Actions and Programs over Description Logic Ontologies

Diego Calvanese[1], Giuseppe De Giacomo[2], Maurizio Lenzerini[2], Riccardo Rosati[2]

[1] Faculty of Computer Science
Free University of Bozen-Bolzano
Piazza Domenicani 3,
I-39100 Bolzano, Italy
calvanese@inf.unibz.it

[2] Dip. di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113,
I-00198 Roma, Italy
lastname@dis.uniroma1.it

Abstract. We aim at representing and reasoning about actions and (high level) programs over ontologies expressed in Description Logics. This is a critical issue that has resisted good solutions for a long time. In particular, while well-developed theories of actions and high-level programs exist in AI, e.g., the ones based on SitCalc, these theories do not apply smoothly to Description Logic ontologies, due to the profoundly non-definitorial nature of such ontologies (cf. cyclic TBoxes). Here we propose a radical solution: we assume a functional view of ontologies and see them as systems that allow for two kinds of operations: ASK, which returns the (certain) answer to a query, and TELL, which produces a new ontology as a result of the application of an atomic action. We base atomic actions on instance level update and instance level erasure on the ontology. Building on this functional view, we introduce Golog/ConGolog-like high-level programs on ontologies. This paper demonstrates the effectiveness of the approach in general, and presents the following specific results: we characterize the notion of single-step executability of such programs, devise methods for reasoning about sequences of actions, and present (nice) complexity results in the case where the ontology is expressed in DL-Lite.

1 Introduction

Description Logics (DLs) [1] are generally advocated as the right tool to express ontologies, and this belief is one of the cornerstones of the Semantic Web [31, 15]. Notably, semantic web services [22] constitute another cornerstone of the Semantic Web. These are essentially high-level descriptions of computations that abstract from the technological issues of the actual programs that realize them. An obvious concern is to combine in some way the static descriptions of the information provided by ontologies with the dynamic descriptions of the computations provided by semantic web services. Interestingly, such a critical issue has resisted good solutions for a long time. Indeed even big efforts such as OWL-S [22] have not really succeeded.

In AI, the importance of combining static and dynamic knowledge has been recognized early [23, 24]. By now, well developed theories of actions and high level programs exist in AI, e.g., the ones based on Reiter's variant of SitCalc [28]. Note that high-level programs [19, 10] share with semantic web services the emphasis on abstracting from

the technological issues of actual programs, and are indeed abstract descriptions of computations over a domain of interest.

Unfortunately, these theories do not apply smoothly to DL ontologies, due to the profoundly non-definitorial nature of such ontologies. Indeed, concepts and roles expressions in a DL do not provide *definitions* of concepts and roles in general, but usually only describe interrelations between them (cf. cyclic TBox interpreted according to the usual descriptive semantics [1]). Such non-definitorial nature of DL ontologies makes them one of the most difficult kinds of domain descriptions for reasoning about actions [2, 20].

Here we propose a radical solution: we assume a functional view [17] of ontologies and see them as systems that allow for two kinds of operations: ASK, which returns the (certain) answer to a query, and TELL, which produces a new ontology as a result of the application of an atomic action. Observe that this approach, whose origins come from [7, 13, 25, 32], has some subtle limitations, due to the fact that we lose the possibility of distinguishing between "knowledge" and "truth" as pointed out in [30]. On the other hand, it has a major advantage: it decouples reasoning on the static knowledge from the one on the dynamics of the computations over such knowledge. As a result, we gain the ability of lifting to DLs many of the results developed in reasoning about actions in the years.

We demonstrate such an approach in this paper. Specifically, we base atomic actions used by the TELL operation on instance level update and instance level erasure on the ontology [8, 9]. Building on this functional view, we introduce Golog/ConGolog-like high level programs over ontologies. We characterize the notion of single-step executability of such programs, devise methods for reasoning about sequences of actions, and present (nice) complexity results in the case where the ontology is expressed in DL-Lite. We stress that this paper is really an illustration of what a functional view on ontologies can bring about in combining static and dynamic aspects in the context of DL ontologies, and that many extensions of this work can be investigated (we will mention some of them in the conclusions).

2 Preliminaries

DL ontologies. Description Logics (DLs) [1] are knowledge representation formalisms that are tailored for representing the domain of interest in terms of *concepts* (or classes), which denote sets of objects, and *roles* (or relations), which denote denote binary relations between objects. DLs *ontologies* (aka knowledge bases) are formed by two distinct parts: the so-called *TBox*, which represents the *intensional level* of the ontology, and contains an intensional description of the domain of interest; and the so-called *ABox*, which represents the *instance level* of the ontology, and contains extensional information.

We give the semantics of a DL ontology in terms of interpretations over a fixed infinite domain Δ of objects. We assume to have a constant for each object in Δ denoting exactly that object. In this way we blur the distinction between constants and objects, so that we can use them interchangeably (with a little abuse of notation) without causing confusion (cf. standard names [18]).

An interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ consists of a first order structure over Δ, where $\cdot^{\mathcal{I}}$ is the interpretation function, i.e., a function mapping each concept to a subset of Δ and each role to a subset of $\Delta \times \Delta$. We say that $\mathcal{I}$ is a *model of a (TBox or ABox) assertion* α, or also that $\mathcal{I}$ *satisfies* α, if α is true in $\mathcal{I}$. We say that $\mathcal{I}$ is a *model of the ontology* $s = \langle \mathcal{T}, \mathcal{A} \rangle$, or also that $\mathcal{I}$ *satisfies the ontology* s, if $\mathcal{I}$ is a model of all the assertions in $\mathcal{T}$ and $\mathcal{A}$. Given a set $\mathcal{S}$ of (TBox or ABox) assertions, we denote as $Mod(\mathcal{S})$ the set of interpretations that are models of all assertions in $\mathcal{S}$. In particular, the set of *models of an ontology* s, denoted as $Mod(s)$, is the set of models of all assertions in $\mathcal{T}$ and $\mathcal{A}$, i.e., $Mod(s) = Mod(\langle \mathcal{T}, \mathcal{A} \rangle) = Mod(\mathcal{T} \cup \mathcal{A})$. An ontology s is *consistent* if $Mod(s) \neq \emptyset$, i.e., it has at least one model. We say that an ontology s *logically implies* an expression α (e.g., an assertion, an instantiated union of conjunctive queries, etc.), written $s \models \alpha$, if for every interpretation $\mathcal{I} \in Mod(s)$, we have $\mathcal{I} \in Mod(\alpha)$, i.e., all the models of s are also models of α. When dealing with queries, we are interested in *query answering* (for CQs and UCQs): given an ontology s and a query $q(\boldsymbol{x})$ over s, return the *certain answers* to $q(\boldsymbol{x})$ over s, i.e., all tuples $\boldsymbol{t}$ of elements of $\Delta^{\mathcal{I}}$ such that, when substituted to $\boldsymbol{x}$ in $q(\boldsymbol{x})$, we have that $s \models q(\boldsymbol{t})$.

***DL-Lite*$_F$.** In this paper, we focus on a particular DL, namely *DL-Lite*$_F$, belonging to the *DL-Lite* family [4, 5] of DLs, which are tailored towards capturing conceptual modeling constructs (such as those typical of UML Class Diagrams or Entity-Relationship Diagrams), while keeping reasoning, including conjunctive query answering, tractable and first-order reducible (i.e., LOGSPACE in data complexity). In *DL-Lite*$_F$, which is the logic originating the whole *DL-Lite* family, concepts are defined as follows:

$$B \ ::= \ A \mid \exists R \qquad\qquad C \ ::= \ B \mid \neg B \qquad\qquad R \ ::= \ P \mid P^{-}$$

where A denotes an atomic concept, P an atomic role, B a *basic concept*, and C a general concept. A basic concept can be either an atomic concept, a concept of the form $\exists P$, i.e. the standard DL construct of unqualified existential quantification on roles, or a concept of the form $\exists P^{-}$, which involves *inverse roles*. A *DL-Lite*$_F$ TBox is a finite set of universal assertions of the form

$$B \sqsubseteq C \qquad \textit{inclusion assertion}$$

$$(\mathsf{funct}\ R) \qquad \textit{functionality assertion}$$

Inclusion assertions are interpreted as usual in DLs, while functionality assertions express the (global) functionality of atomic roles or of inverses of atomic roles.

A *DL-Lite*$_F$ ABox is a finite set of membership assertions of the form, $B(a)$ or $R(a, b)$, which state, respectively, that the object a is an instance of the basic concept B, and that the pair of objects (a, b) is an instance of the role R.

Query answering of *EQL-Lite*(UCQ) queries over *DL-Lite*$_F$ ontologies. As query language, here we consider *EQL-Lite*(UCQ) [6]. This is essentially formed by full (domain-independent) FOL query expressions built on top of atoms that have the form $\mathbf{K}\alpha$, where α is a union of conjunctive queries[1]. The operator $\mathbf{K}$ is a minimal knowl-

[1] For queries consisting of only one atom $\mathbf{K}\alpha$, the $\mathbf{K}$ operator is omitted.

edge operator [17, 27, 18], which is used to formalize the epistemic state of the ontology. Informally, the formula $\mathbf{K}\alpha$ is read as "α is known to hold" or "α is logically implied by the ontology". Answering *EQL-Lite*(UCQ) queries over *DL-Lite*$_F$ ontologies is LOGSPACE, and, notably, can be reduced to evaluating (pure) FOL queries over the ABox, when considered as a database. We refer to [6] for more details.

DL instance-level updates and erasure. Following the work in [8, 9], we adopt Winslett's notion of *update* [33, 34] and its counterpart, defined in [16], as the notion of *erasure*. However, we refine such notions to take into account that we are interested in studying changes at the instance level, while we insist that the intensional level of the ontology is considered stable and hence remains invariant. Intuitively, the result of updating (resp., erasing) an ontology s with a finite set $\mathcal{F}$ of membership assertions is a new ontology that logically implies (resp., does not logically imply) all assertions in $\mathcal{F}$, and whose set of models minimally differs from the set of models of s. Unfortunately, as shown in [21, 8, 9], in general the result of update and erasure cannot be expressed in the same language as the original ontology.[2] Hence, we focus on maximally approximated update and erasure. The *maximally approximated update (erasure)* is an ontology in the same language as the original one and whose models are the models of the update (erasure) which minimally differ from the models of the original ontology.

Below, when we talk about update and erasure, we always consider their approximated versions. More precisely, let $s = \langle \mathcal{T}, \mathcal{A} \rangle$ be an ontology and $\mathcal{F}$ a finite set of membership assertions such that $Mod(\mathcal{T} \cup \mathcal{F}) \neq \emptyset$: we denote by $s \circ_{\mathcal{T}} \mathcal{F}$ the (maximally approximated) *update of s with $\mathcal{F}$*. Similarly, assuming $Mod(\mathcal{T} \cup \neg\mathcal{F}) \neq \emptyset$, where $\neg\mathcal{F}$ denotes the set of membership assertions $\{\neg F_i \mid F_i \in \mathcal{F}\}$[3]: we denote by $s \bullet_{\mathcal{T}} \mathcal{F}$ the (maximally approximated) *erasure of s with $\mathcal{F}$*. Computing both (maximally approximated) update and erasure of a *DL-Lite*$_F$ ontology s with a set $\mathcal{F}$ of membership assertions is polynomial in the sizes of s and $\mathcal{F}$ [9].

3 Atomic actions

Under a functional view [17], ontologies are seen as systems that are able to perform two basic kinds of operations, namely ASK and TELL operations (cf. [17, 18]):

- ASK: given an ontology and a *query* (in the query language recognized by the ontology), returns a *finite* set of tuples of objects (constituting the answers to the query over the ontology).
- TELL: given an ontology and an *atomic action*, returns a new ontology resulting from executing the action, if the action is executable wrt the given ontology.

[2] The form of the *DL-Lite*$_F$ ABox considered above is that of the original proposal in [4], and is a restriction w.r.t. the one studied in [8], where instance-level updates in DLs of the *DL-Lite* family were first introduced. Specifically, here we do not allow for negation and "variables" in the membership assertions, cf. [8]. With this restriction *DL-Lite*$_F$ becomes akin to the vast majority of DLs, see [21], in that the result of updates and erasure is not expressible as a new *DL-Lite*$_F$ ABox, thus requiring approximation [9].

[3] Observe that $\neg F_i$ might not be in the language of ABoxes, see [9].

In this paper, we focus on *DL-Lite*$_F$ [4, 5] as ontology language, and *EQL-Lite*(UCQ) [6] as query language. Hence, we base ASK on certain answers to such queries. Specifically, we denote by $q(\boldsymbol{x})$ an (*EQL-Lite*(UCQ)) query with distinguished variables $\boldsymbol{x}$. We define $\text{ASK}(q(\boldsymbol{x}), s) = \{\boldsymbol{t} \mid s \models q(\boldsymbol{t})\}$, where s is an ontology and $\boldsymbol{t}$ is a tuple of constants of the same arity as $\boldsymbol{x}$. We denote by ϕ queries with no distinguished variables. Such queries are called *boolean* queries and return either *true* (i.e., the empty tuple) or *false* (i.e., no tuples at all).

As for TELL, we base atomic actions on instance level update and erasure [8, 9]. Specifically, we allow for *atomic actions* of the following form:

$$\textbf{update}\ L(\boldsymbol{x})\ \textbf{where}\ q(\boldsymbol{x})$$
$$\textbf{erase}\ L(\boldsymbol{x})\ \textbf{where}\ q(\boldsymbol{x})$$

where $q(\boldsymbol{x})$ stands for a query with $\boldsymbol{x}$ as distinguished variables and $L(\boldsymbol{x})$ stands for a set of membership assertions on constants and variables in $\boldsymbol{x}$. We define

$$\begin{aligned}
&\text{TELL}([\textbf{update}\ L(\boldsymbol{x})\ \textbf{where}\ q(\boldsymbol{x})], s) = s \circ_{\mathcal{T}} \textstyle\bigcup_{\boldsymbol{t} \in \text{ASK}(q(\boldsymbol{x}), s)} L(\boldsymbol{t}) \\
&\qquad \text{if } \mathit{Mod}(\mathcal{T} \cup \textstyle\bigcup_{\boldsymbol{t} \in \text{ASK}(q(\boldsymbol{x}), s)} L(\boldsymbol{t})) \neq \emptyset \\
&\text{TELL}([\textbf{erase}\ L(\boldsymbol{x})\ \textbf{where}\ q(\boldsymbol{x})], s) = s \bullet_{\mathcal{T}} \textstyle\bigcup_{\boldsymbol{t} \in \text{ASK}(q(\boldsymbol{x}), s)} L(\boldsymbol{t}) \\
&\qquad \text{if } \mathit{Mod}(\mathcal{T} \cup \neg \textstyle\bigcup_{\boldsymbol{t} \in \text{ASK}(q(\boldsymbol{x}), s)} L(\boldsymbol{t})) \neq \emptyset
\end{aligned}$$

If the conditions in the equivalences above are not satisfied, we say that the atomic action a is *not executable* in s. We extend ASK to expressions of the form $\text{ASK}([\mathit{executable}(a)], s)$, so as to be able to check executability of actions. Observe that the executability of actions as defined above can indeed be checked on the ontology.

Notice that both ASK and TELL for *DL-Lite*$_F$ defined above can be computed in polynomial time, considering the size of the query fixed [6, 9].

4 Programs

We now consider how atomic actions can be organized within a program. In particular, we focus on a variant of Golog [19, 10, 29] tailored to work on ontologies. Instead of situations, we consider ontologies, or, to be more precise, ontology states. We recall that when considering ontologies we assume the TBox to be invariant, so the only part of the ontology that can change as a result of an action (or a program) is the ABox.

While all constructs of the original Golog/ConGolog have a counterpart in our variant, here for brevity we concentrate on a core fragment only, namely:

a	atomic actions
ϵ	the empty sequence of actions
$\delta_1; \delta_2$	sequential composition
if ϕ **then** δ_1 **else** δ_2	if-then-else
while ϕ **do** δ	while
pick $q(\boldsymbol{x}).\delta[\boldsymbol{x}]$	pick

where a is an atomic instruction which corresponds to the execution of the atomic action a; ϵ is an empty sequence of instructions (needed for technical reasons) **if** ϕ **then** δ_1 **else** δ_2 and **while** ϕ **do** δ are the standard constructs for conditional choice and iteration, where the test condition is a boolean query (or an executability check) to be asked to the current ontology; finally **pick** $q(\boldsymbol{x}).\delta[\boldsymbol{x}]$ picks a tuple $\boldsymbol{t}$ in the answer to $q(\boldsymbol{x})$, instantiates the rest of the program δ by substituting $\boldsymbol{x}$ with $\boldsymbol{t}$ and executes δ. The latter construct is a variant of the pick construct in Golog: the main difference being that $\boldsymbol{t}$ is bounded by a query to the ontology. Also, while in Golog such a choice is nondeterministic, here we think of it as possibly made *interactively*, see below.

The general approach we follow is the *structural operational semantics* approach based on defining a single step of program execution [26, 10]. This single-step semantics is often called *transition semantics* or *computation semantics*. Namely, to formally define the semantics of our programs we make use of a *transition relation*, named $Trans$, and denoted by "$\longrightarrow$":

$$(\delta, s) \xrightarrow{a} (\delta', s')$$

where δ is a program, s is an ontology in which the program is executed, a is the executed atomic action, s' is the ontology obtained by executing a in δ and δ' is what remains to be executed of δ after having executed a.

We also make use of a *final predicate*, named $Final$, and denoted by "$^{\surd}$":

$$(\delta, s)^{\surd}$$

where δ is a program that can be considered (successfully) terminated with the ontology s.

Such a relation and predicate can be defined inductively in a standard way, using the so called *transition (structural) rules*. The structural rules for defining the transition relation and the final predicate are given in Figure 1 and Figure 2 respectively. All structural rules have the following schema:

$$\frac{\text{CONSEQUENT}}{\text{ANTECEDENT}} \text{ if SIDE-CONDITION}$$

$$act: \quad \frac{(a,s) \xrightarrow{a} (\epsilon, \text{TELL}(a,s))}{\textit{true}} \quad \text{if } a \text{ is executable in } s$$

$$seq: \quad \frac{(\delta_1;\delta_2,\ s) \xrightarrow{a} (\delta_1';\delta_2,s')}{(\delta_1,s) \xrightarrow{a} (\delta_1';s')} \qquad \frac{(\delta_1;\delta_2,\ s) \xrightarrow{a} (\delta_2',s')}{(\delta_2,s) \xrightarrow{a} (\delta_2';s')} \quad \text{if } (\delta_1,s)^{\surd}$$

$$if: \quad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \xrightarrow{a} (\delta_1', s')}{(\delta_1,s) \xrightarrow{a} (\delta_1',s')} \quad \text{if } \text{ASK}(\phi,s) = \textit{true}$$

$$\frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s) \xrightarrow{a} (\delta_2', s')}{(\delta_2,s) \xrightarrow{a} (\delta_2',s')} \quad \text{if } \text{ASK}(\phi,s) = \textit{false}$$

$$while: \quad \frac{(\textbf{while } \phi \textbf{ do } \delta, s) \xrightarrow{a} (\delta'; \textbf{while } \phi \textbf{ do } \delta, s')}{(\delta,s) \xrightarrow{a} (\delta',s')} \quad \text{if } \text{ASK}(\phi,s) = \textit{true}$$

$$pick: \quad \frac{(\textbf{pick } q(\boldsymbol{x}).\ \delta[x], s) \xrightarrow{a} (\delta'[\boldsymbol{t}], s')}{(\delta[\boldsymbol{t}],s) \xrightarrow{a} (\delta'[\boldsymbol{t}],s')} \quad \textit{(for } \boldsymbol{t} = \text{CHOICE}[\text{ASK}(q(\boldsymbol{x}),s)]\textit{)}$$

Fig. 1. Transition rules

$$\epsilon: \quad \frac{(\epsilon,s)^{\surd}}{\textit{true}} \qquad seq: \quad \frac{(\delta_1;\delta_2,\ s)^{\surd}}{(\delta_1,s)^{\surd} \ \wedge \ (\delta_2;s)^{\surd}}$$

$$if: \quad \frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s)^{\surd}}{(\delta_1,s)^{\surd}} \quad \text{if } \text{ASK}(\phi,s) = \textit{true}$$

$$\frac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2, s)^{\surd}}{(\delta_2,s)^{\surd}} \quad \text{if } \text{ASK}(\phi,s) = \textit{false}$$

$$while: \quad \frac{(\textbf{while } \phi \textbf{ do } \delta, s)^{\surd}}{\textit{true}} \quad \text{if } \text{ASK}(\phi,s) = \textit{false}$$

$$\frac{(\textbf{while } \phi \textbf{ do } \delta, s)^{\surd}}{(\delta,s)^{\surd}} \quad \text{if } \text{ASK}(\phi,s) = \textit{true}$$

$$pick: \quad \frac{(\textbf{pick } q(\boldsymbol{x}).\ \delta[\boldsymbol{x}], s)^{\surd}}{(\delta[\boldsymbol{t}],s)^{\surd}} \quad \textit{(for } \boldsymbol{t} = \text{CHOICE}[\text{ASK}(q(\boldsymbol{x}),s)]\textit{)}$$

Fig. 2. Final rules

which is to be interpreted logically as:

$$\forall(\text{ANTECEDENT} \wedge \text{SIDE-CONDITION} \ \rightarrow \ \text{CONSEQUENT})$$

where $\forall Q$ stands for the universal closure of all free variables occurring in Q, and, typically, ANTECEDENT, SIDE-CONDITION and CONSEQUENT share free variables. The structural rules define inductively a relation, namely: *the smallest relation satisfying the rules*.

Observe the use of the parameter CHOICE, which denotes a choice function, to determine the tuple to be picked in executing the pick constructs of programs. More precisely, CHOICE stands for any function, depending on an arbitrary number of parameters, returning a tuple from the set $\text{ASK}(q(\boldsymbol{x}), s)$. In the original Golog/ConGolog proposal [19, 10] such a choice function (there also extended to other nondeterministic constructs) is implicit, the idea there being that Golog executions use a choice function that would lead to the termination of the program (angelic nondeterminism). In [29], a choice function is also implicit, but based on the idea that choices are done randomly (devilish nondeterminism). Here, we make use of choice functions explicitly, so as to have control on nondeterministic choices. Indeed, one interesting use of CHOICE is to model the delegation of choices to the client of the program, with the idea that the pick construct is interactive: it presents the result of the query to the client, who chooses the tuple s/he is interested in. For example, if the query is about hotels that are available in Rome, the client sees the list of available hotels resulting from the query and chooses the one s/he likes most. We say that a program is *deterministic* when no pick instructions are present or a fixed choice function for CHOICE is considered.

Examples Let us look at a couple of simple examples of programs. Consider the following ontology on companies and grants.

$$\begin{aligned}
\exists owns &\sqsubseteq Company \\
\exists owns^- &\sqsubseteq Company \\
PublicCompany &\sqsubseteq Company \\
PrivateCompany &\sqsubseteq Company \\
\exists grantAsked &\sqsubseteq exResearchGroup \\
\exists grantAsked^- &\sqsubseteq Company \\
IllegalOwner &\sqsubseteq Company
\end{aligned}$$

The first program we write aims at populating the concept $IllegalOwner$ with those companies that own themselves, either directly or indirectly. We assume $temp$ to be an additional role in the alphabet of the TBox. Then, the following deterministic program `ComputeIllegalOwners` can be used to populate $IllegalOwner$:

```
ComputeIllegalOwners =
  erase temp(x1,x2) where q(x1,x2) <- temp(x1,x2);
  erase IllegalOwner(x) where q(x) <- IllegalOwner(x);
  update temp(x1,x2) where q(x1,x2) <- owns(x1,x2);
  while (q() <- K(temp(y1,z), owns(z,y2)), not K(temp(y1,y2))) do (
    update temp(x1,x2) where
        q(x1,x2) <- K(temp(x1,z), owns(z,x2)), not K(temp(x1,x2))
    );
  update IllegalOwner(x) where q(x) <- temp(x,x)
```

The second program we look at is a program that, given a research group r and a company c, interactively –through a suitable choice function for CHOICE– selects a

public company owned by c to ask a grant to; if c does not own public companies, then it selects the company c itself:

```
askNewGrant(r,c) =
  if (q() <- owns(c,y), PublicCompany(y)) then (
    pick (q(x) <- owns(c,x), PublicCompany(x)). (
      update grantAsked(r,x) where true
    )
  )
  else update grantAsked(r,c) where true
```

5 Results

In this section, we assume that ontologies are expressed in *DL-Lite*$_F$ and that the ASK and TELL operations are those defined for *DL-Lite*$_F$ in Section 3.

Given an ontology s and a program δ, we define the set *next step*, denoted by $Next$, as:

$$Next(\delta, s) = \{\langle a, \delta', s' \rangle \mid (\delta, s) \xrightarrow{a} (\delta', s')\}$$

The following two theorems tell us that programs are indeed computable.

Theorem 1. *Let s be an ontology and δ a program. Then, the set $Next(\delta, s)$ has a finite cardinality, and can be computed in polynomial time in s and δ (considering the size of the queries in δ fixed). Moreover, if δ is deterministic then, for each action a, the number of tuples $\langle a, \delta', s' \rangle \in Next(\delta, s)$ is at most one (one if a is executable, zero otherwise).*

Theorem 2. *Let s be an ontology and δ a program. Then, checking $(\delta, s)^{\surd}$ can be done in polynomial time in s and δ (considering the size of the queries in δ fixed).*

Given an ontology s_0 and a sequence $\rho = a_1 \cdots a_n$ of actions, we say that ρ is a *run* of a program δ_0 over the ontology s_0 if there are (δ_i, s_i), for $i = 1, \ldots, n$, such that

$$(\delta_0, s_0) \stackrel{a_1}{\Longrightarrow} (\delta_1, s_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (\delta_n, s_n)$$

We call δ_n and s_n above respectively the program and the ontology resulting from the run ρ. If (δ_n, s_n) is final (i.e., $(\delta_n, s_n)^{\surd}$), then we say that ρ is a *terminating run*. Note that, if the program δ_0 is deterministic, then (δ_n, s_n) is functionally determined by (δ_0, s_0) and ρ.

Theorem 3. *Let s_0 be an ontology, δ_0 a deterministic program, and $\rho = a_1 \cdots a_n$ a sequence of actions. Then checking whether ρ is a run of δ_0 starting from s_0 can be done in polynomial time in the size of s_0, ρ, and δ_0 (considering the size of the queries in δ_0 fixed)*

Theorem 4. *Let s_0 be an ontology, δ_0 a deterministic program, and ρ a run of δ_0 starting from s_0. Then, computing the resulting program δ_n and the resulting ontology s_n, as well as checking $(\delta_n, s_n)^{\surd}$ and computing a query $q(\boldsymbol{x})$ over s_n, can be done in polynomial time in the size of s_0, ρ, and δ_0 (considering the size of the queries in δ_0 fixed).*

For nondeterministic programs, i.e., when we do not fix a choice function for CHOICE, Theorems 3 and 4 do not hold anymore. Indeed, it can be shown the problems in the theorems become NP-complete.

We conclude this section by turning to the two classical problem in reasoning about actions, namely the executability problem and the projection problem [28]. In our setting such problems are phrased as follows:

- *executability problem*: check whether a sequence of actions is executable in an ontology;
- *projection problem*: compute the result of a query in the ontology obtained by executing a sequence of actions in an initial ontology.

Now, considering that a sequence of actions can be seen as a simple deterministic program, from the theorems above we get the following result:

Theorem 5. *Let s_0 be an ontology and ρ a sequence of actions. Then, checking the executability of ρ in s_0, and computing the result of a query $q(\boldsymbol{x})$ over the ontology obtained by executing ρ in s_0, can both be done in polynomial time in the size of s_0 and ρ.*

In fact, all the above results can be immediately extended (with different complexity bounds) to virtually every DL and associated ASK and TELL operations, as long as ASK and TELL are both decidable.

6 Conclusion

In this paper we have laid the foundations for an effective approach to reasoning about actions and programs over ontologies, based on a functional view of the ontology. Namely, the ontology is seen as a system that can perform two kinds of operations: ASK and TELL. We have focused on DL-Lite, but the approach applies to more expressive DLs. It suffices to have a decidable ASK, i.e., decidable query answering on the chosen query and ontology languages, and a decidable TELL, i.e., define atomic actions so that, through their effects, they produce one successor ontology (or, in fact, a finite number of successor ontologies) and such that their executability can be decided. Works such as those reported in [2, 20, 14] are certainly relevant.

Our approach (and the results for *DL-Lite*$_F$) can be extended to all other programming constructs studied within Golog (i.e., non determinism, procedures) [19], ConGolog (i.e., concurrency, prioritized interrupts) [10] and, with some care –see the discussion on analysis and synthesis below– even to those in IndiGolog (search) [29].

Also, the works on forms of execution developed within Golog/ConGolog/IndiGolog can be lifted to DL ontologies by applying the proposed approach. Specifically, notions like online execution [29], offline execution [19, 10], monitored execution [11], can all be lifted to the setting studied here.

Golog/ConGolog-like programs do not have a store to keep memory of previous results of queries to the ontology. An interesting extension would be to introduce such a store, i.e., variables for storing results of queries or partial computations. Notice that this

would make also the program infinite state in general (the ontology is already infinite state). Also, this would make such programs much more alike programs in standard procedural languages such as *C* or *Java*, which manipulate global data structures –in our case the ontology– and local data structures –in our case the information stored in the variables of the program.

Finally, we can adopt the functional view of ontologies also to specify interactive and nonterminating processes acting on them, similarly to what is currently done when specifying web services on relational databases [3, 12].

We close the paper by noticing that, since the ontology is not finite state, tasks related to automated analysis and automated synthesis of programs (e.g., verifying executability on every ontology, verifying termination, synthesizing a plan that achieves a goal, or synthesizing a service that fulfills a certain specification) are difficult in general. This difficulty is shared with SitCalc-based and Golog/ConGolog-like high-level programs. One of the most promising techniques to effectively tackle such tasks is to rely on a suitable finite state *abstraction* (cf. [35]) of the ontology, and use such an abstraction in the analysis and in the synthesis.

Acknowledgements. This research has been partially supported by the FET project TONES (Thinking ONtologiES), funded within the EU 6th Framework Programme under contract FP6-7603.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. F. Baader, C. Lutz, M. Milicic, U. Sattler, and F. Wolter. Integrating description logics and action formalisms: First results. In *Proc. of AAAI 2005*, pages 572–577, 2005.
3. D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based Semantic Web services with messaging. In *Proc. of VLDB 2005*, pages 613–624, 2005.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. DL-Lite: Tractable description logics for ontologies. In *Proc. of AAAI 2005*, pages 602–607, 2005.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR 2006*, pages 260–270, 2006.
6. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. EQL-Lite: Effective first-order query processing in description logics. In *Proc. of IJCAI 2007*, pages 274–279, 2007.
7. G. De Giacomo, L. Iocchi, D. Nardi, and R. Rosati. Moving a robot: the KR&R approach at work. In *Proc. of KR'96*, pages 198–209, 1996.
8. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the update of description logic ontologies at the instance level. In *Proc. of AAAI 2006*, 2006.
9. G. De Giacomo, M. Lenzerini, A. Poggi, and R. Rosati. On the approximation of instance level update and erasure in description logics. In *Proc. of AAAI 2007*, 2007.
10. G. De Giacomo, Y. Lespérance, and H. J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.
11. G. De Giacomo, R. Reiter, and M. Soutchanski. Execution monitoring of high-level robot programs. In *Proc. of KR'98*, pages 453–465, 1998.

12. A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *Proc. of PODS 2006*, pages 90–99, 2006.
13. D. G. Giuseppe and R. Rosati. Minimal knowledge approach to reasoning about actions and sensing. *ETAI*, 3, Section C, 1999.
14. Y. Gu and M. Soutchanski. Decidable reasoning in a modified situation calculus. In *Proc. of IJCAI 2007*, pages 1891–1897, 2007.
15. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.
16. H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. of KR'91*, pages 387–394, 1991.
17. H. J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23:155–212, 1984.
18. H. J. Levesque and G. Lakemeyer. *The Logic of Knowledge Bases*. The MIT Press, 2001.
19. H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming*, 31:59–84, 1997.
20. H. Liu, C. Lutz, M. Milicic, and F. Wolter. Reasoning about actions using description logics with general TBoxes. In *Proc. of JELIA 2006*, 2006.
21. H. Liu, C. Lutz, M. Milicic, and F. Wolter. Updating description logic ABoxes. In *Proc. of KR 2006*, pages 46–56, 2006.
22. D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, Solanki, N. Srinivasan, and K. Sycara. Bringing semantics to web services: The OWL-S approach. In *Proc. of the 1st Int. Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, 2004.
23. J. McCarthy. Towards a mathematical science of computation. In *Proc. of the IFIP Congress*, pages 21–28, 1962.
24. J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of aritificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
25. R. P. A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of KR 2004*, pages 613–622, 2004.
26. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.
27. R. Reiter. What should a database know? *J. of Logic Programming*, 14:127–153, 1990.
28. R. Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.
29. S. Sardiña, G. De Giacomo, Y. Lespérance, and H. J. Levesque. On the semantics of deliberation in IndiGolog - from theory to implementation. *Ann. of Mathematics and Artificial Intelligence*, 41(2–4):259–299, 2004.
30. S. Sardiña, G. De Giacomo, Y. Lespérance, and H. J. Levesque. On the limits of planning over belief states under strict uncertainty. In *Proc. of KR 2006*, pages 463–471, 2006.
31. M. K. Smith, C. Welty, and D. L. McGuiness. OWL Web Ontology Language guide. W3C Recommendation, Feb. 2004. Available at `http://www.w3.org/TR/owl-guide/`.
32. M. B. van Riemsdijk, F. S. de Boer, M. Dastani, and J.-J. C. Meyer. Prototyping 3APL in the Maude term rewriting language. In *Proc. of 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 1279–1281, 2006.
33. M. Winslett. Reasoning about action using a possible models approach. In *Proc. of AAAI'98*, 1988.
34. M. Winslett. *Updating Logical Databases*. Cambridge University Press, 1990.
35. L. D. Zuck and A. Pnueli. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures*, 30(3–4):139–169, 2004.