the working memory that is distributed. We do not want that if a server goes down, part of our working memory becomes false and the engine starts firing rules indiscriminately based on that. However, for the portion of the working memory that is locally stored in our server, it is perfectly fine to assume that we have complete information about it, and take the traditional approach.

Rule Effects: Rule application in traditional PSs is straightforward. For our combination, we focus on how to interpret the *retraction* of facts, since we assume that in any case insertion amounts to adding facts. Consider Rule 2 in our running example instantiated with $x = \text{Kim}$, and suppose that the working memory resulting from applying that rule is WM. In the context of DLs we could expect that applying $\sim\texttt{Usr}(x)$ changes the truth value of $\texttt{Usr}(\text{Kim})$ from true to false. This means that the resulting working memory WM entails $\neg\texttt{Usr}(\text{Kim})$. This is a difficult problem and moreover, such updates are not always expressible in a standard DL like $\mathcal{ALC}$ [6,4]. A second option is to give $\sim$ the traditional PS meaning, that is, just remove the atom from the working memory. In our view, the latter approach has four important advantages: *(i)* the result of applying an action is always expressible, independently of the language of the ontology, *(ii)* we do not need to deal with inconsistencies, *(iii)* the semantics remains the same as in traditional PS semantics, and *(iv)* to compute the resulting working memory is almost trivial. We opt for this last option. Note that removal produces a syntactic change, but not always a semantic change. For instance, applying $\sim\texttt{Usr}(\text{Kim})$ to WM_0 in our running example, does not change the truth value of $\texttt{Usr}(\text{Kim})$. That is because removal cannot retract *consequences* of a knowledge base. It is worth noticing that this interpretation of negation is compliant with the current version of DELETE in SPARQL 1.1.[2]

Executability: Traditionally, rules are fired only if they change the working memory. However, now two syntactically different working memories can be semantically equivalent, for instance, the working memory WM_0 in our running example and WM'_0 introduced above. Therefore, we have two options: *(i)* keep the traditional semantics, and fire a rule if it syntactically changes the working memory, or *(ii)* fire the rule only if it changes the semantics of the working memory. If the ontology does not cover the vocabulary of the PS, and therefore has no complete information about the relation between the concepts and rules in the PS (like in our running example), then semantically equivalent working memories might not represent the same information. Consider the working memories WM_0 and WM'_0 above. If we remove $\mathtt{Staff}(\mathrm{Kim})$ from WM_0, Kim is no longer a user. On the contrary, Kim is still a user after removing $\mathtt{Staff}(\mathrm{Kim})$ from WM'_0. This shows that having $\mathtt{Usr}(\mathrm{Kim})$ in the working memory should be interpreted as $\mathtt{Usr}(\mathrm{Kim})$ is "independent" (with respect to removal actions) from the taxonomical information in the ontology. This independency is desirable if there is some relation (neither stated by the ontology nor shared by all the institutions) between a concept in the ontology and some local concept. This last observation led us to choose to keep the traditional semantics. Section 3 will provide further details.

3 Production Systems and Ontologies Semantics

In this section we introduce **POS** (**P**roduction system and **O**ntologies **S**emantics). The design decisions we took when defining POS can be summarized in three main concepts. **Consistency**: POS is consistent with both, the traditional PS semantics, and the DL semantics. We require consistency with the traditional semantics of PS to be able to build up the combination with ontologies over the existing PS technologies. The consistency with the DL semantics is required not only to be able to exploit the existing DL reasoners but because we assume that we may not have complete knowledge about the data, thus, if some fact is not known to be true should not be assumed to be false. **DL Independence**: rule execution and rule effect are independent of the ontology language. Although in this paper we are working with $\mathcal{ALC}$ for concreteness, we want the semantics to be as general as possible. Recall that, for instance, an update that is possible in $\mathcal{ALCO}$ might not be possible in $\mathcal{ALC}$ [6]. **Partial Coverage**: the vocabulary of the ontology need not fully cover the vocabulary of the PS.

A *Production System* is a tuple $\mathsf{PS} = (\Sigma, \mathcal{T}, L, R)$, where $\Sigma = (P, C)$ is a first-order signature where the set P of predicate symbols is split into a set of DL predicates P_{DL} and a set of PS predicates P_{PS} and where C is a countably infinite set of constant symbols; $\mathcal{T}$ is a $\mathcal{ALC}$ TBox whose predicates belong to P_{DL}; L is a set of rule labels; and R is a set of rules, which are statements of the form (1) as shown in the introduction. A ***concrete*** PS (CPS) is a pair $(\mathsf{PS}, \mathsf{WM}_0)$, where WM_0 is a working memory.

[2] `http://www.w3.org/TR/sparql11-update/`

Next, we define when a rule's condition is satisfied in a working memory given an ontology (i.e., a TBox) and a valuation. Let WM be a working memory. An interpretation $\mathcal{I}$ is a model of WM iff for every atom $p(\boldsymbol{c}) \in \mathsf{WM}$, we have that $\boldsymbol{c} \in p^{\mathcal{I}}$.

Definition 1 (Satisfaction). *A model $\mathcal{I}$ of a working memory* WM satisfies *an atom $p(\boldsymbol{x})$ with a valuation σ, relative to* WM, *denoted $\mathcal{I}, \sigma \models_{\mathsf{WM}} p(\boldsymbol{x})$, iff*

- *$p \in P_{PS}$, and $p(\sigma(\boldsymbol{x})) \in \mathsf{WM}$, i.e., we take* WM *as a* model *for P_{PS} atoms, or*
- *$p \in P_{DL}$, and $\mathcal{I} \models p(\sigma(\boldsymbol{x}))$, i.e., we take* WM *as a* theory *for P_{DL} atoms.*

For formulas $\phi = \neg\phi_1,\ \phi_1 \wedge \phi_2,\ \phi_1 \vee \phi_2,\ \exists x\colon \phi_1$, or $\forall x\colon \phi_1$ we define $\mathcal{I}, \sigma \models_{\mathsf{WM}} \phi$ recursively as usual. A formula ϕ **holds** *in a working memory* WM *with a valuation σ, relative to a theory $\mathcal{T}$, denoted $\mathsf{WM}, \sigma \models_{\mathcal{T}} \phi$, iff $\mathcal{I}, \sigma \models_{\mathsf{WM}} \phi$ for every model $\mathcal{I}$ of $\mathcal{T} \cup \mathsf{WM}$.* □

Observe that in the absence of PS predicates, this definition coincides with the definition of satisfaction in DL. Analogously, in the absence of DL predicates, the previous definition coincides with the definition of satisfaction in the traditional semantics as stated in [3]. A rule $r \in R$ of the form (1) is **fireable** in a working memory WM with a valuation σ iff ϕ_r holds in WM, the resulting working memory $\mathsf{WM}' = (\mathsf{WM} \setminus \{\sigma(b_1, \ldots, b_l)\}) \cup \{\sigma(a_1, \ldots, a_k)\}$ is consistent and distinct from WM. Observe that the definition of rule application coincides with the one in traditional PS, focussing only on the syntactic changes of the working memory. We say that there is an (r,σ)-***transition*** from WM to WM', denoted $\mathsf{WM} \overset{r(\sigma(\boldsymbol{x}))}{\rightsquigarrow} \mathsf{WM}'$, iff the rule r is fireable in WM with valuation σ, and WM' is the working memory resulting from firing r in WM as defined above. To formalize the *runs*, we use transition graphs. The definition is as usual: the nodes of the graph are working memories, and the edges are transitions. Since we apply all actions simultaneously, rule applications are represented by single edges. A ***run*** for a concrete production system $(\mathsf{PS}, \mathsf{WM}_0)$ is a maximal path in such graph.

4 Declarative Semantics and Decidability Results

In this section we give an intuition of an embedding, Φ_{POS}, of POS into FPL, which models the runs of a PS. We exploit the fix point operator in FPL to guarantee that there is not an infinite backward chain from any state to the initial state in the model. This property can not be expressed in FOL and without this we lose completeness of the axiomatization, since we get runs with a transfinite number of states. In order to model the sequence structure of the runs there is a set of *foundational axioms*, $\mathcal{F}$. The whole behavior of a CPS will be axiomatized in one formula Φ_{POS}, which has the form

$$\Phi_{\text{POS}} = \exists y\colon \mathsf{InitialState}(y) \wedge \forall x\colon x > y \rightarrow (\mathsf{Intermediate}(x) \vee \mathsf{End}(x))$$

Intuitively, Φ_{POS} states that there is an initial state y where no rule has been applied, it is consistent with $\mathcal{T}$, and either no rule is executable in y and there is no successor, or at least one rule is executable in y, there is a successor, and all the successors are either intermediate states (states with a successor), or final states (without successor). Intermediate and final states are required to be the result of some rule application, respect the inertia laws for the atoms in the working memory, and be consistent with the ontology. In addition, intermediate states are supposed to have a successor resulting from the application of a rule, whereas final states are not. Theorem 1 establishes the correspondence between the models of our formalization and the runs of a production system. Due to the space limitation, we give only an intuitive formulation of these theorems. Details can be found in [7].

Theorem 1 (Soundness and Completeness). *Let* $(\mathsf{PS}, \mathsf{WM}_0)$ *be a concrete PS, and* $\Phi_{POS} \cup \mathcal{F}$ *the FPL embedding of* $(\mathsf{PS}, \mathsf{WM}_0)$.

- $\Phi_{POS} \cup \mathcal{F}$ *entails that a fact* f *holds in the initial state if and only if* $\mathsf{WM}_0 \models_{\mathcal{T}} f$.
- $\Phi_{POS} \cup \mathcal{F}$ *entails that a fact* f *holds after applying* $r_1(\boldsymbol{c_1}) \dots r_n(\boldsymbol{c_n})$ *if and only if there is a run* $\mathcal{R}$ *of* PS *of the form* $\mathsf{WM}_0 \overset{r_1(\boldsymbol{c_1})}{\rightsquigarrow} \mathsf{WM}_1 \dots \mathsf{WM}_{n-1} \overset{r_n(\boldsymbol{c_n})}{\rightsquigarrow} \mathsf{WM}_n$ *such that* $\mathsf{WM}_n \models_{\mathcal{T}} f$

Now we turn to the problem of how to check properties of CPS. Typical properties of (concrete) production systems one would like to check are termination (all the runs are finite) and confluence (all the runs terminate with the same working memory). A complete list of these properties, their formal definition and logic encoding can be found in [3]. For concreteness and space limitation, we will only discuss termination, but our results hold for other properties as well. In [7] we define two types of concrete PS: **regular** and **light**. Intuitively, in a regular CPS, all rule conditions can be satisfied only by constants in the working memory, and actions behave either monotonically (never removing anything) or anti-monotonically (never adding anything). In light CPS, the ontology is such that all its consequences—given the working memory—can be finitely computed. An interesting class of such ontologies is the set of acyclic DL-lite TBoxes[2].

Theorem 2. *Checking termination of regular and of light concrete PS is decidable.*

5 Conclusion

In this paper we discussed different design options for the combination of production systems with DL ontologies, and presented a new syntax and a new semantics, named POS, for such combination, and explained its advantages. We also presented an embedding of POS into Fixed-Point Logic, giving in this way a model-theoretic semantics to the combination. Finally, we studied how, in some restricted cases, the static analysis of production rules, and the FPL embedding can be used to check properties (like termination) by means of logic entailment. These properties are not decidable in the general case. In the future, we plan to extend this work to cover the whole RIF-PRD language and to handle inconsistencies introduced by the rule applications.

Acknowledgments. We thank Jos de Bruijn for starting this work with us and the anonymous reviewers for useful comments and feedback. The work presented in this paper was partially supported by the European Commission under the project ONTORULE (IST-2009-231875).

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook. Cambridge University Press, Cambridge (2003)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-lite family. J. of Automated Reasoning 39(3), 385–429 (2007)
3. 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)
4. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On instance-level update and erasure in description logic ontologies. J. Log. and Comput. 19, 745–770 (2009)
5. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. 19(1), 17–37 (1982)
6. Liu, H., Lutz, C., Milicic, M., Wolter, F.: Updating Description Logic ABoxes. In: Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2006), pp. 46–56 (2006)
7. Rezk, M., de Bruijn, J., Nutt, W.: Combining production systems and ontologies. Technical report, Free University of Bolzano (2011), http://www.inf.unibz.it/~mrezk/techreportPOS.pdf