1

Actions and Programs over Description Logic Knowledge Bases: A Functional Approach

DIEGO CALVANESE, GIUSEPPE DE GIACOMO, MAURIZIO LENZERINI, AND RICCARDO ROSATI

ABSTRACT. We aim at reasoning about actions and about high-level programs over knowledge bases (KBs) expressed in Description Logics (DLs). This is a critical issue that has resisted good, robust 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 the Situation Calculus, these theories do not apply to DL KBs, since these impose very rich state constraints (all the intensional part of the ontology itself). Here we propose a radical solution: we assume a Levesque's functional view of KBs 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 knowledge base as the result of the application of an atomic action. In particular, we consider DL KBs formed by two components: a TBox, providing the intensional knowledge about the domain of interest, which we assume to be immutable over time; and an ABox, providing (incomplete) information at the extensional level, which we assume to be changed by atomic actions based on generic forms of instance level updates. The only requirement that we pose on such updates is that the resulting ABox is still expressible in the original DL language. We demonstrate the effectiveness of the approach by introducing Golog/ConGolog-like high-level programs on DL KBs, characterizing the notion of single-step executability of such programs, and devising nice methods for reasoning about sequences of actions generated by such programs. All our basic results are parametric wrt the specific DL language. Though for concreteness we present them using a particularly well behaved DL, namely $\mathit{DL\text{-}Lite}_{A,id}$.

1 Introduction

In this paper we look at reasoning about actions over Description Logic (DL) knowledge bases (KBs). In doing so, we merge two areas to which Hector Levesque has profoundly contributed: that of DLs, where the Brachman & Levesque's seminal paper on the trade-off between expressiveness and computational complexity in DLs, presented at AAAI'84 [Brachman and Levesque 1984], has shaped nearly all successive research in the field; and that of high-level programs directly based on logics, in particular Golog and ConGolog based on the Situation Calculus (SitCalc), where the work of Levesque and Reiter has demonstrated, possibly for the first time, the maturity of the area of reasoning about actions [Levesque et al. 1997; De Giacomo et al. 2000; Reiter 2001]. Interestingly, in order to make reasoning about actions over DL KBs feasible, we resort to a third foundational contribution by Hector Levesque: the so-called "functional view of knowledge bases" presented in [Levesque 1984], in which KBs are seen as sophisticated objects whose basic operation are "ASK", to extract knowledge from the KB, and "TELL", to update the knowledge in the KB, and where both operation are based on well characterized logical reasoning tasks.

In fact, research on reasoning about actions over DL KBs is currently of particular interest. Indeed, DL KBs [Baader et al. 2003] are generally advocated as the right tool to express ontologies, and this belief is one of the cornerstones of the Semantic Web [Smith et al. 2004; Horrocks et al. 2003]. Notably, semantic web services [Martin et al. 2004] 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. However, such a critical issue has resisted good solutions for a long time, and even big efforts such as OWL-S [Martin et al. 2004] have not really succeeded.

In AI, the importance of combining static and dynamic knowledge has been recognized early [McCarthy 1962; McCarthy and Hayes 1969]. By now, well developed theories of actions and high-level programs, such as Levesque and others' Golog/ConGolog, exist. Note that high-level programs 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 easily to general DL KBs, which impose a very rich kind of state constraints.

DL KBs are formed by two components, a TBox and an ABox. The TBox provides intensional knowledge about the domain of interest, expressed in terms of assertions about concepts, denoting sets (or classes) of individuals, and roles, denoting binary relationships (or associations) between individuals in such classes. The ABox provides facts asserting the membership of single individuals to classes or of pairs of individuals to roles. Such facts constitute an incomplete description of the information about the domain of interest at the extensional level. The TBox assertions in a DL KB typically do not provide *definitions* of concepts and roles, but only interrelations between them (cf. cyclic TBox interpreted according to the so-called descriptive semantics [Baader et al. 2003]). Such non-definitorial nature of DL KBs makes them one of the most difficult kinds of domain descriptions for reasoning about actions, since to come with them means to come with complex forms of state constraints that must be maintained over time [Baader et al. 2005; Liu et al. 2006a].

As mentioned, here we propose a radical solution: we assume a functional view [Levesque 1984] of KBs and see them as systems that allow for two kinds of operations: ASK, which returns the (certain) answer to a query over teh KB, and TELL, which produces a new KB as a result of the application of an atomic action. Observe that this approach, whose origins come from [De Giacomo et al. 1996; Giuseppe and Rosati 1999; Petrick and Bacchus 2004; van Riemsdijk et al. 2006], has some subtle limitations, due to the fact that we lose the possibility of distinguishing between "knowledge" and "truth" as pointed out in [Sardiña et al. 2006]. On the other hand, it has a major advantage: it decouples reasoning on the static knowledge from reasoning 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 assume the TBox of a KB to be immutable over time, while the ABox might be changed by atomic actions used by the TELL operation and based on generic forms of instance level updates. The only requirement that we pose on such updates is that the resulting ABox is still expressible in the original KB language. Building on this functional view, we introduce Golog/ConGolog-like high level programs over DL KBs, we characterize the notion of single-step executability of such programs, and we devise methods for reasoning about sequences of actions generated by such programs. All our basic results are parametric with respect to the specific DL language. Though for concreteness we present them using a particularly well behaved DL, namely *DL-Lite*$_{A,id}$, which is an expressive member of the *DL-Lite* family [Calvanese et al. 2007b], a family of DLs that enjoys particularly nice computational properties when reasoning about knowledge at the instance level. We stress that this paper is really an illustration of what a functional view on KBs can bring about in combining static and dynamic aspects in the context of DL KBs, 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) [Baader et al. 2003] 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 *knowledge bases* (KBs) are based on an alphabet of object, concept, and role symbols, and are formed by two distinct parts: the so-called *TBox*, which represents the *intensional level* of the KB, and contains an intensional description of the domain of interest; and the so-called *ABox*, which represents the *instance level* of the KB, and contains extensional information.

We give the semantics of a DL KB in terms of interpretations over a fixed infinite domain Δ of objects. We assume to have one constant in the alphabet 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 [Levesque and Lakemeyer 2001]).

An *interpretation* $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ consists of a first order structure over Δ, and an interpretation function $\cdot^{\mathcal{I}}$, 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 KB* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, or also that $\mathcal{I}$ *satisfies* $\mathcal{K}$, 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 by $Mod(\mathcal{S})$ the set of interpretations that are models of all assertions in $\mathcal{S}$. In particular, the set of *models of* $\mathcal{K}$, denoted $Mod(\mathcal{K})$, is the set of models of all assertions in $\mathcal{T}$ and $\mathcal{A}$, i.e., $Mod(\mathcal{K}) = Mod(\langle \mathcal{T}, \mathcal{A} \rangle) = Mod(\mathcal{T} \cup \mathcal{A})$. A KB $\mathcal{K}$ is *consistent* if $Mod(\mathcal{K}) \neq \emptyset$, i.e., it has at least one model. We say that a KB $\mathcal{K}$ *logically implies* an expression α (e.g., an assertion, an instantiated conjunctive query (CQ), or an instantiated union of conjunctive queries (UCQ), etc.), written $\mathcal{K} \models \alpha$, if for every interpretation $\mathcal{I} \in Mod(\mathcal{K})$, we have that $\mathcal{I} \in Mod(\alpha)$, i.e., all the models of $\mathcal{K}$ are also models of α.

When dealing with queries, we are interested in *query answering* (for CQs and UCQs): given a KB $\mathcal{K}$ and a query $q(\vec{x})$ over $\mathcal{K}$, return the *certain answers* to $q(\vec{x})$ over $\mathcal{K}$, i.e., all tuples $\vec{t}$ of elements of Δ such that $\mathcal{K} \models q(\vec{t})$, where $q(\vec{t})$ denotes the query obtained from $q(\vec{x})$ by substituting $\vec{x}$ with $\vec{t}$.

***DL-Lite*$_{A,id}$.** The *DL-Lite* family [Calvanese et al. 2007b] is a family of low complexity DLs particularly suited for dealing with KBs with very large ABoxes, and forms the basis of OWL 2 QL, one of the profiles of OWL 2, the official ontology specification language of the World-Wide-Web Consortium (W3C)[1].

We now present the DL *DL-Lite*$_{A,id}$ [Calvanese et al. 2008], which is the most expressive logic in the family. Expressions in *DL-Lite*$_{A,id}$ are formed according to the following syntax:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
B & \longrightarrow & A \mid \exists Q \mid \delta(U) & E & \longrightarrow & \rho(U) \\
C & \longrightarrow & B \mid \neg B & F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \\
Q & \longrightarrow & P \mid P^- & V & \longrightarrow & U \mid \neg U \\
R & \longrightarrow & Q \mid \neg Q & & &
\end{array}$$

where A, P, and U denote respectively an atomic concept name, an atomic role name, and an attribute name, $T_1, \ldots, T_n$ are the value-domains allowed in the logic (which correspond to the data types adopted by Resource Description Framework (RDF)[2]), $\top_D$ denotes the union of all domain values, P^- denotes the inverse of P, $\exists Q$ denotes the objects related to some object by the role Q, $\neg$ denotes negation of concepts, roles, or attributes, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values, and $\rho(U)$ denotes the *range* of U, i.e., the set of values to which U relates objects.

A *DL-Lite*$_{A,id}$ TBox $\mathcal{T}$ contains intensional assertions of three types, namely inclusion assertions, functionality assertions, and identification assertions (IDs) [Calvanese et al. 2008]. More precisely, *DL-Lite*$_{A,id}$ assertions are of the form:

$B \sqsubseteq C$	*concept inclusion assertion*
$E \sqsubseteq F$	*value-domain inclusion assertion*
$Q \sqsubseteq R$	*role inclusion assertion*
$(\mathsf{funct}\ Q)$	*role functionality assertion*
$(\mathsf{funct}\ U)$	*attribute functionality assertion*
$(\mathsf{id}\ B\ \pi_1, ..., \pi_n)$	*identification assertions*

In the identification assertions, π denotes a *path*, which is an expression built according to the following syntax rule:

$$\pi \quad\longrightarrow\quad S \mid B? \mid \pi_1 \circ \pi_2$$

where S denotes an atomic role, the inverse of an atomic role, or an atomic attribute, $\pi_1 \circ \pi_2$ denotes the composition of the paths π_1 and π_2, and $B?$, called *test relation*, represents the identity relation on instances of the concept B. The *length* of a path is inductively defined as follows: the length of a path whose form is S or $B?$ is 1; the length of a path of the form $\pi_1 \circ \pi_2$ is the sum of the lengths of π_1 and of π_2. In *DL-Lite*$_{A,id}$, identification assertions are *local*, i.e., at least one $\pi_i \in \{\pi_1, ..., \pi_n\}$ has length 1. In what follows, we only refer to IDs which are local.

[1]`http://www.w3.org/TR/2008/WD-owl2-profiles-20081008/`
[2]`http://www.w3.org/RDF/`

A concept inclusion assertion specifies that a (basic) concept B is subsumed by a (general) concept C. Analogously for the other types of inclusion assertions. Attribute functionality assertions are used to impose that attributes are actually functions from objects to domain values. Finally, an ID $(\mathsf{id}\ B\ \pi_1, ..., \pi_n)$ asserts that for any two different instances a, b of B, there is at least on π_i such that a and b differ in the set of their π_i-fillers.

In order to guarantee the good computational properties of the DLs of the *DL-Lite* family, a *DL-Lite*$_{A,id}$ TBox $\mathcal{T}$ has to satisfy the following conditions:

- for each atomic role P, if either $(\mathsf{funct}\ P)$ or $(\mathsf{funct}\ P^-)$ occur in $\mathcal{T}$, then $\mathcal{T}$ does not contain assertions of the form $Q \sqsubseteq P$ or $Q \sqsubseteq P^-$, where Q is a basic role;
- for each ID α in $\mathcal{T}$, every role that occurs (in either direct or inverse direction) in a path of α, does not appear in the right-hand side of assertions of the form $Q \sqsubseteq Q'$.

Intuitively, these conditions say that, in *DL-Lite*$_{A,id}$ TBoxes, roles occurring in functionality or identification assertions cannot be specialized.

We observe that *DL-Lite*$_{A,id}$ is able to capture all essential features of conceptual modeling formalisms, such as UML Class Diagrams or Entity-Relationship schemas, with the notable exception of *covering* constraints in generalization hierarchies, which would require the introduction of disjunction. Indeed, it can be shown that, if disjunction was added to *DL-Lite*$_{A,id}$, then query answering would become intractable with respect to the size of the ABox [Calvanese et al. 2006].

A *DL-Lite*$_{A,id}$ ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$, $P(a, b)$, and $U(a, v)$, where A, P, and U are as above, a and b are object constants, and v is a value constant.

Answering *EQL-Lite*(UCQ) queries over *DL-Lite*$_{A,id}$ knowledge bases. As query language, here we consider *EQL-Lite*(UCQ) [Calvanese et al. 2007a]. This language 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[3]. The operator $\mathbf{K}$ is a minimal knowledge operator [Levesque 1984; Reiter 1990; Levesque and Lakemeyer 2001], which is used to formalize the epistemic state of the KB. Informally, the formula $\mathbf{K}\alpha$ is read as "α is known to hold" or "α is logically implied by the knowledge base".

Note that answering *EQL-Lite*(UCQ) queries over *DL-Lite*$_{A,id}$ KBs $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is LOGSPACE with respect to the size of $\mathcal{A}$. Notably, query answering can be reduced to evaluating (pure) FOL queries over the ABox, considered as a database. We refer to [Calvanese et al. 2007a] for more details.

DL instance-level update and erasure. Besides answering queries, Description Logic systems should be able to cope with the *evolution of the KB*. There are two types of evolution operators, corresponding to inserting and deleting chunks of knowledge, respectively. In the case of insertion, the aim is to incorporate new knowledge into the KB, and the corresponding operator should be defined in such a way to compute a consistent KB that supports the new knowledge. In the case of deletion, the aim is to come up with a consistent KB where the retracted knowledge is not valid. Many recent papers demonstrate that the interest towards a well-defined approach to KB evolution is growing significantly [Flouris et al. 2008; Liu et al. 2006b; De Giacomo et al. 2009; Wang et al. 2010; Calvanese et al. 2010].

Following the tradition of the work on knowledge revision and update [Katsuno and Mendelzon 1991], all the above papers advocate some minimality criterion in the changes

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

of the KB that must be undertaken to realize the evolution operations. In other words, the need is commonly perceived of keeping the distance between the original KB and the KB resulting from the application of an evolution operator minimal. There are two main approaches to define such a distance, called *model-based* and *formula-based*, respectively. In the model-based approaches, the result of an evolution operation applied to the KB $\mathcal{K}$ is defined in terms of a set of models, with the idea that such a set should be as close as possible to the models of $\mathcal{K}$. One basic problem with this approach is to characterize the language needed to express the KB that exactly captures the resulting set of models. Conversely, in the formula-based approaches, the result is explicitly defined in terms of a formula, by resorting to some minimality criterion with respect to the formula expressing $\mathcal{K}$. Here, the basic problem is that the formula constituting the result of an evolution operation is not unique in general.

Virtually all model-based approaches suffer from the expressibility problem (see, e.g., [Liu et al. 2006b; De Giacomo et al. 2009; Calvanese et al. 2010]). For this reason, we adopt here a formula-based approach, inspired in particular by the work developed in [Fagin et al. 1983] for updating logical theories. As in [Fagin et al. 1983], we consider both insertions and deletions, but we limit our attention to insertions and deletions of ABox assertions. In other words, we consider the evolution of the ABox of a KB under an invariant TBox, on the basis of the fact that, in many applications, the TBox represents a stable representation of the intensional knowledge about the domain. The specific approach we take is inspired by two recent methods proposed for the evolution of KBs expressed in the *DL-Lite* family [Calvanese et al. 2010; Lenzerini and Savo 2011]. Both methods follow the formula-based approach, and guarantee that the result of the evolution of a KB is always expressible in the DL used to specify the original KB.

3 Atomic actions

Under Levesque's functional view, KBs are seen as systems that are able to perform two basic kinds of operations, namely ASK and TELL operations [Levesque 1984; Levesque and Lakemeyer 2001]:

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

We consider KBs expressed in arbitrary DLs languages, except that we require that they are able to deal with *EQL-Lite*(UCQ) as query language. This implies that they need to be able to handle certain answers of unions of conjunctive queries in a decidable/effective way. With this assumption in place, we base ASK on certain answers to *EQL-Lite*(UCQ). Specifically, we denote by $q(\vec{x})$ an (*EQL-Lite*(UCQ)) query with distinguished variables $\vec{x}$. For a KB $\mathcal{K}$, we define $\text{ASK}(q(\vec{x}), \mathcal{K}) = \{\vec{t} \mid \mathcal{K} \models q(\vec{t})\}$, where $\vec{t}$ denotes a tuple of constants of the same arity as $\vec{x}$. 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 [De Giacomo et al. 2006; De Giacomo et al. 2007]. Specifically, we allow for *atomic actions* of the form

$$\mathbf{update}_{op}\ L(\vec{x})\ \mathbf{where}\ q(\vec{x})$$

where $q(\vec{x})$ stands for a query with $\vec{x}$ as distinguished variables, $L(\vec{x})$ stands for a set of membership assertions on constants and variables in $\vec{x}$, and **update**$_{op}$ is an update operator that makes use of $L(\vec{x})$ to update the KB.

We allow for several update operators, for various forms of update and erasure [Katsuno and Mendelzon 1991; Eiter and Gottlob 1992]. However, we require that by applying one such update operator **update**$_{op}$ to a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ we get a single new KB $\mathcal{K}' = (\mathcal{T}, \mathcal{A}')$ with $\mathcal{A}'$ still in the same DL language of $\mathcal{K}$, or we fail. The new ABox is defined as $\mathcal{A}' = f_{op}(L(\vec{x}), q(\vec{x}), \mathcal{K})$, where f_{op} characterizes the semantics of the update operator **update**$_{op}$. If for any reason $f_{op}(L(\vec{x}), q(\vec{x}), \mathcal{K})$ is not defined, then the update fails, and the action is not executable.

Requiring that the result of the update is still a single KB in the same language as the original one is somehow a severe restriction, since we know that most classical knowledge update operators [Katsuno and Mendelzon 1991; Eiter and Gottlob 1992] applied to DL KBs produce results that are not expressible in the DL of the original KB [De Giacomo et al. 2006; De Giacomo et al. 2007]. On the other hand, from a pragmatical point of view, such an assumption is essential, since it guarantees that by applying the update we can still use the reasoning techniques/algorithms that we used for the original KB. Dropping such assumption would have a disruptive effect on the system: the reasoning techniques in the various states of our system would need to be different.[4]

Let $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, then we define:

$$\textsc{tell}([\textbf{update}_{op}\, L(\vec{x})\, \textbf{where}\, q(\vec{x})], \mathcal{K}) = \begin{cases} \bot, & \text{if } f_{op}(L(\vec{x}), q(\vec{x}), \mathcal{K}) \text{ is undefined} \\ (\mathcal{T}, f_{op}(L(\vec{x}), q(\vec{x}), \mathcal{K}), & \text{otherwise} \end{cases}$$

If $\bot$ is returned by the TELL operation, we say that the atomic action a is *not executable* in $\mathcal{K}$. We extend ASK with expressions of the form $\textsc{ask}([\mathit{executable}(a)], \mathcal{K})$, 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 KB.

We close the section by discussing these notions on *DL-Lite*$_{A,id}$ KBs. First *DL-Lite*$_{A,id}$ allows for computing certain answers of *EQL-Lite*(UCQ) in LOGSPACE in data complexity (as in relational databases) and PTIME in KB complexity, making ASK particularly effective (we assume the size of the query fixed). As for TELL we consider two forms of updates in *DL-Lite*$_{A,id}$: **update**$_{add}$ is based on the notion of update mentioned above, and **update**$_{erase}$ is based on the notion of erasure. Both these operation can be done in polynomial time wrt the KB, and hence also TELL is polynomial (again we assume the size of the query fixed).

4 Programs

We now consider how atomic actions can be organized within a program. In particular, we focus on a variant of Golog [Levesque et al. 1997; De Giacomo et al. 2000; Sardiña et al. 2004] tailored to work on KBs. Instead of situations, we consider KBs, or, to be more precise, KB states. We recall that when considering KBs we assume the TBox to be invariant, so the only part of the KB that can change as a result of an action (or a program) is the ABox.

[4]In fact one aspect of this assumption could be dropped: the fact that the update results in a *single* resulting ABox. We could possibly assume that the resulting ABoxes could be many. Now if they could be finitely many, then the results here could be easily extended, if they could be infinitely many, then more work needs to be done.

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 $q(\vec{x}).\delta[\vec{x}]$	pick

where a is an atomic instruction that 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 KB; finally, **pick** $q(\vec{x}).\delta[\vec{x}]$ picks a tuple $\vec{t}$ in the answer to $q(\vec{x})$, instantiates the rest of the program δ by substituting $\vec{x}$ with $\vec{t}$ and executes δ. The latter construct is a variant of the pick construct in Golog: the main difference being that $\vec{t}$ is bounded by a query to the KB. 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 [Plotkin 1981; De Giacomo et al. 2000]. 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 "$\longrightarrow$":

$$(\delta,\mathcal{K}) \xrightarrow{a} (\delta',\mathcal{K}')$$

where δ is a program, $\mathcal{K}$ is a KB in which the program is executed, a is the executed atomic action, $\mathcal{K}'$ is the KB obtained by executing a in $\mathcal{K}$, 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 "$\cdot^{\surd}$":

$$(\delta,\mathcal{K})^{\surd}$$

where δ is a program that can be considered (successfully) terminated with the KB $\mathcal{K}$.

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:

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

which is to be interpreted logically as:

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

$$
\begin{array}{llll}
\mathit{act}: & \dfrac{(a,\mathcal{K}) \xrightarrow{a} (\epsilon, \text{TELL}(a,\mathcal{K}))}{\mathit{true}} & \text{if } a \text{ is executable in } \mathcal{K} & \\[2ex]
\mathit{seq}: & \dfrac{(\delta_1;\delta_2,\ \mathcal{K}) \xrightarrow{a} (\delta_1';\delta_2,\mathcal{K}')}{(\delta_1,\mathcal{K}) \xrightarrow{a} (\delta_1';\mathcal{K}')} & \dfrac{(\delta_1;\delta_2,\ \mathcal{K}) \xrightarrow{a} (\delta_2',\mathcal{K}')}{(\delta_2,\mathcal{K}) \xrightarrow{a} (\delta_2';\mathcal{K}')} & \text{if } (\delta_1,\mathcal{K})\surd \\[2ex]
\mathit{if}: & \dfrac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2,\mathcal{K}) \xrightarrow{a} (\delta_1',\mathcal{K}')}{(\delta_1,\mathcal{K}) \xrightarrow{a} (\delta_1',\mathcal{K}')} & \text{if ASK}(\phi,\mathcal{K}) = \mathit{true} & \\[2ex]
 & \dfrac{(\textbf{if } \phi \textbf{ then } \delta_1 \textbf{else } \delta_2,\mathcal{K}) \xrightarrow{a} (\delta_2',\mathcal{K}')}{(\delta_2,\mathcal{K}) \xrightarrow{a} (\delta_2',\mathcal{K}')} & \text{if ASK}(\phi,\mathcal{K}) = \mathit{false} & \\[2ex]
\mathit{while}: & \dfrac{(\textbf{while } \phi \textbf{ do } \delta,\mathcal{K}) \xrightarrow{a} (\delta';\textbf{while } \phi \textbf{ do } \delta,\mathcal{K}')}{(\delta,\mathcal{K}) \xrightarrow{a} (\delta',\mathcal{K}')} & \text{if ASK}(\phi,\mathcal{K}) = \mathit{true} & \\[2ex]
\mathit{pick}: & \dfrac{(\textbf{pick } q(\vec{x}).\ \delta[x],\mathcal{K}) \xrightarrow{a} (\delta'[\vec{t}],\mathcal{K}')}{(\delta[\vec{t}],\mathcal{K}) \xrightarrow{a} (\delta'[\vec{t}],\mathcal{K}')} & (\mathit{for}\ \vec{t} = \text{CHOICE}[\text{ASK}(q(\vec{x}),\mathcal{K})]) &
\end{array}
$$

Figure 1. Transition rules

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 $\text{ASK}(q(\vec{x}),\mathcal{K})$. In the original Golog/ConGolog proposal [Levesque et al. 1997; De Giacomo et al. 2000] 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 [Sardiña et al. 2004], 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.[5] We say that a program is *deterministic* when no pick instructions are present or a fixed choice function for CHOICE is considered.

[5]Note that, since the query q is an *EQL-Lite*(UCQ) query, it is range restricted by definition (cf. [Calvanese et al. 2007a]).

$$\epsilon : \quad \frac{(\epsilon, \mathcal{K})^{\checkmark}}{\mathit{true}} \qquad\qquad \mathit{seq} : \quad \frac{(\delta_1; \delta_2,\ \mathcal{K})^{\checkmark}}{(\delta_1, \mathcal{K})^{\checkmark} \ \wedge\ (\delta_2; \mathcal{K})^{\checkmark}}$$

$$\mathit{if} : \quad \frac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2, \mathcal{K})^{\checkmark}}{(\delta_1, \mathcal{K})^{\checkmark}} \quad \text{if } \textsc{ask}(\phi, \mathcal{K}) = \mathit{true}$$

$$\frac{(\textbf{if}\ \phi\ \textbf{then}\ \delta_1 \textbf{else}\ \delta_2, \mathcal{K})^{\checkmark}}{(\delta_2, \mathcal{K})^{\checkmark}} \quad \text{if } \textsc{ask}(\phi, \mathcal{K}) = \mathit{false}$$

$$\mathit{while} : \quad \frac{(\textbf{while}\ \phi\ \textbf{do}\ \delta, \mathcal{K})^{\checkmark}}{\mathit{true}} \quad \text{if } \textsc{ask}(\phi, \mathcal{K}) = \mathit{false}$$

$$\frac{(\textbf{while}\ \phi\ \textbf{do}\ \delta, \mathcal{K})^{\checkmark}}{(\delta, \mathcal{K})^{\checkmark}} \quad \text{if } \textsc{ask}(\phi, \mathcal{K}) = \mathit{true}$$

$$\mathit{pick} : \quad \frac{(\textbf{pick}\ q(\vec{x}).\ \delta[\vec{x}], \mathcal{K})^{\checkmark}}{(\delta[\vec{t}], \mathcal{K})^{\checkmark}} \quad \mathit{(for\ \vec{t} = \textsc{choice}[\textsc{ask}(q(\vec{x}), \mathcal{K})])}$$

Figure 2. Final rules

Examples. Let us look at some simple examples of programs. Consider the KB on companies and grants shown in Figure 3, also depicted graphically in Figure 4.

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 =
  UPDATEerase temp(x1,x2) where q(x1,x2) <- temp(x1,x2);
  UPDATEerase IllegalOwner(x) where q(x) <- IllegalOwner(x);
  UPDATEadd 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 (
    UPDATEadd temp(x1,x2) where
        q(x1,x2) <- K(temp(x1,z), owns(z,x2)), not K(temp(x1,x2))
    );
  UPDATEadd 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)). (
```

$$
\begin{array}{rcl}
\exists owns & \sqsubseteq & Company \\
\exists owns^- & \sqsubseteq & Company \\
\multicolumn{3}{c}{(\mathsf{funct}\ owns^-)} \\
PublicCompany & \sqsubseteq & Company \\
PrivateCompany & \sqsubseteq & Company \\
PublicCompany & \sqsubseteq & \neg PrivateCompany \\
IllegalOwner & \sqsubseteq & Company \\
\exists grantAsked & \sqsubseteq & ResearchGroup \\
\exists grantAsked^- & \sqsubseteq & Company \\
ResearchGroup & \sqsubseteq & \exists grantAsked \\
\exists belongsTo & \sqsubseteq & ResearchGroup \\
\exists belongsTo^- & \sqsubseteq & ResearchDept \\
ResearchGroup & \sqsubseteq & \exists belongsTo \\
\multicolumn{3}{c}{(\mathsf{funct}\ belongsTo)} \\
\delta(rid) & \sqsubseteq & ResearchGroup \\
\rho(rid) & \sqsubseteq & String \\
\multicolumn{3}{c}{(\mathsf{id}\ ResearchGroup\ rid, belongsTo)}
\end{array}
$$

Figure 3. A simple *DL-Lite*$_{A,id}$ TBox

```
    UPDATEadd grantAsked(r,x) where true
  )
)
else UPDATEadd grantAsked(r,c) where true
```

Finally, consider the case of a program that erases a pair (g, d) from the $belongsTo$ role, where g is an instance of $ResearchGroup$ and d an instance of $ResearchDept$. Since $belongsTo$ is a functional role, in the models of $\mathcal{K}$ before the erasure operation, there cannot be any other object d' connected to g via $belongsTo$. However, the inclusion assertion $ResearchGroup \sqsubseteq \exists belongsTo$ in the TBox "enforces" in all models the existence of such an object, possibly an existentially implied one. Notice also that this does not lead in any way to a violation of the ID $(\mathsf{id}\ ResearchGroup\ rid, belongsTo)$.

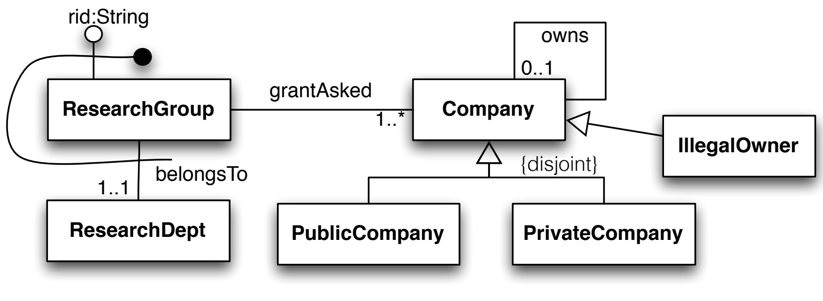

Figure 4. The *DL-Lite*$_{A,id}$ TBox of Figure 3 rendered as a UML Class Diagram

5 Results

In this section, we assume that KBs are expressed in *DL-Lite*$_{A,id}$ and that the ASK and TELL operations are defined for *DL-Lite*$_{A,id}$ using query answering, update (add), and erasure discussed in Section 2 and Section 3.

Given a KB $\mathcal{K}$ and a program δ, we define the set *next step*, denoted by $Next$, as:

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

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

THEOREM 1. *Let $\mathcal{K}$ be a KB and δ a program. Then, the set $Next(\delta,\mathcal{K})$ has a finite cardinality, and can be computed in polynomial time in $\mathcal{K}$ 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 $\langle a,\delta',\mathcal{K}'\rangle \in Next(\delta,\mathcal{K})$ is at most one (one if a is executable, zero otherwise).*

THEOREM 2. *Let $\mathcal{K}$ be a KB and δ a program. Then, checking $(\delta,\mathcal{K})^{\checkmark}$ can be done in polynomial time in $\mathcal{K}$ and δ (considering the size of the queries in δ fixed).*

Given a KB $\mathcal{K}_0$ 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 KB $\mathcal{K}_0$ if there are $(\delta_i,\mathcal{K}_i)$, for $i = 1,\ldots,n$, such that

$$(\delta_0,\mathcal{K}_0) \xrightarrow{a_1} (\delta_1,\mathcal{K}_1) \xrightarrow{a_2} \cdots \xrightarrow{a_n} (\delta_n,\mathcal{K}_n)$$

We call δ_n and $\mathcal{K}_n$ above respectively the program and the KB resulting from the run ρ. If $(\delta_n,\mathcal{K}_n)$ is final (i.e., $(\delta_n,\mathcal{K}_n)^{\checkmark}$), then we say that ρ is a *terminating run*. Note that, if the program δ_0 is deterministic, then $(\delta_n,\mathcal{K}_n)$ is functionally determined by $(\delta_0,\mathcal{K}_0)$ and ρ.

THEOREM 3. *Let $\mathcal{K}_0$ be a KB, 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 $\mathcal{K}_0$ can be done in polynomial time in the size of $\mathcal{K}_0$, ρ, and δ_0 (considering the size of the queries in δ_0 fixed)*

THEOREM 4. *Let $\mathcal{K}_0$ be a KB, δ_0 a deterministic program, and ρ a run of δ_0 starting from $\mathcal{K}_0$. Then, computing the resulting program δ_n and the resulting KB $\mathcal{K}_n$, as well as checking $(\delta_n,\mathcal{K}_n)^{\checkmark}$ and computing a query $q(\vec{x})$ over $\mathcal{K}_n$, can be done in polynomial time in the size of $\mathcal{K}_0$, 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 [Reiter 2001]. In our setting such problems are phrased as follows:

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

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 $\mathcal{K}_0$ be a KB and ρ a sequence of actions. Then, checking the executability of ρ in $\mathcal{K}_0$, and computing the result of a query $q(\vec{x})$ over the KB obtained by executing ρ in $\mathcal{K}_0$, can both be done in polynomial time in the size of $\mathcal{K}_0$ 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 decidable and conform to the requirements mentioned in Section 3.

6 Conclusion

In this paper we have laid the foundations for an effective approach to reasoning about actions and programs over KBs, based on Levesque's functional view of the KB. Namely, the KB 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 KB languages, and a decidable TELL, i.e., define atomic actions so that, through their effects, they produce one successor KB (or, in fact, a finite number of successor KBs) and such that their executability can be decided. Works such as those reported in [Baader et al. 2005; Liu et al. 2006a; Gu and Soutchanski 2007] are certainly relevant.

Our approach (and the results for $\textit{DL-Lite}_{A,id}$) can be extended to all other programming constructs studied within Golog (i.e., non determinism, procedures) [Levesque et al. 1997], ConGolog (i.e., concurrency, prioritized interrupts) [De Giacomo et al. 2000] and, with some care –see the discussion on analysis and synthesis below– even to those in IndiGolog (search) [Sardiña et al. 2004].

Also, the works on forms of execution developed within Golog/ConGolog/IndiGolog can be lifted to DL KBs by applying the proposed approach. Specifically, notions like online execution [Sardiña et al. 2004], offline execution [Levesque et al. 1997; De Giacomo et al. 2000], monitored execution [De Giacomo et al. 1998], 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 KB. 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 KB 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 KB—and local data structures, in our case the information stored in the variables of the program.

Finally, we can adopt the functional view of KBs also to specify interactive and nonterminating processes acting on them, similarly to what is done when specifying web services on relational databases [Berardi et al. 2005; Deutsch et al. 2009].

We close the paper by noticing that the KB is not finite state if we allow for introducing new individuals in its updates. This infinite state nature makes tasks related to automated analysis and automated synthesis of programs (e.g., verifying executability on every KB, verifying termination, synthesizing a plan that achieves a goal, or synthesizing a service that fulfills a certain specification) difficult in general. This difficulty is shared with Situation Calculus based and Golog/ConGolog-like high-level programs. On the other hand if we bound the number of new individuals that can be introduced by updates then the KB becomes finite state. It is not obvious whether we can in practice bound the numbers of individual a priori, but if we cannot, we could still make use of forms of *abstraction*, studied in verification (see e.g.,. [Zuck and Pnueli 2004]), to force finite states.

References

Baader, F., D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider (Eds.) [2003]. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Baader, F., C. Lutz, M. Milicic, U. Sattler, and F. Wolter [2005]. Integrating description logics and action formalisms: First results. In *Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005)*, pp. 572–577.

Berardi, D., D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella [2005]. Automatic composition of transition-based Semantic Web services with messaging. In *Proc. of the 31st Int. Conf. on Very Large Data Bases (VLDB 2005)*, pp. 613–624.

Brachman, R. J. and H. J. Levesque [1984]. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI'84)*, pp. 34–37.

Calvanese, D., G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati [2006]. Data complexity of query answering in description logics. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pp. 260–270.

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

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

Calvanese, D., G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati [2008]. Path-based identification constraints in description logics. In *Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008)*, pp. 231–241.

Calvanese, D., E. Kharlamov, W. Nutt, and D. Zheleznyakov [2010]. Evolution of *DL-Lite* knowledge bases. In *Proc. of the 9th Int. Semantic Web Conf. (ISWC 2010)*, Volume 6496 of *Lecture Notes in Computer Science*, pp. 112–128. Springer.

De Giacomo, G., L. Iocchi, D. Nardi, and R. Rosati [1996]. Moving a robot: the KR&R approach at work. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pp. 198–209.

De Giacomo, G., M. Lenzerini, A. Poggi, and R. Rosati [2006]. On the update of description logic ontologies at the instance level. In *Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006)*, pp. 1271–1276.

De Giacomo, G., M. Lenzerini, A. Poggi, and R. Rosati [2007]. On the approximation of instance level update and erasure in description logics. In *Proc. of the 22nd AAAI Conf. on Artificial Intelligence (AAAI 2007)*, pp. 403–408.

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

De Giacomo, G., Y. Lespérance, and H. J. Levesque [2000]. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121*(1–2), 109–169.

De Giacomo, G., R. Reiter, and M. Soutchanski [1998]. Execution monitoring of high-level robot programs. In *Proc. of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pp. 453–465.

Deutsch, A., R. Hull, F. Patrizi, and V. Vianu [2009]. Automatic verification of data-centric business processes. In *Proc. of the 12th Int. Conf. on Database Theory (ICDT 2009)*, pp. 252–267.

Eiter, T. and G. Gottlob [1992]. On the complexity of propositional knowledge base revision, updates and counterfactuals. *Artificial Intelligence 57*, 227–270.

Fagin, R., J. D. Ullman, and M. Y. Vardi [1983]. On the semantics of updates in databases. In *Proc. of the 2nd ACM SIGACT SIGMOD Symp. on Principles of Database Systems (PODS'83)*, pp. 352–365.

Flouris, G., D. Manakanatas, H. Kondylakis, D. Plexousakis, and G. Antoniou [2008]. Ontology change: Classification and survey. *Knowledge Engineering Review 23*(2), 117–152.

Giuseppe, D. G. and R. Rosati [1999]. Minimal knowledge approach to reasoning about actions and sensing. *Electronic Trans. on Artificial Intelligence 3*(C), 1–18.

Gu, Y. and M. Soutchanski [2007]. Decidable reasoning in a modified situation calculus. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pp. 1891–1897.

Horrocks, I., P. F. Patel-Schneider, and F. van Harmelen [2003]. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics 1*(1), 7–26.

Katsuno, H. and A. Mendelzon [1991]. On the difference between updating a knowledge base and revising it. In *Proc. of the 2nd Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'91)*, pp. 387–394.

Lenzerini, M. and F. Savo [2011]. On the evolution of the instance level of *DL-Lite* knowledge bases. Submitted for publication.

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

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

Levesque, H. J., R. Reiter, Y. Lesperance, F. Lin, and R. Scherl [1997]. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming 31*, 59–84.

Liu, H., C. Lutz, M. Milicic, and F. Wolter [2006a]. Reasoning about actions using description logics with general TBoxes. In *Proc. of the 10th Eur. Conference on Logics in Artificial Intelligence (JELIA 2006)*, Volume 4160 of *Lecture Notes in Computer Science*. Springer.

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

Martin, D., M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, Solanki, N. Srinivasan, and K. Sycara [2004]. 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)*.

McCarthy, J. [1962]. Towards a mathematical science of computation. In *Proc. of the IFIP Congress*, pp. 21–28.

McCarthy, J. and P. J. Hayes [1969]. Some philosophical problems from the standpoint of aritificial intelligence. *Machine Intelligence 4*, 463–502.

Petrick, R. P. A. and F. Bacchus [2004]. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. of the 9th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2004)*, pp. 613–622.

Plotkin, G. D. [1981]. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus.

Reiter, R. [1990]. What should a database know? *J. of Logic Programming 14*, 127–153.

Reiter, R. [2001]. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

Sardiña, S., G. De Giacomo, Y. Lespérance, and H. J. Levesque [2004]. On the semantics of deliberation in IndiGolog - from theory to implementation. *Ann. of Mathematics and Artificial Intelligence 41*(2–4), 259–299.

Sardiña, S., G. De Giacomo, Y. Lespérance, and H. J. Levesque [2006]. On the limits of planning over belief states under strict uncertainty. In *Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006)*, pp. 463–471.

Smith, M. K., C. Welty, and D. L. McGuiness [2004, February]. OWL Web Ontology Language guide. W3C Recommendation, World Wide Web Consortium. Available at `http://www.w3.org/TR/owl-guide/`.

van Riemsdijk, M. B., F. S. de Boer, M. Dastani, and J.-J. C. Meyer [2006]. Prototyping 3APL in the Maude term rewriting language. In *Proc. of 5th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pp. 1279–1281.

Wang, Z., K. Wang, and R. W. Topor [2010]. A new approach to knowledge base revision in *DL-Lite*. In *Proc. of the 24th AAAI Conf. on Artificial Intelligence (AAAI 2010)*.

Zuck, L. D. and A. Pnueli [2004]. Model checking and abstraction to the aid of parameterized systems (a survey). *Computer Languages, Systems & Structures 30*(3–4), 139–169.