# Generalized Ontology-Based Production Systems

**Riccardo Rosati**
DIS, Sapienza Università di Roma, Italy
*rosati@dis.uniroma1.it*

**Enrico Franconi**
Free University of Bozen-Bolzano, Italy
*franconi@inf.unibz.it*

**Abstract**

We define generalized ontology-based production systems (GOPSs), which formalize a very general and powerful combination of ontologies and production systems. We show that GOPSs capture and generalize many existing formal notions of production systems. We introduce a powerful verification query language for GOPSs, which is able to express the most relevant formal properties of production systems previously considered in the literature. We establish a general sufficient condition for the decidability of answering verification queries over GOPSs. Then, we define Lite-GOPS, a particular class of GOPSs based on the use of a light-weight ontology language (*DL-Lite*$_{\mathcal{A}}$), a light-weight ontology query language (*EQL-Lite(UCQ)*), and a tractable semantics for updates over Description Logic ontologies. We show decidability of all the above verification tasks over Lite-GOPSs, and prove tractability of some of such tasks.

## Introduction

### Motivation

The integration of ontologies and production rules is a challenging task. Many approaches have recently dealt with this problem (Raschid 1994; Damasio, Alferes, and Leite 2010; Baral and Lobo 1995; Kowalski and Sadri 2009; Rezk and Nutt 2011; de Bruijn and Rezk 2009). However, known approaches suffer from the following limitations: (i) there is no unifying approach capturing all the above proposals within a coherent formal setting; (ii) no approach seems to be flexible enough to allow for the combination of *arbitrary* ontologies with production rules; (iii) there are very few results concerning the computational properties of such extended forms of production systems (an exception is (de Bruijn and Rezk 2009)).

The goal of this paper is to identify and explore a general way of combining ontologies and production rules. We argue that there are at least two strong motivations for pursuing such a goal. First, the existence of a framework which is general enough to capture the main existing proposals for the combination of ontologies and production rules makes it possible to easily and effectively study and compare the different proposals in a coherent formal setting. Second, this general framework makes it possible to identify and study decidability and complexity of reasoning in classes of systems combining ontologies and production rules. In this paper we are particularly interested in the *static analysis* of such systems (Lunardhi and Passino 1995; de Bruijn and Rezk 2009): thus, the form of reasoning we are interested in is the verification of dynamic properties in systems combining ontologies and production rules.



### Contribution

Our approach is based on a very abstract vision of an ontology, whose roots lie in the principles of knowledge representation (Levesque 1984): we see the ontology as a knowledge base defined through a *functional specification*. More precisely, the ontology is equipped with a *query language* and an *update language*. Such languages are provided with a semantics given by a function $ASK$ and a function $TELL$, respectively. The function $ASK$ provides the semantics of queries posed to an ontology; the function $TELL$ provides the semantics of updates over an ontology.

According to this view, the integration of ontologies with production rules is very simple and natural. Roughly speaking, production rules are "if *condition* then *action*" statements. Now, the functional specification of an ontology makes it very simple to define a combination of ontologies and production systems. This combination is based on the following, almost straightforward, considerations: when combining ontologies and production rules, it is natural to use ontology query languages to express rule conditions and ontology update languages to express rule actions; to interpret the meaning of such conditions and of such actions, it is natural to use the semantics of ontology queries and ontology updates, respectively. Thus, production rules are executed on an ontology: the condition language corresponds to the ontology query language, while the action language corresponds to the ontology update language. The production system uses the ontology as a working memory, and the semantics of the system is then given by the usual operational semantics of production rules, using the $ASK$ and $TELL$ functions to interpret conditions and actions, respectively.

Hence, this approach to the integration of ontologies and production rules is very natural. However, to effectively exploit such an approach, we have to face a big issue: in fact, very few ontology languages are equipped with a satisfac-

tory query language and/or a satisfactory update language.[1] Few results are available in the field of updating ontologies, in particular Description Logic (DL) ontologies.

This approach might seem just elegant but of no use: fortunately, this is not true for at least two reasons. First of all, the functional view of ontologies makes it clear that the technical obstacles towards the combination of ontologies and production systems are only due to the ontology component, in the sense that such obstacles are due to the fact that the specifications of ontologies are often still incomplete (they lack a proper query functionality and/or a proper update functionality). Moreover, some recent results allow us to identify ontology specifications that actually match the requirements for a meaningful combination with production rules. In fact, many expressive (and decidable) query languages have been defined for DL ontologies (e.g., (Sirin and Parsia 2007; Calvanese et al. 2007a; Glimm 2011)), and some recent approaches have proposed interesting semantics for updates over DL ontologies, as well as algorithms for effectively computing such updates (e.g., (De Giacomo et al. 2009; Calvanese et al. 2010; Lenzerini and Savo 2011; Liu et al. 2011)).

We formalize the above vision of the combination of ontologies and production systems as follows.

- We define *generalized ontology-based production systems (GOPSs)*, which formalize a very general and powerful combination of ontologies and production systems based on the functional specification of ontologies.
- We introduce a powerful verification query language for GOPSs, able to express the most relevant dynamic properties of production systems considered in the literature. In particular, we define a variant of $\mu$-calculus (Emerson 1996) which is tailored for expressing properties of GOPSs.
- We study reasoning over GOPSs and establish a general sufficient condition for the decidability of answering verification queries over GOPSs.
- Next, we turn our attention to specific ontology languages. More specifically, we define *Lite-GOPSs*, a particular class of GOPSs based on the use of a light-weight ontology language (*DL-Lite*$_\mathcal{A}$) (Poggi et al. 2008), a light-weight ontology query language (*EQL-Lite(UCQ)*) (Calvanese et al. 2007a), and a tractable semantics for updates over Description Logic ontologies (Lenzerini and Savo 2011).
- We show decidability of the above verification tasks over Lite-GOPSs, and prove tractability of some of such tasks.

[1] Here, by *satisfactory* language we mean that both syntax (i.e., the expressiveness) and semantics of the language should be adequate. For instance, we argue that a semantics for the update action of inserting an axiom which corresponds to the simple syntactic addition of the axiom to the DL ontology is to be considered as unsatisfactory, since it is not coherent with the semantics of the ontology itself.

## Preliminaries

We start by briefly recalling production rules and Description Logics.

Concerning production rules, essentially we stick to the RIF-PRD specification (de Sainte Marie, Hallmark, and Paschke editors 2010). A production rule system (or production system) is a pair $\langle F_0, P\rangle$ where $F_0$ is a state of a *fact base* and $P$ is a set of *production rules*. A state of a fact base is simply a finite set of facts (ground atomic formulas). A production rule is an expression of the form

$$FORALL\ \vec{x} : IF\ \phi(\vec{x})\ THEN\ \alpha_1(\vec{x}), \ldots, \alpha_n(\vec{x})$$

such that:

- $\phi(\vec{x})$ is a first-order formula, called *condition*, with free variables $\vec{x}$;
- every $\alpha_i(\vec{x})$ is such that for every *ground substitution* $\langle \vec{x}, \vec{c}\rangle$, i.e., a substitution of the rule variables with constants, $\alpha_i(\vec{c})$ is an *action*, i.e., an expression of the form `Assert`$(f)$ or `Retract`$(f)$, where $f$ is a fact.

A *rule instance* is the variable-free rule obtained applying a ground substitution to a production rule. A *priority* is associated to every rule instance.

The semantics of production systems is expressed in terms of a transistion system. Such a notion is based on the semantics of execution of a production rule over a fact base and on the notion of *conflict resolution strategy*. Given a production rule $p$ of the above form, a ground substitution $\langle \vec{x}, \vec{c}\rangle$, and a state of the fact base $F$, the rule instance $p(\vec{c})$ *matches* $F$ if the formula $\phi(\vec{c})$ is satisfied by $F$. The set of rule instances matching a state $F$ of a fact base is called the *conflict set* of $F$. The *execution* of $p(\vec{c})$ over $F$ is a state of the fact base $F'$ obtained from $F$ by applying the sequence of fact addition (`Assert`) and fact deletion (`Retract`) actions corresponding to $\alpha_1(\vec{c}), \ldots, \alpha_n(\vec{c})$. A *conflict resolution strategy* is a function that, given a conflict set and information on the previous history of the system, picks one rule instance among the ones in the conflict set.

The operational semantics of a production system is given by a *transition system*: roughly, every state[2] of a transition system represents a state of the fact base plus the information needed by the conflict resolution strategy, and there is a transition from one state $s$ to another state $s'$ (labeled by the sequence of actions of $p(\vec{c})$) if the rule $p(\vec{c})$ is the one picked by the conflict resolution strategy in $s$ and $s'$ is the state resulting by the execution of rule $p(\vec{c})$ in $s$: in particular, the state $F'$ of the fact base of $s'$ is the state of the fact base resulting from the execution of rule $p(\vec{c})$ on the state $F$ of the fact base of $s$. For more details, we refer to (de Sainte Marie, Hallmark, and Paschke editors 2010).

As explained above, the conflict resolution strategy is a central aspect of the semantics of production systems. The RIF-PRD specification formalizes a particular conflict resolution strategy, whose principles are the same as the conflict resolution principles of real prodution rule systems: the so-called `rif:forwardChaining` strategy. The

[2] This refers to the so-called *cyclic states* of the transition system, which are distinct from the *transitional states*.

`rif:forwardChaining` strategy can be described as follows. Given a conflict set:

1. (*refraction step*) if a rule instance has been executed in a given state of the system, it is no longer eligible for execution as long as it satisfies the states of facts associated to all the subsequent system states; this property is called *refraction*. Therefore, all the refracted rule instances are removed from further consideration;
2. (*priority step*) the remaining rule instances are ordered by decreasing priority, and only the rule instances with the highest priority are kept for further consideration;
3. (*recency step*) the rule instances are ordered by the number of consecutive system states in which they have been in the conflict set, and only the most recent rule instances are kept for further consideration;
4. (*tie-break step*) any remaining tie is broken is some way, and a single rule instance is kept for firing.

Description Logics (DLs) (Baader et al. 2003) allow for expressing knowledge in terms of *atomic concepts*, i.e., unary predicates, and *atomic roles*, i.e., binary predicates. General concepts and roles are built through the constructs allowed in the DL: such constructs are usually expressible in first-order logic (FOL). A *DL ontology* is formed by a set of *assertions*, typically divided into a *TBox*, expressing intensional knowledge, and an *ABox*, expressing extensional knowledge. Again, usually such assertions can be expressed as FOL sentences (i.e., closed FOL formulas). Thus, in most cases DL ontologies can be seen as FOL theories (of specific forms). The only notable exceptions are those DLs that include some form of second-order constructs, such as transitive closure or fixpoints (Baader et al. 2003).

In the following, we will speak about a *specification language* for ontologies, about *queries* to an ontology, and about *updates* to an ontology. There are several approaches that define and study query languages over DL ontologies (e.g. (Levy and Rousset 1998; Calvanese, De Giacomo, and Lenzerini 1998; Calvanese et al. 2007b; Lutz 2008)) and updates over DL ontologies (e.g., (Liu et al. 2006; De Giacomo et al. 2009; Calvanese et al. 2010; Lenzerini and Savo 2011)).

## GOPS

In this section we define syntax and semantics of generalized ontology-based production systems (GOPSs).

### Syntax

We start from the following pairwise disjoint alphabets: an alphabet of predicates $Pred$, an alphabet of constants $Const$, an alphabet of variables $Var$ and an alphabet of production rule identifiers $RuleID$. Our notion of production system builds on three languages over the above alphabets $Pred$, $Const$ and $Var$:

- an *ontology specification language* $\mathcal{OL}$ which specifies the syntax of the ontology;
- an *ontology query language* $\mathcal{QL}$ which defines the queries over the ontology;
- an *ontology update language* $\mathcal{UL}$ which defines the updates over the ontology.

An *ontology* is a set of formulas from $\mathcal{OL}$. An *ontology query* is a formula of $\mathcal{QL}$. An *ontology update* is a formula of $\mathcal{UL}$. An ontology query with no occurrences of free variables is called a *Boolean* ontology query. A *ground* ontology update is an update with no occurrences of free variables.

We denote by $\phi(\vec{x})$ a formula $\phi$ containing the free variables $\vec{x}$ ($\vec{x}$ is a tuple of variable symbols). Given an $n$-tuple $\vec{x}$, a *ground substitution* of $\vec{x}$ is a pair $\langle \vec{x}, \vec{c} \rangle$, where $\vec{c}$ is an $n$-tuple of constants from $Const$. Given a formula with $n$ free variables $\phi(\vec{x})$, and an $n$-tuple of constants $\vec{c}$, the formula $\phi(\vec{c})$ obtained from $\phi$ by applying the ground substitution $\langle \vec{x}, \vec{c} \rangle$, i.e., by replacing the variables in $\vec{x}$ with the corresponding constants in $\vec{c}$, is called a *grounding* of $\phi(\vec{x})$.

**Definition 1** *A* Generalized Ontology-Based Production System (GOPS) $\mathcal{G}$ *is a pair* $\langle \mathcal{O}_{in}, \mathcal{P} \rangle$ *where:*

- $\mathcal{O}_{in}$ *is an ontology;*
- $\mathcal{P}$ *is a set of* production rules*. A production rule is an expression of the form*

$$FORALL\ \vec{x} : IF\ \phi(\vec{x})\ THEN\ \alpha_1(\vec{x}), \ldots, \alpha_n(\vec{x}) \quad (1)$$

*such that:*

  - $\phi(\vec{x})$ *is an ontology query over* $\mathcal{O}$ *(i.e., a query posed to* $\mathcal{O}$*) with free variables* $\vec{x}$*. The variables in* $\vec{x}$ *are symbols from* $Var$ *and are called the* variables *of the production rule;*
  - *every* $\alpha_i(\vec{x})$ *is such that every grounding* $\alpha_i(\vec{c})$ *of* $\alpha_i(\vec{x})$ *is an ontology update.*

*Every production rule has an associated* rule identifier*, i.e., a symbol from* $RuleID$ *which is associated with no other production rule. A* ground *production rule is a production rule without variables:*

$$IF\ \phi\ THEN\ \alpha_1, \ldots, \alpha_n \quad (2)$$

*where* $\phi$ *is an ontology query without free variables and every* $\alpha_i$ *is an ontology update.*

Given a production rule with identifier $p$ of the form (1), an *instance* of rule $p$ is a ground production rule obtained from (1) by applying a ground substitution $\langle \vec{x}, \vec{c} \rangle$, i.e., a ground production rule of the form

$$IF\ \phi(\vec{c})\ THEN\ \alpha_1(\vec{c}), \ldots, \alpha_n(\vec{c})$$

The *rule instance identifier* associated with the above rule instance is $p(\vec{c})$ (which represents the fact that the above rule can be obtained by instantiation of the rule $p$ with constants $\vec{c}$).

### Semantics

The semantics of GOPSs is based on three functions: (i) the function $ASK$, which provides the semantics of ontology queries; (ii) the function $TELL$, which provides the semantics of ontology updates; (iii) the function $\Phi$, which governs the execution of production rules. These functions are introduced in the following.

**Ontology queries.** The semantics of ontology queries is defined by the function $ASK(\mathcal{O}, \phi(\vec{x}))$. For every ontology $\mathcal{O}$ and for every ontology query $\phi(\vec{x})$ with free variables $\vec{x}$, $ASK(\mathcal{O}, \phi(\vec{x}))$ is a set of ground substitutions for $\vec{x}$. Notice that, if $\phi$ has no free variables, i.e., $\phi$ is a Boolean query, then $ASK(\mathcal{O}, \phi)$ is either the set of ground substitutions containing the empty ground substitution $\{\emptyset\}$ (which means that the query $\phi$ is entailed by $\mathcal{O}$) or the empty set of ground substitutions $\emptyset$ (which means that $\phi$ is not entailed by $\mathcal{O}$).

**Ontology updates.** The semantics of ontology updates is defined by the partial function $TELL(\mathcal{O}, \alpha)$. More precisely, given an ontology $\mathcal{O}$ and an ontology update $\alpha$, $TELL(\mathcal{O}, \alpha)$ is either undefined or is equal to one ontology $\mathcal{O}'$. Intuitively, the case when $TELL(\mathcal{O}, \alpha)$ is undefined encodes those situations in which it is impossible (according to the intended semantics of ontology updates) to update the ontology $\mathcal{O}$ according to the update action $\alpha$.

**Production rules.** Given a ground production rule $p_g$ of the form (2) and an ontology $\mathcal{O}$, we say that $p_g$ is *fireable* in $\mathcal{O}$ if the following conditions hold:

1. $ASK(\phi, \mathcal{O}) \neq \emptyset$;
2. there is a sequence of ontologies $\mathcal{O}_0, \ldots, \mathcal{O}_n$ such that $\mathcal{O}_0 = \mathcal{O}$ and, for every $i$ such that $0 \leq i \leq n-1$, $TELL(\mathcal{O}_i, \alpha_{i+1})$ is defined and is equal to $\mathcal{O}_{i+1}$.

Moreover, if $p_g$ is fireable in $\mathcal{O}$, we denote by $EXEC(p_g, \mathcal{O})$ the result of the execution of $p_g$ over $\mathcal{O}$; i.e., the above ontology $\mathcal{O}_n$.

Given a GOPS $\mathcal{G}$ and an ontology $\mathcal{O}$, we denote by $CS(\mathcal{O}, \mathcal{G})$ the *conflict set* for $\mathcal{O}$ and $\mathcal{G}$, i.e., the set of instances $p_g$ of the production rules from $\mathcal{P}$ such that $p_g$ is fireable in $\mathcal{O}$.

**GOPS.** A *GOPS graph* $GG = \langle N, S^{in}, E, L^e \rangle$ is a directed graph: $N$ is the set of nodes, called *GOPS states*, which are pairs of the form $\langle id, \mathcal{O} \rangle$ where $id$ is the *state identifier* and $\mathcal{O}$ is an ontology; $S^{in}$ is a node of $N$, called the *initial state* of $GG$; $E$ is the set of edges, i.e., pairs of GOPS states; and $L^e$ is function which labels the edges with rule instance identifiers. A *GOPS path* is a path of a GOPS graph.

A *(partial) conflict resolution function* is a function $\Phi$ over paths of a GOPS graph. Let $\mathcal{G}$ be a GOPS and let $\pi$ be a GOPS path. Let $S_e = \langle id, \mathcal{O} \rangle$ be the ending state of $\pi$. Then, the value of $\Phi(\mathcal{G}, \pi)$ is $\emptyset$ if $CS(\mathcal{O}, \mathcal{G}) = \emptyset$; otherwise, the value of $\Phi(\mathcal{G}, \pi)$ is a non-empty set of ground production rules $\mathcal{P}_g$ such that $\mathcal{P}_g \subseteq CS(\mathcal{O}, \mathcal{G})$. If the value of the function $\Phi$ is always either the empty set or a singleton set (i.e., $\Phi$ selects at most one rule among the ones in $CS(\mathcal{O}, \mathcal{G})$), then we call $\Phi$ a *total conflict resolution function*. Informally, a partial conflict resolution function selects a subset of the rule instances in the conflict resolution set $CS(\mathcal{O}, \mathcal{G})$. Intuitively, such a function chooses a subset of the rules in the conflict set: any of the rules in such a subset could be chosen for execution. On the other hand, a total conflict resolution function actually resolves the conflict among the rules, since it just selects one rule. While real production systems only adopt total conflict resolution functions, we introduce partial conflict resolution functions because this allows us to study the properties of *families* of conflict resolution stategies. For instance, the `rif:forwardChaining` strategy previously described can be seen as a partial conflict resolution function, since it does not actually specify the final *tie-break* step. So, studying GOPS under this partial conflict resolution function makes it possible to verify the formal properties of the `rif:forwardChaining` strategy independently of the particular implementation of the tie-break step.

Notice also that the conflict resolution function does not only depend on the current state of the ontology (i.e., the ontology labeling the final state of the GOPS path), but depends on a GOPS path, which represents the whole "history" of the GOPS evolution. This reflects the conflict resolution strategies adopted in real systems: e.g., the above described `rif:forwardChaining` strategy depends not only on the current state of the fact base, but also on the previous states of the transition system.

The semantics of a GOPS $\mathcal{G}$ is defined by the notion of transition system.

**Definition 2** *Given a GOPS $\mathcal{G} = \langle \mathcal{O}_{in}, \mathcal{P} \rangle$, the* transition system of $\mathcal{G}$*, denoted by $TS(\mathcal{G})$, is the GOPS graph $\langle N, S^{in}, E, L^e \rangle$ defined inductively as follows:*

1. *the initial state $S^{in}$ is the state $\langle S^{\mathcal{G}}_{in}, \mathcal{O}_{in} \rangle$;*
2. *for every state $S$ of the form $\langle id, \mathcal{O} \rangle$ belonging to $N$, let $\pi(S)$ be the path of $TS(\mathcal{G})$ starting in $S^{in}$ and ending at $S$. If $p_g$ is the identifier of a rule instance such that $p_g \in \Phi(\mathcal{G}, \pi(S))$ then: (i) $\langle p_g(id), \mathcal{O}' \rangle \in N$ with $\mathcal{O}' = EXEC(p_g, \mathcal{O})$, (ii) $\langle S, \langle p_g(id), \mathcal{O}' \rangle \rangle \in E$, and (iii) $L^e(\langle S, \langle p_g(id), \mathcal{O}' \rangle \rangle = p_g$.*

Informally, $TS(\mathcal{G})$ is the GOPS graph built starting from the initial state and adding, for every state $S$, an edge (transition) and a new state for every rule instance selected by the conflict resolution function $\Phi((\mathcal{G}, \pi(S))$: the new state is the state obtained by the execution of the rule instance on $S$.

We call *run* of $\mathcal{G}$ any path of $TS(\mathcal{G})$ starting at the state whose identifier is $S^{\mathcal{G}}_{in}$ and such that the path either is infinite or ends at a sink node (i.e., a node which does not have any outcoming edge). Of course, if $\Phi$ is a total conflict resolution function, then $TS(\mathcal{G})$ contains only one run. Intuitively, the transition system $TS(\mathcal{G})$ represents all the possible runs of $\mathcal{G}$, i.e., all the possible sequences of execution of production rule instances starting from the initial ontology. Of course, the transition system may be infinite, since there may be infinite runs of $\mathcal{G}$.

## Verification query language

In this section we define the verification query language $\mathcal{V}(\mathcal{QL})$. The language $\mathcal{V}(\mathcal{QL})$ is an extension of $\mu$-calculus where state formulas are formulas of the ontology query language $\mathcal{QL}$. This language builds from an analogous previous proposal (Cangialosi et al. 2010) in the field of artifact-centric services, and is also related to the verification language proposed in (de Bruijn and Rezk 2009).

To specify dynamic properties, we will use a variant of $\mu$-calculus (Emerson 1996). The choice of $\mu$-calculus is a very natural one in our setting, since: (i) we are interested in the static analysis of GOPSs; (ii) the semantics of GOPSs is

given in terms of labelled transition systems: (iii) $\mu$-calculus is a powerful language for expressing properties of labelled transition systems. Moreover, the known verification techniques for $\mu$-calculus may constitute the basis for verification techniques in our setting.

The variant of $\mu$-calculus that we propose conforms with the basic assumption of our formalism: the use of ontologies and ontology queries to describe the properties of a state.

To define verification queries, we need a further alphabet, the alphabet of predicate variables $\mathcal{PV}$, which is pairwise disjoint with all the alphabets introduced in the previous section.

**Definition 3** *A* verification query *is specified by the following abstract syntax:*

$$\psi ::= \phi \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid [p]\psi \mid \langle p\rangle\psi \mid \mu Z.\psi \mid \nu Z.\psi$$

*where $\phi$ is a Boolean ontology query (i.e., a formula from $\mathcal{QL}$ without free variables), $Z$ is a predicate variable symbol from $\mathcal{PV}$, and $p$ is: (i) a rule instance identifier; or (ii) a rule identifier; or (iii) the symbol $-$. The* verification query language $\mathcal{V}(\mathcal{QL})$ *is the language of verification queries.*

The symbols $\mu$ and $\nu$ can be considered as quantifiers, and we make use of the notions of scope, bound and free occurrences of variables, closed formulas, etc. The definitions of these notions are the same as in first-order logic, treating $\mu$ and $\nu$ as quantifiers. For formulas of the form $\mu Z.\psi$ and $\nu Z.\psi$, we require the *syntactic monotonicity* of $\psi$ wrt $Z$: Every occurrence of the variable $Z$ in $\psi$ must be within the scope of an even number of negation signs. In $\mu$-calculus, given the requirement of syntactic monotonicity, the least fixpoint $\mu Z.\psi$ and the greatest fixpoint $\nu Z.\psi$ always exist.

Let $GG = \langle N, S^{in}, E, L^e\rangle$ be a GOPS graph. A valuation on $GG$ is a mapping from the predicate variables appearing in $\psi$ to subsets of $N$. We assign meaning to $\mathcal{V}(\mathcal{QL})$ formulas through an *evaluation function* $Eval_{GG}(\cdot)$, which maps $\mathcal{V}(\mathcal{QL})$ formulas to subsets of $N$. In the following, $id$ represents a rule identifier, and the $id(\vec{c})$ represents the identifier of an instance of the rule $id$.

**Definition 4** *The* evaluation function $Eval_{GG}(\cdot)$ *for a GOPS graph $GG = \langle N, S^{in}, E, L^e\rangle$ is defined inductively as follows:*

$Eval_{GG}(\phi) = \{s = \langle s_{id}, \mathcal{O}\rangle \mid s \in N \text{ and } ASK(\phi, \mathcal{O}) \neq \emptyset\}$
$Eval_{GG}(Z) = N' \subseteq N$
$Eval_{GG}(\neg\psi) = N - Eval_{GG}(\psi)$
$Eval_{GG}(\psi_1 \wedge \psi_2) = Eval_{GG}(\psi_1) \cap Eval_{GG}(\psi_2)$
$Eval_{GG}(\langle id(\vec{c})\rangle\psi) = \{s \in N \mid \exists s'.\ \langle s, s'\rangle \in E \text{ and } L^e(s, s') = id(\vec{c}) \text{ and } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}([id(\vec{c})]\psi) = \{s \in N \mid \forall s'.\ (\langle s, s'\rangle \in E \text{ and } L^e(s, s') = id(\vec{c})) \text{ implies } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}(\langle id\rangle\psi) = \{s \in N \mid \exists s', \vec{c}.\ \langle s, s'\rangle \in E \text{ and } L^e(s, s') = id(\vec{c}) \text{ and } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}([id]\psi) = \{s \in N \mid \forall s', \vec{c}.\ (\langle s, s'\rangle \in E \text{ and } L^e(s, s') = id(\vec{c})) \text{ implies } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}(\langle -\rangle\psi) = \{s \in N \mid \exists s'.\ \langle s, s'\rangle \in E \text{ and } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}([-]\psi) = \{s \in N \mid \forall s'.\ \langle s, s'\rangle \in E \text{ implies } s' \in Eval_{GG}(\psi)\}$
$Eval_{GG}(\mu Z.\psi) = \bigcap\{N' \subseteq N \mid Eval_{GG}[Z \leftarrow N'](\psi) \subseteq N'\}$
$Eval_{GG}(\nu Z.\psi) = \bigcup\{N' \subseteq N \mid N' \subseteq Eval_{GG}[Z \leftarrow N'](\psi)\}$

*where $Eval_{GG}[X \leftarrow N'](\psi)$ represents the value $Eval_{GG}(\psi)$ when every occurrence of the predicate variable $X$ is evaluated as the set of states $N'$.*

Intuitively, the evaluation function $Eval_{GG}(\cdot)$ assigns to the various constructs of $\mu$-calculus the following meanings:

- The boolean connectives have the expected meaning.
- The evaluation of $\langle id(\vec{c})\rangle\psi$ includes the states $s \in N$ such that at state $s$ there is an execution of the rule instance $id(\vec{c})$ that leads to a state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of $\langle id(\vec{c})\rangle\psi$ is "there exists an execution of rule instance $id(\vec{c})$ that leads to a state where $\psi$ holds".
- The evaluation of $[id(\vec{c})]\psi$ includes the states $s \in N$ such that each execution of the rule instance $id(\vec{c})$ at state $s$ leads to some state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of the operator $[id(\vec{c})]$ is "every execution of the rule instance $id(\vec{c})$ leads to a state where $\psi$ holds".
- The evaluation of $\langle id\rangle\psi$ includes the states $s \in N$ such that at state $s$ there is an execution of any instance of the rule $id$ that leads to a state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of $\langle id\rangle\psi$ is "there exists an execution of an instance of rule $id$ that leads to a state where $\psi$ holds".
- The evaluation of $[id]\psi$ includes the states $s \in N$ such that each execution of any instance of the rule $id$ at state $s$ leads to some state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of the operator $[id]$ is "every execution of an instance of rule $id$ leads to a state where $\psi$ holds".
- The evaluation of $\langle -\rangle\psi$ includes the states $s \in N$ such that at state $s$ there is an execution of an arbitrary rule instance that leads to a state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of $\langle -\rangle\psi$ is "there exists an execution of a rule instance that leads to a state where $\psi$ holds".
- The evaluation of $[-]\psi$ includes the states $s \in N$ such that each execution of any arbitrary rule instance at state $s$ leads to some state $s'$ included in the evaluation of $\psi$. Thus, the intuitive meaning of the operator $[-]$ is "every execution of rule instances leads to a state where $\psi$ holds".
- The evaluation of $\mu Z.\psi$ is the *smallest subset* $N'$ of $N$ such that, assigning to $Z$ the evaluation $N'$, the resulting evaluation of $\psi$ is contained in $N'$.
- Similarly, the evaluation of $\nu Z.\psi$ is the *greatest subset* $N'$ of $N$ such that, assigning to $Z$ the evaluation $N'$, the resulting evaluation of $\psi$ contains $N'$.

The reasoning problem we are interested in is *model checking*. Let $GG = \langle N, S^{in}, E, L^e\rangle$ be a GOPS graph, let $s \in N$ be one of its states, and let $\psi$ be a verification query. The related model checking problem is to verify whether $s \in Eval_{GG}(\psi)$.

In particular, in the following we focus on the Boolean problem of verifying whether the *initial state* $S^{in}$ is in the evaluation of a verificaton query $\psi$. Thus, for every verification query $\psi$ and for every GOPS graph $GG = \langle N, S^{in}, E, L^e \rangle$, we define $Entailed(\psi, GG)$ as $true$ if $S^{in} \in Eval_{GG}(\psi)$, and $false$ otherwise.

It is immediate to verify that checking verification queries over a finite model is decidable if answering ontology queries is decidable. In particular, the following property holds.

**Theorem 1** *Checking a verification query $\psi$ over a finite GOPS graph $GG = \langle N, S^{in}, E, L^e \rangle$ can be done in time*

$$O((|GG| \cdot |\psi|)^k)$$

*where $|GG| = |N| + |E|$, i.e., the number of states plus the number of transitions of GG, $|\psi|$ is the size of formula $\psi$ (in fact, considering propositional formulas as atomic), and $k$ is the number of nested fixpoints, i.e., fixpoints whose variables are one within the scope of the other, using an oracle for deciding $ASK(\phi, \mathcal{O}) \neq \emptyset$ for every verification query $\phi$ and ontology $\mathcal{O}$.*

Finally, we introduce a notion of bisimilarity for GOPS graph which will be important to establish decidability results on answering verification queries over GOPSs.

Let $S_1 = \langle id_1, \mathcal{O}_1 \rangle$, $S_2 = \langle id_2, \mathcal{O}_2 \rangle$ be two GOPS states. We say that $S_1$ and $S_2$ are *locally bisimilar* if, for every ontology query $\phi$, $ASK(\phi, \mathcal{O}_1) = ASK(\phi, \mathcal{O}_2)$.

Given two GOPS graphs $GG_1 = \langle N_1, S_1^{in}, E_1, L_1^e \rangle$, $GG_2 = \langle N_2, S_2^{in}, E_2, L_2^e \rangle$, we say that $GG_1$ and $GG_2$ are *bisimilar* if there exists a function $\beta$ from the states of $GG_1$ to the states of $GG_2$ such that $\beta(S_1^{in}) = S_2^{in}$ and, for every state $S$ of $GG_1$, the following conditions hold:

1. $S$ and $\beta(S)$ are locally bisimilar;
2. for each state $S'$ such that $\langle S, S' \rangle$ is an edge of $GG_1$, $\langle \beta(S), \beta(S') \rangle$ is an edge of $GG_2$, and $L_1^e(\langle S, S' \rangle) = L_2^e(\langle \beta(S), \beta(S') \rangle)$;
3. for each state $S''$ such that $\langle \beta(S), S'' \rangle$ is an edge of $GG_2$, there exists a state $S'$ of $GG_1$ such that $\langle S, S' \rangle$ is an edge of $GG_1$, $S'' = \beta(S')$, and $L_1^e(\langle S, S' \rangle) = L_2^e(\langle \beta(S), \beta(S') \rangle)$.

**Theorem 2** *If $GG_1$ and $GG_2$ are bisimilar GOPS graphs, then for every verification query $\psi$, $Entailed(\psi, GG_1) = Entailed(\psi, GG_2)$.*

Given a GOPS $\mathcal{G}$ and a verification query $\psi$, we say that $\psi$ *is entailed by* $\mathcal{G}$ if $Entailed(\psi, TS(\mathcal{G})) = true$. Moreover, we define $Ans_{GOPS}(\psi, \mathcal{G})$ as $Entailed(\psi, TS(\mathcal{G}))$. The problem of *answering a verification query $\psi$ over a GOPS $\mathcal{G}$* amounts to establishing whether $Ans_{GOPS}(\psi, \mathcal{G}) = true$.

## Expressing dynamic properties of GOPSs

In this section we show that the verification query language defined in the previous section is a very powerful tool for the static analysis of GOPSs. In particular, using the verification query language $\mathcal{V}(\mathcal{QL})$, we provide a formalization of some interesting and complex dynamic properties of GOPSs. In the following, we refer to a GOPS $\mathcal{G}$ and use the symbol $\mathcal{I}_\mathcal{G}$ to denote the set of production rule identifiers of the GOPS $\mathcal{G}$.

Moreover, without loss of generality, we assume that the language allows to express a Boolean ontology query which holds in every ontology, and we denote by $TRUE$ such a query (and denote by $FALSE$ the formula $\neg TRUE$). In fact, even in the case when the ontology query language does not allow for such a tautological query, it is possible to easily modify the specification of the GOPS $\mathcal{G}$ in a way such that there exists a Boolean ontology query that holds in every state of the transition system of $\mathcal{G}$: this kind of "locally tautological" ontology query for $\mathcal{G}$ is enough for our purposes.

As an extension of $\mu$-calculus, the language $\mathcal{V}(\mathcal{QL})$ has the same abilities of $\mu$-calculus of expressing complex dynamic properties over transition systems. As we will see in the following examples, the typical relevant properties of dynamic systems which are considered in verification can be expressed over GOPSs through $\mathcal{V}(\mathcal{QL})$.

The following list presents some relevant dynamic properties of a GOPS $\mathcal{G}$ expressed in terms of $\mathcal{V}(\mathcal{QL})$ queries:

- *Every run terminates in a state where the ontology query $\phi$ holds* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \mu X.(([\text{-}]FALSE \wedge \phi) \vee (\langle - \rangle TRUE \wedge [\text{-}]X))$
- *Every run is finite* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \mu X.[\text{-}]X$
- *An ontology query $\phi$ eventually holds forever in some run* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \mu X.((\nu Y.\phi \wedge \langle - \rangle Y) \vee \langle - \rangle X)$
- *An ontology query $\phi$ eventually holds forever in every run* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \mu X.((\nu Y.\phi \wedge [\text{-}]Y) \vee (\langle - \rangle TRUE \wedge [\text{-}]X))$
- *The production rule $id$ is applied in every run* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \mu X.(\langle id \rangle.(TRUE \vee (\langle - \rangle TRUE \wedge [\text{-}]X)))$
- *Every production rule is applied in every run* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \bigwedge_{id \in \mathcal{I}_\mathcal{G}} \mu X.(\langle id \rangle.TRUE \vee (\langle - \rangle TRUE \wedge [\text{-}]X))$
- *Rule $id$ is never applied* if and only if $Entailed(\psi_1^{id}, TS(\mathcal{G})) = true$, where $\psi_1^{id}$ is the formula:
  $\psi_1^{id} = \mu X.([\text{-}].FALSE \vee ([id]FALSE \wedge [\text{-}]X))$
- *Rule $id$ is applied at most once in every run* if and only if $Entailed(\psi_2^{id}, TS(\mathcal{G})) = true$, where $\psi_2^{id}$ is the formula:
  $\psi_2^{id} = \mu Y.(\psi_1^{id}) \vee (([id](\psi_1^{id}) \wedge \bigwedge_{id' \in \mathcal{I}_G - \{id\}} [id']Y)$
  and $\psi_1^{id}$ is the formula defined in the previous point.
- *Every rule is applied at most once in every run* if and only if $Entailed(\psi, TS(\mathcal{G})) = true$, where $\psi$ is the formula:
  $\psi = \bigwedge_{id \in \mathcal{I}_G} (\psi_2^{id})$
  and $\psi_2^{id}$ is the formula defined in the previous point.

## Reasoning over GOPSs

In this section we focus on the reasoning task of answering verification queries over GOPSs. We start from the following undecidability result.

**Theorem 3** *If either the function $ASK$ is undecidable or the function $TELL$ is undecidable,*[3] *then answering verification queries over GOPSs is undecidable.*

As a consequence of the above result, we have that we must carefully choose the ontology query language to be coupled with a DL ontology language: in fact, all the typical relational database query languages are undecidable over DL ontologies, and even fragments of such languages are undecidable for many DL languages (see e.g. (Rosati 2007)). Indeed, the language of unions of conjunctive queries (a subset of FOL queries) is one of the most expressive languages which is decidable over (almost all) DL ontologies (see e.g. (Lutz 2007)). Analogous considerations hold in principle for updates over DL ontologies, even though research in this field is at an earlier stage than research in query answering, and a classification of decidable and undecidable update languages and update semantics for DLs is not available yet.

We focus now to *decidable* classes of GOPS. Our aim is to provide sufficient conditions for the decidability of answering verification queries over GOPSs. Our analysis starts from the fact that, according to Theorem 1, answering verification queries over a GOPS is decidable if the transition system $TS(\mathcal{G})$ of $\mathcal{G}$ can be built in a finite amount of time. This is not the case, of course, when $TS(\mathcal{G})$ is infinite: however, it might still be possible to construct in a finite amount of time a GOPS graph $GG$ that is bisimilar to $TS(\mathcal{G})$. This implies decidability of our reasoning task, since, due to Theorem 2, we can evaluate verification queries using $GG$ instead of $TS(\mathcal{G})$. In the following, we look for sufficient conditions (on the specification of $\mathcal{G}$) for the construction of a GOPS graph that is bisimilar to $TS(\mathcal{G})$.

Let $\mathcal{GO}$ be the set of all GOPS, let $\mathcal{GP}$ be the set of GOPS paths, let $\mathcal{GR}$ be the set of all ground production rules, and let $\mathcal{D}$ be an arbitrary set. A *finite transformation* of a conflict resolution function $\Phi$ is a pair of functions $\langle \tau_\pi, \tau_\Phi \rangle$ where $\tau_\pi : \mathcal{GP} \rightarrow \mathcal{D}$, $\tau_\Phi : \mathcal{GO} \times \mathcal{D} \rightarrow 2^{\mathcal{GR}}$, such that, for every GOPS $\mathcal{G}$: (i) for every path $\pi \in TS(\mathcal{G})$, $\Phi(\mathcal{G}, \pi) = \tau_\Phi(\mathcal{G}, \tau_\pi(\pi))$; (ii) the set $\{d \mid d = \tau_\pi(\pi)$ and $\pi$ is a path of $TS(\mathcal{G})\}$ is finite.

The idea behind a finite transformation of a conflict resolution function is that it is possible to formulate the function without using all the information on the previous history of the system (represented by the whole GOPS path ending in the current state), but using only an approximation of such information, such that the number of possible instances of such an approximation which is relevant for a given transition system is finite. Notice that the number of possible GOPS paths is infinite, and that in general a conflict resolution function may admit no finite transformations. The idea of finite transformation is crucial to prove a general result on the finite representability of the transition system of a GOPS, and hence a general decidability result on answering verification queries over GOPSs.

Given a GOPS $\mathcal{G}$ and a finite transformation $T = \langle \tau_\pi, \tau_\Phi \rangle$ for $\Phi$, the *$T$-core* of $\mathcal{G}$, denoted by *$T$-core*$(\mathcal{G})$, is the GOPS graph obtained from $TS(\mathcal{G})$ by collapsing every pair of states $S, S'$ such $S$ and $S'$ are locally bisimilar and $\tau_\pi(\pi_S) = \tau_\pi(\pi_{S'})$, where $\pi_S$ and $\pi_{S'}$ are the paths of $TS(\mathcal{G})$ ending in $S$ and $S'$, respectively.

**Lemma 1** *Given a GOPS $\mathcal{G}$ and a finite transformation $T = \langle \tau_\pi, \tau_\Phi \rangle$ for $\Phi$, the $T$-core of $TS(\mathcal{G})$ is a finite GOPS graph.*

The following lemma states that a $T$-core of $\mathcal{G}$ constitutes a correct representation of $TS(\mathcal{G})$ with respect to the verification query language $\mathcal{V}(\mathcal{QL})$.

**Lemma 2** *For every GOPS $\mathcal{G}$ and finite transformation $T$ for $\Phi$, $TS(\mathcal{G})$ and $T$-core$(\mathcal{G})$ are bisimilar.*

*Proof (sketch).* From the definition of finite transformation, it follows that the outcoming edges of two states $s, s'$ such that $\tau_\pi(\pi_s) = \tau_\pi(\pi_{s'})$ are the same, and since $s$ and $s'$ are locally bisimilar, every pair of corresponding successor states of $s$ and $s'$ are locally bisimilar as well. □

**Theorem 4** *Suppose $\langle \tau_\pi, \tau_\Phi \rangle$ is a finite transformation for $\Phi$ such that both $\tau_\pi$ and $\tau_\Phi$ are decidable. Then, answering verification queries over a GOPS $\mathcal{G}$ is decidable.*

*Proof (sketch).* First, it can be shown that the hypotheses and Lemma 1 imply that the $T$-core of $\mathcal{G}$ can be computed in a finite amount of time. Then, by Lemma 2 and since the evaluation of a verification query over a finite GOPS graph is decidable, the thesis follows. □

We now prove that the conflict resolution function corresponding to the `rif:forwardChaining` strategy (without the definition of a tie-break rule) admits a finite transformation, under the condition that the number of different states of the ontology is finite. Given an ontology $\mathcal{O}$, we define the update-closure of $\mathcal{O}$ as the set $\mathcal{C}$ inductively defined as follows: (i) $\mathcal{O} \in \mathcal{C}$; (ii) if $\mathcal{O}' \in \mathcal{C}$ and the formula $\phi$ is an ontology update using the constants occurring in $\mathcal{O}$, then $TELL(\mathcal{O}', \phi) \in \mathcal{C}$. We say that *updates have a finite evolution* if, for every ontology $\mathcal{O}$, the update-closure of $\mathcal{O}$ contains a finite number of locally-bisimilar equivalence classes.

**Theorem 5** *Suppose $\Phi$ corresponds to the `rif:forwardChaining` conflict resolution strategy, and suppose that $ASK$ is decidable, $TELL$ is decidable, and updates have a finite evolution. Then, answering verification queries over GOPSs is decidable.*

*Proof (sketch).* The definition of the conflict resolution strategy `rif:forwardChaining` implies that the corresponding conflict resolution function $\Phi$ actually uses a small amount of the information on the previous history of the system. This property, together with the hypothesis, allows for defining a function $\tau_\pi$ and a function $\tau_\Phi$ which satisfy the hypotheses of Theorem 4. Consequently, the thesis follows. □

[3] More precisely, either the problem of establishing whether $ASK(\phi, \mathcal{O}) \neq \emptyset$ for a Boolean ontology query $\phi$ is undecidable, or, given two ontologies $\mathcal{O}, \mathcal{O}'$ and an ontology update $\alpha$, establishing whether $\mathcal{O}' = TELL(\mathcal{O}, \alpha)$ is undecidable.

We conclude by considering the case when the ontology is a relational database without integrity constraints. More precisely, we call DB-GOPSs the GOPSs defined based on the following assumptions:

- the ontology language is the language of ground atoms: that is, ontologies correspond to databases (sets of facts);
- the ontology update language consists of assert and retract actions of single ground atoms;
- the ontology query language consists of domain-independent FOL queries (i.e., a subclass of SQL queries);
- the $ASK$ function evaluates queries according to the standard semantics of domain-independent FOL queries in relational databases (i.e., a database is considered as an interpretation over which the FOL queries are evaluated according to the standard FOL semantics);
- the $TELL$ function corresponds to the syntactic additions and deletions of facts in the database.

Now, it is immediate to verify that, in DB-GOPSs, updates have a finite evolution.

**Corollary 1** *Answer verification queries over DB-GOPSs under the* `rif:forwardChaining` *conflict resolution strategy is decidable.*

## Light-weight GOPS

In this section we present Lite-GOPSs, a special class of GOPSs is based on three light-weight ontology languages: the *DL-Lite*$_\mathcal{A}$ ontology specification language, the *EQL-Lite(UCQ)* ontology query language, and the $\mathcal{UL}$*-Lite* ontology update language. We show that Lite-GOPSs enjoy nice computational properties.

### Light-weight ontologies, queries and updates

We start by recalling the DL *DL-Lite*$_\mathcal{A}$ (Poggi et al. 2008), the ontology query language *EQL-Lite(UCQ)* (Calvanese et al. 2007a), and the update operators for DL ontologies defined in (Lenzerini and Savo 2011).

**The DL *DL-Lite*$_\mathcal{A}$** The DL *DL-Lite*$_\mathcal{A}$ (Poggi et al. 2008) is a member of the *DL-Lite* family of tractable Description Logics. *DL-Lite*$_\mathcal{A}$ distinguishes concepts, which denote sets of objects, from value-domains, which denote sets of (data) values, and roles, which denote relations between objects, from attributes, which denote binary relations between objects and values. The semantics of *DL-Lite*$_\mathcal{A}$ is given in terms of first-order logic interpretations, and the notions of interpretation of *DL-Lite*$_\mathcal{A}$ constructs and assertions, model of an ontology, and entailment are given in the standard way.

For our purposes, a crucial aspect is that all the standard reasoning tasks are tractable in *DL-Lite*$_\mathcal{A}$: in particular, answering unions of conjunctive queries (UCQs) over a *DL-Lite*$_\mathcal{A}$ ontology is tractable with respect to the size of the ontology. For further details on *DL-Lite*$_\mathcal{A}$ we refer the reader to (Poggi et al. 2008).

**The ontology query language *EQL-Lite(UCQ)*** Informally, *EQL-Lite(UCQ)* is the FOL query language with equality whose atoms are epistemic formulas of the form $\mathbf{K}q$, where $q$ is a UCQ. Like FOL queries, such a language allows for posing very expressive queries with arbitrary negation and quantification: the difference lies in the presence of the epistemic operator, whose interpretation (based on a minimal knowledge semantics) allows for preserving the decidability of reasoning.

Formally, an *EQL-Lite(UCQ)* query is a possibly open formula built according to the following syntax:

$$\phi \ ::= \ \mathbf{K}q \mid x_1 = x_2 \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists x.\phi$$

where $q$ is a UCQ. Formulas without occurrences of $\mathbf{K}$ are said to be *objective*.

In *EQL-Lite(UCQ)*, the modal operator is used to formalize the epistemic state of the DL ontology, according to the minimal knowledge semantics (see later). Informally, the formula $\mathbf{K}\phi$ should be read as "$\phi$ is known to hold (by the ontology)".

We interpret DL ontologies on interpretations sharing the *same infinite countable domain* $\Delta$, and we assume that the language includes an infinitely countable set of disjoint constants corresponding to elements of $\Delta$, known as *standard names* (Levesque and Lakemeyer 2001). A *world* is a FOL interpretation over $\Delta$. An *epistemic interpretation* is a pair $E, w$, where $E$ is a (possibly infinite) set of worlds, and $w$ is a world in $E$. The definition of when a sentence (i.e., a closed formula) $\phi$ is satisfied in an epistemic interpretation $E, w$, written $E, w \models \phi$, is the usual one in modal logic S5 (see (Calvanese et al. 2007a)). Among the various epistemic interpretations, only the ones that represent a *minimal epistemic state* of the DL ontology, i.e., the state in which the ontology has minimal knowledge, are considered. Formally: let $\mathcal{O}$ be a DL ontology (TBox and ABox), and let $Mod(\mathcal{O})$ be the set of all FOL-interpretations that are models of $\mathcal{O}$. Then a $\mathcal{O}$*-EQL-interpretation* is an epistemic interpretation $E, w$ for which $E = Mod(\mathcal{O})$.

A sentence $\phi$ is *EQL-entailed* by $\mathcal{O}$, written $\mathcal{O} \models_{EQL} \phi$, if for every $\mathcal{O}$-EQL-interpretation $E, w$ we have $E, w \models \phi$. Observe that for objective formulas such a definition becomes the standard one, namely $w \models \phi$ for all $w \in Mod(\mathcal{O})$, denoted by $\mathcal{O} \models \phi$.

Let $\phi$ be an *EQL*-query with free variables $\vec{x}$, where the arity of $\vec{x}$ is $n \geq 0$, and is called the arity of $\phi$. We will use the notation $\phi[\vec{c}]$ to denote $\phi^{\vec{x}}_{\vec{c}}$ (i.e., the formula obtained from $\phi$ by substituting each free occurrence of the variable $x_i$ in $\vec{x}$ with the constant $c_i$ in $\vec{c}$, where obviously $\vec{x}$ and $\vec{c}$ must have the same arity). The *certain answers* to a query $\phi[\vec{x}]$ over an ontology $\mathcal{O}$ are defined as follows:

$$ans_{EQL}(\phi, \mathcal{O}) = \{\vec{c} \in \Delta \times \cdots \times \Delta \mid \mathcal{O} \models_{EQL} \phi[\vec{c}]\}$$

As usual in the literature on query answering over DL ontologies (e.g., (Lutz 2008)), in the following we call *data complexity* the complexity measured with respect to the size of the ABox.

The following property, shown in (Calvanese et al. 2007a), states that answering *EQL-Lite(UCQ)* queries in *DL-Lite*$_\mathcal{A}$ is tractable with respect to data complexity.

**Theorem 6** *Answering domain independent EQL-Lite(UCQ) queries in DL-Lite is in PTIME (in particular, in $AC^0$) with respect to data complexity.*

**Instance-level DL ontology update semantics** We now briefly recall the update operators for the evolution of DL ontologies presented in (Lenzerini and Savo 2011). This approach has three key aspects. First, it concerns the so-called *instance-level updates* to DL ontologies (De Giacomo et al. 2009). In fact, this approach only considers update operations corresponding to the assertion or retraction of sets of ABox assertions, i.e., only the extensional component (ABox) of the DL ontology is updated, while the intensional component (TBox) is unchanged. Second, it proposes a semantics for ontology updates which is closed with respect to the DL language. This implies that, given a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$, the result of any update operation can always be expressed in terms of a *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}'$. Finally, the proposed semantics has nice computational properties.

In the following, we assume that the initial ontology $\mathcal{O} = \langle T, \mathcal{A}\rangle$ is a satisfiable *DL-Lite*$_\mathcal{A}$ ontology. In other words, we do not consider the evolution of unsatisfiable ontologies. Also, we denote by $\mathsf{cl}_\mathcal{T}(\mathcal{A})$ the $\mathcal{T}$-closure of $\mathcal{A}$, i.e., the set of atomic ABox assertions that are entailed by $\langle \mathcal{T}, \mathcal{A}\rangle$.

Let $\mathcal{A}', F$ be ABoxes. Then, $\mathcal{A}'$ *accomplishes the insertion of* $F$ *into* $\langle \mathcal{T}, \mathcal{A}\rangle$ if $\mathcal{A}'$ is $\mathcal{T}$-consistent, and $\langle \mathcal{T}, \mathcal{A}'\rangle \models F$ (i.e., $F \subseteq \mathsf{cl}_\mathcal{T}(\mathcal{A}')$). Similarly, $\mathcal{A}'$ *accomplishes the deletion of* $F$ *from* $\langle \mathcal{T}, \mathcal{A}\rangle$ if $\mathcal{A}'$ is $\mathcal{T}$-consistent, and $\langle \mathcal{T}, \mathcal{A}'\rangle \not\models F$ (i.e., $F \not\subseteq \mathsf{cl}_\mathcal{T}(\mathcal{A}')$). Then, we say that $\mathcal{A}'$ accomplishes the insertion (deletion) of $F$ into (from) $\langle \mathcal{T}, \mathcal{A}\rangle$ *minimally* if $\mathcal{A}'$ accomplishes the insertion (deletion) of $F$ into (from) $\langle \mathcal{T}, \mathcal{A}\rangle$, and there is no $\mathcal{A}''$ that accomplishes the insertion (deletion) of $F$ into (from) $\langle \mathcal{T}, \mathcal{A}\rangle$, and has fewer changes than $\mathcal{A}'$ with respect to $\langle \mathcal{T}, \mathcal{A}\rangle$.[4]

Let $\mathcal{U} = \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be the set of all ABoxes accomplishing the insertion (deletion) of $F$ into (from) $\langle \mathcal{T}, \mathcal{A}\rangle$ minimally, and let $\mathcal{A}'$ be an ABox. Then, $\langle \mathcal{T}, \mathcal{A}'\rangle$ is the result of changing $\langle \mathcal{T}, \mathcal{A}\rangle$ with the insertion (deletion) of $F$ if (1) $\mathcal{U}$ is empty, and $\langle \mathcal{T}, \mathsf{cl}_\mathcal{T}(\mathcal{A}')\rangle = \langle \mathcal{T}, \mathsf{cl}_\mathcal{T}(\mathcal{A})\rangle$, or (2) $\mathcal{U}$ is nonempty, and $\langle \mathcal{T}, \mathsf{cl}_\mathcal{T}(\mathcal{A}')\rangle = \langle \mathcal{T}, \bigcap_{1\leq i\leq n} \mathsf{cl}_\mathcal{T}(\mathcal{A}_i)\rangle$.

It is immediate to verify that, up to logical equivalence, the result of changing $\langle \mathcal{T}, \mathcal{A}\rangle$ with the insertion or the deletion of $F$ is unique. The result of changing $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ with the insertion of $F$ according to the above semantics will be denoted by *LS-assert*$(\mathcal{O}, F)$. Moreover, the result of changing $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ with the deletion of $F$ according to the above semantics will be denoted by *LS-retract*$(\mathcal{O}, F)$. Notice that, in the case where $F$ is $\mathcal{T}$-inconsistent, the result of changing $\langle \mathcal{T}, \mathcal{A}\rangle$ with both the insertion and the deletion of $F$ is logically equivalent to $\langle \mathcal{T}, \mathcal{A}\rangle$ itself.

The following theorem (shown in (Lenzerini and Savo 2011)) summarizes the computational properties of the above update operators.

**Theorem 7** *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ be a DL-Lite*$_\mathcal{A}$ *ontology, and let $\mathcal{A}'$ be a DL-Lite*$_\mathcal{A}$ *ABox. Then:*

1. *LS-assert$(\mathcal{O}, \mathcal{A}')$ can be computed in PTIME.*
2. *LS-retract$(\mathcal{O}, \mathcal{A}')$ can be computed in PTIME.*

[4] We refer to (Lenzerini and Savo 2011) for further details on this definition.

We point out that the above theorem is a very interesting result, since it states that computing updates according to the semantics above illustrated can be done in polynomial time, although such a notion of updates does not boil down to purely syntactic insertion/deletion operations.

From now on, we call $\mathcal{UL}$*-Lite* the instance-level ontology update language which consists of every formula of the form *Assert*$(F)$ or *Retract*$(F)$, where $F$ is a finite set of atomic ABox assertions (i.e., a conjunction of ground atoms).

## Lite-GOPS

We are finally ready to define Lite-GOPSs. A *Lite-GOPS* is a GOPS as in Definition 1 under the following assumptions:

- the ontology language $\mathcal{OL}$ is *DL-Lite*$_\mathcal{A}$;
- the ontology query language $\mathcal{QL}$ is *EQL-Lite(UCQ)*;
- the ontology update language $\mathcal{UL}$ is $\mathcal{UL}$*-Lite*;
- the function $ASK$ which provides the semantics for ontology queries corresponds to the semantics for *EQL* queries described above. More precisely, for every *DL-Lite*$_\mathcal{A}$ ontology $\mathcal{O}$, and for every *EQL-Lite(UCQ)* query $\phi$, $ASK(\phi, \mathcal{O}) = ans_{EQL}(\phi, \mathcal{O})$;
- the function $TELL$ which provides the semantics for ontology updates corresponds to the semantics for ontology update described in the previous section. More precisely:
  - for every ontology update $\alpha$ of the form *Assert*$(\Gamma)$, $TELL(\mathcal{O}, \alpha) =$ *LS-assert*$(\mathcal{O}, \alpha)$;
  - for every ontology update $\alpha$ of the form *Retract*$(\Gamma)$, $TELL(\mathcal{O}, \alpha) =$ *LS-retract*$(\mathcal{O}, \alpha)$;
- the conflict resolution strategy $\Phi$ admits a finite transformation.

We now show that answering verification queries over Lite-GOPSs is decidable. We also identify some subclasses of queries of the verification query language which can be answered in polynomial time when evaluated over Lite-GOPSs. It is immediate to verify that the chosen semantics of ontology updates and the adoption of the *DL-Lite*$_\mathcal{A}$ language imply that updates have a finite evolution in Lite-GOPSs. We have the following decidability and complexity results.

**Theorem 8** *Answering verification queries over Lite-GOPSs is decidable.*

*Proof.* First, it is immediate to verify that updates have a finite evolution in Lite-GOPSs. Thus, from Theorem 6, Theorem 7, and Theorem 5, the thesis follows. □

**Theorem 9** *Answering verification queries over Lite-GOPSs under the* `rif:forwardChaining` *conflict resolution strategy is in EXPTIME with respect to data complexity.*

*Proof (sketch).* The thesis follows from Theorem 1, Theorem 6 and Theorem 7. □

We now show a tractability result for reasoning over Lite-GOPSs. In particular, we prove that reasoning over GOPSs is tractable for verification queries without fixpoints. We say

that a verification query is *simple* if it does not contain fixpoint operators.

**Theorem 10** *Answering simple verification queries over Lite-GOPSs is in PTIME with respect to data complexity.*

*Proof (sketch).* The key property on which the proof is based is that it is sufficient to build a small (polynomial) part of the transition system. In particular, if $k$ is the size of the simple verification query (actually $k$ should represent the maximum nesting level of the modalities), then it is sufficient to build the paths of the transition system that start at the initial state and have a length less than or equal to $k$.

Now, the number of outcoming edges from a state for the same production rule $p$ is polynomial with respect to data complexity, since there is only a polynomial number of ground substitutions for the free variables of $p$, consequently the number of outcoming edges from a state is polynomial. This in turn implies that the number of the above paths of length $\leq k$ is polynomial with respect to data complexity (since $k$ does not depend on the size of the ABox), therefore this portion of the transition system can be built in polynomial time.

Thus, from Theorem 1 it follows that the evaluation of the query over such a polynomial model is polynomial with respect to data complexity, which implies the thesis. □

## Conclusions

In this paper we have presented generalized ontology-based production systems (GOPSs), which constitute a very general framework for the combination of ontologies and production rules. The GOPS approach is based on a functional specification of ontologies, which views ontologies as knowledge bases which can be accessed through a query service and an update service. In this way, the semantics of the execution of production rules over ontologies is straightforward, and fully relies on the semantics of queries and updates over ontologies. Then, we have defined an expressive language for formalizing verification tasks over GOPS specifications, and have shown that typical static analysis tasks can easily be expressed through such a language. Moreover, we have studied the computational properties of reasoning in the framework of GOPSs, providing very general sufficient conditions for undecidability and decidability of reasoning. Finally, we have analyzed a specific combination of ontologies and production rules, called Lite-GOPSs. We have established decidability and complexity results for reasoning in such a class of GOPS, showing that Lite-GOPSs constitute a very good trade-off between the complexity of reasoning and the expressive power of the ontology component of the GOPSs.

This approach can be further extended in several directions. First of all, in a way analogous to Lite-GOPSs, other notable specific combinations of ontologies and production rules can be defined and studied within the GOPS framework. Then, it would be very interesting to further extend the GOPS framework, by adding other forms of production rules beyond those considered in the RIF-PRD specification. Finally, it would also be interesting to extend this approach to more powerful verification languages. For instance, it would be very easy to extend $\mathcal{V}(\mathcal{QL})$ to allow for the presence of free variables.

## Acknowledgments

We thank the anonymous reviewers for their precious comments. The work presented in this paper has been funded by the European project ONTORULE – Ontologies meet business rules (ICT-231875).

## References

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Baral, C., and Lobo, J. 1995. Characterizing production systems using logic programming and situation calculus. `http://www.public.asu.edu/~cbaral/papers/char-prod-systems.ps`.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007a. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, 274–279.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007b. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *Journal of Automated Reasoning* 39(3):385–429.

Calvanese, D.; Kharlamov, E.; Nutt, W.; and Zheleznyakov, D. 2010. Evolution of *DL-Lite* knowledge bases. In *Proceedings of the Ninth International Semantic Web Conference (ISWC 2010)*, volume 6496 of *Lecture Notes in Computer Science*, 112–128. Springer.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proceedings of the Seventeenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)*, 149–158.

Cangialosi, P.; De Giacomo, G.; De Masellis, R.; and Rosati, R. 2010. Conjunctive artifact-centric services. In *Proceedings of the Eighth International Joint Conference on Service Oriented Computing (ICSOC 2010)*, volume 6470 of *Lecture Notes in Computer Science*, 318–333. Springer.

Damasio, C. V.; Alferes, J. J.; and Leite, J. 2010. Declarative semantics for the rule interchange format production rule dialect. In *Proceedings of the Ninth International Semantic Web Conference (ISWC 2010)*, 798–813.

de Bruijn, J., and Rezk, M. 2009. A logic based approach to the static analysis of production systems. In *Proceedings of the Third International Conference on Web Reasoning and Rule Systems (RR 2009)*, 254–268.

De Giacomo, G.; Lenzerini, M.; Poggi, A.; and Rosati, R. 2009. On instance-level update and erasure in description logic ontologies. *Journal of Logic and Computation, Special Issue on Ontology Dynamics* 19(5):745–770.

de Sainte Marie, C.; Hallmark, G.; and Paschke, A. (editors) 2010. RIF production rule dialect. W3C Recommendation, http://www.w3.org/TR/rif-prd/.

Emerson, E. A. 1996. Model checking and the mu-calculus. In *Descriptive Complexity and Finite Models*, 185–214.

Glimm, B. 2011. Using sparql with rdfs and owl entailment. In *Reasoning Web 2011*, volume 6848 of *Lecture Notes in Computer Science*, 137–201.

Kowalski, R., and Sadri, F. 2009. Integrating logic programming and production systems in abductive logic programming agents. In *Proceedings of the 3rd International Conference on Web Reasoning and Rule Systems*, RR '09, 1–23. Springer-Verlag.

Lenzerini, M., and Savo, D. F. 2011. On the evolution of the instance level of *DL-Lite* knowledge bases. In *Proceedings of the Twentyfourth International Workshop on Description Logic (DL 2011)*, volume 745 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/.

Levesque, H. J., and Lakemeyer, G. 2001. *The Logic of Knowledge Bases*. The MIT Press.

Levesque, H. J. 1984. Foundations of a functional approach to knowledge representation. *Artificial Intelligence* 23:155–212.

Levy, A. Y., and Rousset, M.-C. 1998. Combining Horn rules and description logics in CARIN. *Artificial Intelligence* 104(1–2):165–209.

Liu, H.; Lutz, C.; Milicic, M.; and Wolter, F. 2006. Updating description logic ABoxes. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, 46–56.

Liu, H.; Lutz, C.; Milicic, M.; and Wolter, F. 2011. Foundations of instance level updates in expressive description logics. *Artif. Intell.* 175(18):2170–2197.

Lunardhi, A. D., and Passino, K. M. 1995. Verification of qualitative properties of rule-based expert systems. *Applied Artificial Intelligence* 9(6):587–621.

Lutz, C. 2007. Inverse roles make conjunctive queries hard. In *Proceedings of the Twentieth International Workshop on Description Logic (DL 2007)*, volume 250 of *CEUR Electronic Workshop Proceedings,* http://ceur-ws.org/, 100–111.

Lutz, C. 2008. The complexity of conjunctive query answering in expressive description logics. In *Proceedings of the Fourth International Joint Conference on Automated Reasoning (IJCAR 2008)*, volume 5195 of *Lecture Notes in Artificial Intelligence*, 179–193. Springer.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *Journal on Data Semantics* X:133–173.

Raschid, L. 1994. A semantics for a class of stratified production system programs. *J. Log. Program.* 21(1):31–57.

Rezk, M., and Nutt, W. 2011. Combining production systems and ontologies. In *Proceedings of the Fifth International Conference on Web Reasoning and Rule Systems (RR 2011)*, 287–293.

Rosati, R. 2007. The limits of querying ontologies. In *Proceedings of the Eleventh International Conference on Database Theory (ICDT 2007)*, volume 4353 of *Lecture Notes in Computer Science*, 164–178. Springer.

Sirin, E., and Parsia, B. 2007. SPARQL-DL: SPARQL query for OWL-DL. In *Proc. OWLED 2007*.