Towards More Informative Answers in Terminological Logics*

Margarida Mamede and Luís Monteiro

Departamento de Informática
Universidade Nova de Lisboa
2825 Monte da Caparica, Portugal
{mm,lm}@fct.unl.pt

Abstract. In this paper, the query-answering problem (also called retrieval problem, in terminological logics) is studied and a new framework is proposed. The declarative semantics extends the classical definition of answer in order to refer to some elements that exist in all models although they may not possess a name. With an appropriate representation of these elements, answers are regular languages and are described by regular expressions. We claim that these regular expressions give a better understanding of the knowledge base that is being queried. The operational semantics relies on a finite automaton which is built, independently of the query, from some completions of the knowledge base. Once this automaton has been generated, answers can be efficiently computed. Although sound, the operational semantics is not complete in the general case, but we state sufficient conditions on the knowledge base under which the two semantics are equivalent.

1 Introduction

Although there are several well-known problems in concept languages (see [2, 3, 12]), in this work we only tackle the retrieval problem. According to the classical definition ([2, 3, 9]), this problem consists of computing the set of individuals that belong to a given concept in all models of the knowledge base.

Some recent terminological languages are quite expressive, as, for example, [1, 2, 5, 6, 11, 13]. So, complex knowledge bases may be created. Nevertheless, the information returned in answers is only a small fraction of the information knowledge bases possess. Although we can formalize complex structures with terminological axioms and universal and existential quantifiers, answers are just a (finite) subset of the initial individual constants. Let us illustrate our point with a simple example. Suppose that a given knowledge base has the following two assertions: "zero is a natural number" and "every natural number has a sucessor, which is a natural number". In spite of the fact that every model of this knowledge base has a "successor of zero", which is a natural number, a

* This work was partially supported by Esprit Project BRA 6810 COMPULOG 2 and JNICT. The referees are thanked for their comments.

"sucessor of a sucessor of zero", which is also a natural number, etc., if we asked for the natural numbers, the answer would be "zero".

This work aimed at closing the gap between the expressive power of terminological languages, in which knowledge bases are written, and the language of answers. To this end, a new framework for dealing with elements that have no explicit representation in the knowledge base is proposed. The advantages of our work are twofold. First, answers refer to these existential elements. Second, and due to the fact that the language of answers is infinite, the query-answering algorithm does not work by generate-and-test, as is usually the case. Instead, the knowledge base is preprocessed, resulting in a finite automaton that captures the elements existing in all models and their properties.

In [10] we studied this problem with $\mathcal{AL}$, a simpler language that did not allow negation of arbitrary concepts, concept disjunction, qualified existential quantification, and terminological axioms. In that case, there was only a finite number of existential elements and each element was related to at most one other element, which made everything much easier. This work extends the previous one, since the language now considered is $\mathcal{ALC}$ with conjunction of roles and generic terminological axioms.

The paper is organized as follows. In Sect. 2, we present the language in which the knowledge is expressed and define some basic notions. In Sect. 3, we give in more detail the motivation of our work and introduce intuitively our proposal. Section 4 is devoted to characterize the query language and Sect. 5 presents the declarative semantics, i.e., contains the definition of answer. The operational semantics is described in Sect. 6. Section 6.1 shows how to build an automaton from some completions of the knowledge base and Sect. 6.2 presents the query-answering algorithm. Then, Sect. 7 focusses on the relations between the declarative and the operational semantics. Finally, Sect. 8 includes some comments on the reasearch done and some directions of future work. Due to space limitations, all proofs are omitted.

2 Knowledge Bases

The language studied in this work is $\mathcal{ALC}$ ([15]) with conjunction of roles and terminological axioms. Given a tuple $(\mathrm{Ind}, \mathrm{Prim}, \mathrm{Rel})$ of pairwise disjoint sets of *individuals* $i \in \mathrm{Ind}$, *primitive concepts* $P \in \mathrm{Prim}$ and *primitive binary relations* $s \in \mathrm{Rel}$, an $\mathcal{ALC}$*-concept* C, an $\mathcal{ALC}$*-relation* r, and an $\mathcal{ALC}$*-assertion* σ are defined by the following grammar:

$$
\begin{array}{l}
C \longrightarrow \top \mid \bot \mid P \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall r.C \mid \exists r.C \\
r \longrightarrow s \mid r_1 \sqcap r_2 \\
\sigma \longrightarrow i : C \mid r(i_1, i_2) \mid C_1 \sqsubseteq C_2
\end{array}
$$

An $\mathcal{ALC}$*-knowledge base* is a finite set of $\mathcal{ALC}$-assertions[2]. Assertions of the form $C_1 \sqsubseteq C_2$ are usually called *terminological axioms*.

[2] Sometimes we omit the prefix "$\mathcal{ALC}$", writing only the words *concept*, *relation*, *assertion*, and *knowledge base*.

An *interpretation* of $\mathcal{ALC}$ over (Ind, Prim, Rel) is a pair $I = (\Delta^I, \cdot^I)$, where Δ^I is a nonempty set, called the *interpretation domain*, and $\cdot^I$ is the *interpretation function* which associates: $i^I \in \Delta^I$ with every $i \in \text{Ind}$, in such a way that, for every individuals $i_1, i_2 \in \text{Ind}$, if $i_1 \neq i_2$ then $i_1^I \neq i_2^I$; $P^I \subseteq \Delta^I$ with every $P \in \text{Prim}$; and $s^I \subseteq \Delta^I \times \Delta^I$ with every $s \in \text{Rel}$.

The interpretation function is extended to arbitrary concepts and relations in the following way, where $r^I(a) = \{b \mid (a,b) \in r^I\}$, for every $a \in \Delta^I$ and every relation r.

$$\begin{array}{rlrl} \top^I &= \Delta^I & (C_1 \sqcap C_2)^I &= C_1^I \cap C_2^I \\ \bot^I &= \emptyset & (C_1 \sqcup C_2)^I &= C_1^I \cup C_2^I \\ (\neg C)^I &= \Delta^I \setminus C^I & (\forall r.C)^I &= \{a \in \Delta^I \mid r^I(a) \subseteq C^I\} \\ (r_1 \sqcap r_2)^I &= r_1^I \cap r_2^I & (\exists r.C)^I &= \{a \in \Delta^I \mid r^I(a) \cap C^I \neq \emptyset\} \end{array}$$

To simplify the notation, the functions r^I are extended in the usual way to arbitrary subsets X of the interpretation domain, i.e., $r^I(X) = \bigcup_{a \in X} r^I(a)$.

An interpretation I *satisfies* an assertion $i : C$ if $i^I \in C^I$, I satisfies $r(i_1, i_2)$ when $(i_1^I, i_2^I) \in r^I$, and I satisfies the terminological axiom $C_1 \sqsubseteq C_2$ if $C_1^I \subseteq C_2^I$. If $\mathcal{I}$ is a set of interpretations, $\mathcal{I} \models \sigma$ means that every element of $\mathcal{I}$ satisfies σ.

A knowledge base Σ is *satisfiable* if there is an interpretation I that satisfies all of its assertions. In this case, I is called a *model* of Σ. When Σ is not satisfiable, we say Σ is *unsatisfiable*. We denote by $\mathcal{M}(\Sigma)$ the set of models of Σ. An assertion σ is a *logical consequence* of the knowledge base Σ if $\mathcal{M}(\Sigma) \models \sigma$.

A concept C is in *normal form* if negation is applied only to primitive concepts. More formally, concepts in normal form are generated by the grammar:

$$\begin{array}{rl} C & \longrightarrow \top \mid \bot \mid P \mid \neg P \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall r.C \mid \exists r.C \\ r & \longrightarrow s \mid r_1 \sqcap r_2 \end{array}$$

Concerning assertions, $i : C$ is in *normal form* if C is in normal form, $r(i_1, i_2)$ is in *normal form* if r is a primitive relation, and $C_1 \sqsubseteq C_2$ is in *normal form* if $C_1 = \neg C_1'$, and C_1' and C_2 are in normal form. Finally, a knowledge base is in *normal form* if all of its assertions are in normal form.

Since every knowledge base can be rewritten in an equivalent knowledge base in normal form (see [15]), from now on, we assume that knowledge bases are in normal form. Also to simplify, we assume that (Ind, Prim, Rel) is the alphabet over which concepts, relations, assertions and knowledge bases are written.

3 The Retrieval Problem

According to the classical definition ([2, 3, 9]), the retrieval problem consists of computing the set of individuals that belong to a given concept in all models of the knowledge base. Given a knowledge base Σ and a concept Q (the query), the *classical answer* to Q is $\text{ClassicalAnswer}(\Sigma, Q) = \{i \in \text{Ind} \mid \mathcal{M}(\Sigma) \models i : Q\}$.

The simplest algorithm for solving this problem generates all individuals i appearing in Σ and, for each one of them, checks the satisfiability of the

knowledge base $\Sigma \cup \{i : \neg Q\}$. Actually, $i : Q$ is a logical consequence of Σ if and only if $\Sigma \cup \{i : \neg Q\}$ is unsatisfiable. Note that when Σ is not satisfiable, the retrieval problem is trivially solved because the answer to any query is the set Ind. For this reason, Σ is checked for satisfiability before being queried.

When the query language is part of the assertional language and is closed for negation of concepts, as in the system KRIS ([2, 3]), the assertions generated by the retrieval algorithm belong to the assertional language, so one simply needs to call the satisfiability algorithm a finite number of times. In [9], both these conditions did not hold and the satisfiability algorithm was modified so that it could handle the kind of assertions the retrieval algorithm generates.

A framework for adding an epistemic operator to the query language was proposed in [7, 8]. This operator, called **K**, increases the expressive power of the query language because it allows to distinguish between the situations in which the knowledge base contains complete information to answer the query and the situations in which the available information is incomplete. The algorithm described in [7, 8] to find out the answer is similar to the classical ones.

In our system, the epistemic operator is not allowed. Actually, the query language is just a subset of the language of $\mathcal{ALC}$-concepts. To illustrate our ideas, let us consider the knowledge base Σ_{nat}. Zero is a natural number and every natural number has a sucessor, which is a natural number.

$$\texttt{Nat} \sqsubseteq \exists \texttt{sucessor}.\texttt{Nat} \qquad \texttt{0} : \texttt{Nat}$$

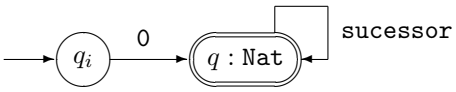
Consider also the query $Q_{nat} = \texttt{Nat}$, which asks for the natural numbers. The classical answer is `0`. Of course zero is the only individual that is a natural number in all models of the knowledge base. However, Σ_{nat} contains much more information about the natural numbers than the one that is being returned. The truth is that, in all models of Σ_{nat}, there is also a sucessor of zero that is a natural number, and a sucessor of a sucessor of zero that is a natural number, and so on. Now the biggest problem, which has neither an easy nor a unique solution, is how to refer to these elements. We represent "a sucessor of zero" by the expression "`0 sucessor`", "a sucessor of a sucessor of zero" by "`0 sucessor sucessor`", etc. Since the language of these terms is regular, it can be described by a regular expression, as, for instance, $\texttt{0 sucessor}^*$, which is, in fact, our answer. Note that with each word, of the form $\texttt{0 sucessor} \cdots \texttt{sucessor}$, is associated an element in $\texttt{sucessor}^M \circ \cdots \circ \texttt{sucessor}^M(\texttt{0}^M)$, for every $M \in \mathcal{M}(\Sigma_{nat})$, that is a natural number, i.e., that belongs to $\texttt{Nat}^M$. Therefore, each word characterizes somehow an unspecified element of Q_{nat}, often called "existential element".

We claim that by taking into account other elements besides individuals, we are able to return answers containing more information. Essentially, because these existential elements, common to all models, allow to understand the structure of the knowledge represented in the base. From our point of view, there is a big gap between the expressive power of terminological languages, in which knowledge bases are written, and the expressive power of the language of answers. Note that we can formalize complex structures, with terminological axioms, and existential and universal quantifiers. Nevertheless, the classical answers to our queries are just a finite set of individual constants. This trivial example shows

our point. Both the classical and the epistemic queries about the natural numbers that exist (resp., `Nat` and **K**`Nat`) give rise to the constant zero, whereas, in our setting, an infinite answer is returned: `0 sucessor`*.

As we said before, the usual query-answering algorithms work by generate-and-test. If Σ is the knowledge base and Q is the query, every individual i is generated once, and the test "Is $\Sigma \cup \{i : \neg Q\}$ satisfiable ?" is made. Note that this method cannot be applied in our case, due to the fact that answers may contain an infinite set of existential elements. Instead, the knowledge base is first preprocessed, resulting in a finite automaton that captures all the information needed to answer queries. Then, given a query Q, the answer to Q is found by performing some computations on this automaton.

Let us illustrate these notions with the automaton obtained by preprocessing Σ_{nat}, which is represented below. The query-answering algorithm performs two steps. First, the final states that "satisfy" the query are selected. Then, a regular expression denoting the language accepted by those states is computed and returned. Back to our query example, the set of selected states would be $\{q\}$ because the only final state q satisfies `Nat`. Intuitively, this means that with any word accepted by q is associated an element, in all models of Σ_{nat}, that is a natural number. So, the computed answer would be `0 sucessor`*.



4 The Query Language

A *query* Q is an $\mathcal{ALC}$-concept in normal form without disjunctions and universal quantifiers[3]. More precisely, queries are generated by the following grammar:

$$\begin{array}{l} Q \longrightarrow \top \mid \bot \mid P \mid \neg P \mid Q_1 \sqcap Q_2 \mid \exists r.Q \\ r \longrightarrow s \mid r_1 \sqcap r_2 \end{array}$$

Since a query Q is an $\mathcal{ALC}$-concept, given an interpretation I, the interpretation of Q by I is the interpretation of the $\mathcal{ALC}$-concept Q by I, i.e., is Q^I.

5 Declarative Semantics

Before presenting the definition of answer, we formalize the notion of term, which is built with individuals and "simple relations". Intuitively, a simple relation is a relation in which any primitive relation occurs at most once.

A relation r of the form $s_1 \sqcap \cdots \sqcap s_m$, where $s_1, \ldots, s_m \in$ Rel and $m \geq 1$, is called a *simple relation* if it satisfies the property: $s_j = s_k \Rightarrow j = k$, for every $1 \leq j, k \leq m$. We denote by Rel$^\diamond$ the set of simple relations built with the symbols of Rel.

[3] This constraint is justified by soudness and completeness reasons (see Sect. 7).

Let Term be the least set satisfying the two following conditions: Ind $\subseteq$ Term; and if $t \in$ Term and $r \in \text{Rel}^{\diamond}$, then $tr \in$ Term. A *term* is an element of Term.

It is easy to verify that every term t has the form $ir_1 \cdots r_n$, for some $n \geq 0$, where i is an individual and $r_1, \ldots, r_n$ are simple relations.

The *denotation* of a term t by an interpretation I, written $[\![t]\!]^I$, is defined by induction on the structure of t as follows: $[\![i]\!]^I = \{i^I\}$ and $[\![t'r]\!]^I = r^I([\![t']\!]^I)$.

Intuitively, the denotation of a term $t = ir_1 \cdots r_n$ by an interpretation I gives the set of elements $a \in \Delta^I$ at which we arrive following a path $r_1^I \cdots r_n^I$ starting from i^I. Given a query Q, let us interpret the expression $t : Q$ as stating that there is an element that belongs to Q in a path indicated by t. An interpretation I "satisfies" this statement if there is an element in $[\![t]\!]^I$ that belongs to Q^I, i.e., if $[\![t]\!]^I \cap Q^I \neq \emptyset$. In some sense, t can be seen as a partial characterization of an element existing in Q^I. Now suppose that every model of a knowledge base Σ "satisfies" $t : Q$. This means that t partially characterizes an element of Q^M, for every model M of Σ, so t will belong to the answer to Q. The next definitions formalize these intuitions.

An *existential statement* is an expression of the form $t : Q$, where t is a term and Q is a query. An interpretation I *satisfies* an existential statement $t : Q$, denoted by $I \models t : Q$, if $[\![t]\!]^I \cap Q^I \neq \emptyset$. When $\mathcal{I}$ is a set of interpretations, $\mathcal{I} \models t : Q$ represents that every element of $\mathcal{I}$ satisfies $t : Q$.

Note that when t is an individual, the latter definition coincides with the one given in Sect. 2, i.e., an interpretation I satisfies an assertion $i : Q$ if and only if it satisfies the existential statement $i : Q$. The proof follows directly from the equivalence $i^I \in Q^I \Leftrightarrow \{i^I\} \cap Q^I \neq \emptyset$. Now, we present the definition of answer.

Let Σ be an $\mathcal{ALC}$-knowledge base and Q be a query. The *answer* to Q is

$$\text{Answer}(\Sigma, Q) = \{t \in \text{Term} \mid \mathcal{M}(\Sigma) \models t : Q\}.$$

This definition extends the classical definition of answer. Besides returning some initial individual constants, our answers also refer to some terms that intend to partially characterize existential elements. The next proposition justifies somehow this designation.

Proposition 1. *Let Σ be an $\mathcal{ALC}$-knowledge base and Q be a query.*

1. *Answer$(\Sigma, Q) \cap$ Ind $=$ ClassicalAnswer(Σ, Q).*
2. *Let $t = ir_1 \cdots r_n$ be a term, for some $n \geq 1$.*
   Then, $t \in$ Answer$(\Sigma, Q) \Leftrightarrow i \in$ ClassicalAnswer$(\Sigma, \exists r_1. \cdots . \exists r_n . Q)$.

There are two special situations that deserve attention. First, when the knowledge base is unsatisfiable, the answer to any query is the set of all terms. Moreover, if no individual occurs in the knowledge base – in which case all assertions are terminological axioms – the set Term is empty, as the answer to any query. Since answers are trivially computed in both situations, these cases will not be studied any further. Notice also that while it is easy to check whether there are individuals or not, the satisfiability test is not so simple. For this reason, in the sequel, we shall assume that knowledge bases contain at least one individual, but we will not consider that they are satisfiable.

6 Operational Semantics

6.1 Automaton for a Satisfiable $\mathcal{ALC}$-Knowledge Base

To begin with, we define another language of assertions, $\mathcal{ALC}_{var}$, which accepts *variables* $v \in \mathrm{Var}$ in any place individuals may occur in $\mathcal{ALC}$-assertions. The set Var is also disjoint from Ind, Prim and Rel. An *object* o is either an individual or a variable, and Obj is the set of objects, i.e., $\mathrm{Obj} = \mathrm{Ind} \cup \mathrm{Var}$.

An interpretation of $\mathcal{ALC}_{var}$ differs from an interpretation of $\mathcal{ALC}$ only in the fact that it associates an element of the domain with every variable. The definitions of satisfaction of an $\mathcal{ALC}_{var}$-assertion, (un)satisfiable $\mathcal{ALC}_{var}$-knowledge base, and model of an $\mathcal{ALC}_{var}$-knowledge base are equal to the ones given in Sect. 2 for the language $\mathcal{ALC}$.

Note that an $\mathcal{ALC}$-assertion is an $\mathcal{ALC}_{var}$-assertion. When the distinction is not important or when it is clear from the context we use only the word *assertion*. Similarly, a *knowledge base* is either a set of $\mathcal{ALC}$-assertions or a set of $\mathcal{ALC}_{var}$-assertions.

The idea is to start generating some assertions that explicitly express information given in a more implicit way, which is performed by applying "completion rules". The most difficult case concerns $\sqcup$-assertions. For example, if $i : C_1 \sqcup C_2 \in \Sigma$, every model of Σ satisfies either $i : C_1$ or $i : C_2$, but, in general, none of these assertions is satisfied by all models. The crucial point is that in order to collect the properties that are common to all models, we have to reason by case analysis, exploring the two different situations.

Let us now summarize the completion process, to give some hints of how completion rules are used. Completions are computed in two steps. First, we generate all assertions concerning individuals, which are $\mathcal{ALC}$-assertions, and we obtain a set of knowledge bases, called "precompletions" of the initial $\mathcal{ALC}$-knowledge base. Each precompletion reflects a different combination of choices made in the application of the $\sqcup$-rule. Second, we resume the completion of the precompletions, generating variables and $\mathcal{ALC}_{var}$-assertions. But now both alternatives of the $\sqcup$-rule are stored in the same set.

Completion rules differ from the usual ones in three respects. First, there is a new rule, called rule (s-r), for dealing with individuals and primitive concepts. It is used only in the first step. Second, there are two different rules for $\sqcup$-assertions. One is nondeterministic, as usual, is applied to individuals and is used during the first step, while the other is deterministic, is applied to variables and is used in the second step. Third, there are some differences concerning application conditions. Rules for $\sqcup$-assertions are always applied once, even when one disjunct already belongs to the knowledge base, and the application condition for the rule that deals with existential assertions is defined from the outside, by a general policy that controls step two.

Before defining the completion rules, we introduce some notation. Let $r = s_1 \sqcap \cdots \sqcap s_m$ and $r' = s'_1 \sqcap \cdots \sqcap s'_n$ be arbitrary relations (where $m, n \geq 1$), and o_1 and o_2 be two objects. The expression $\Sigma \vdash r(o_1, o_2)$ abbreviates $s_j(o_1, o_2) \in \Sigma$ for every $1 \leq j \leq m$, and $r \subseteq r'$ means $\{s_j \mid 1 \leq j \leq m\} \subseteq \{s'_j \mid 1 \leq j \leq n\}$.

The following rules are called *completion rules*, where Σ is a knowledge base:

(s-r) **If** $s(i_1, i_2) \in \Sigma$, $P \in \text{Prim}$ occurs in Σ **and** $i_2 : P \sqcup \neg P \notin \Sigma$
then $\Sigma \leftarrow \Sigma \cup \{i_2 : P \sqcup \neg P\}$.

($\sqcap$-r) **If** $o : C_1 \sqcap C_2 \in \Sigma$ **and** $o : C_1 \notin \Sigma$ or $o : C_2 \notin \Sigma$
then $\Sigma \leftarrow \Sigma \cup \{o : C_1, o : C_2\}$.

($\sqcup_i$-r) **If** $i : C_1 \sqcup C_2 \in \Sigma$ **and** this rule has not been applied to $i : C_1 \sqcup C_2$ yet
then either $\Sigma \leftarrow \Sigma \cup \{i : C_1\}$ or $\Sigma \leftarrow \Sigma \cup \{i : C_2\}$.

($\sqcup_v$-r) **If** $v : C_1 \sqcup C_2 \in \Sigma$ **and** this rule has not been applied to $v : C_1 \sqcup C_2$ yet
then let v' be a new variable and

$$\Sigma' = \{v' : C \mid v : C \in \Sigma\} \cup \{s(o, v') \mid s(o, v) \in \Sigma\}$$

in $\Sigma \leftarrow \Sigma \cup \Sigma' \cup \{v : C_1\} \cup \{v' : C_2\}$.

($\forall$-r) **If** $o : \forall r.C \in \Sigma$, $\Sigma \vdash r(o, o')$ **and** $o' : C \notin \Sigma$
then $\Sigma \leftarrow \Sigma \cup \{o' : C\}$.

($\top$-r) **If** $\neg C_1 \sqsubseteq C_2 \in \Sigma$, $o \in \text{Obj}$ occurs in Σ **and** $o : C_1 \sqcup C_2 \notin \Sigma$
then $\Sigma \leftarrow \Sigma \cup \{o : C_1 \sqcup C_2\}$.

($\exists$-r) **If** $o : \exists r.C \in \Sigma$, with $r = s_1 \sqcap \cdots \sqcap s_n$ for some $n \geq 1$,
then let v be a new variable
in $\Sigma \leftarrow \Sigma \cup \{s_1(o, v), \ldots, s_n(o, v), v : C\}$.

Basically, a precompletion of an $\mathcal{ALC}$-knowledge base Σ is a superset of Σ containing, in an explicit way, one possible combination of the constraints put on individuals. The precise definition is presented next.

Let Σ be an $\mathcal{ALC}$-knowledge base. A *precompletion* of Σ is an $\mathcal{ALC}$-knowledge base obtained by applying the completion rules (s-r), ($\sqcap$-r), ($\sqcup_i$-r), ($\forall$-r) and ($\top$-r), by any order, until none of these rules can be applied.

A knowledge base Σ has a *clash* if it contains: an assertion of the form $o : \bot$, for some object o; or two assertions $o : P$ and $o : \neg P$, for some $o \in \text{Obj}$ and some $P \in \text{Prim}$. Otherwise, we say that Σ is *clash-free*.

Now we state two lemmas. The first concerns computational aspects whereas the second establishes some relations between the satisfiability of the initial knowledge base and the satisfiability of its precompletions.

Lemma 2. *Let Σ be an $\mathcal{ALC}$-knowledge base. The algorithm that computes a precompletion of Σ always terminates. Moreover, the set of precompletions of Σ is finite and nonempty, and can be computed with an appropriate policy.*

Lemma 3. *Let Σ be an $\mathcal{ALC}$-knowledge base.*

1. *Every model of Σ is a model of some precompletion of Σ.*
2. *Σ is satisfiable if and only if some precompletion of Σ is satisfiable.*
3. *If Σ is satisfiable, there is a clash-free precompletion of Σ.*

By Lemma 3 (3.), if there is no clash-free precompletion, the knowledge base is unsatisfiable, so the process stops after this first step. When this is not the case, we proceed with the completion of the clash-free precompletions.

Since there are terminological axioms and existential concepts in $\mathcal{ALC}$, a policy to specify how rules may be applied is required to prevent infinite computations. The policy works as follows. We select an object o that satisfies certain conditions and apply rule ($\exists$-r) to every existential assertion concerning o – what we call "to expand" o. Then, we apply all rules except ($\exists$-r). This procedure is repeated until every object that satisfies those conditions is selected once. Concerning the conditions, the main idea is not to expand variables that are "equivalent" to already expanded variables, in the sense that they are attached to the same "relevant" concepts. All these notions are now formalized.

Let Σ be a knowledge base and o be an object. The *set of relevant concepts* of o is $\Gamma_o = \{C \mid o : C \in \Sigma,$ and C is of the form $\bot, P, \neg P, \forall r.C'$ or $\exists r.C'\}$.

A set of concepts Γ has a *clash* if, either $\bot \in \Gamma$, or $\{P, \neg P\} \subseteq \Gamma$ for some primitive concept P. When this is not the case, we say Γ is *clash-free*.

Objects o_1 and o_2 are *equivalent*, in which case we write $o_1 \equiv o_2$, if $\Gamma_{o_1} = \Gamma_{o_2}$.

Let Σ be a knowledge base and o be an object occurring in Σ. *To expand* object o means to apply rule ($\exists$-r) once for each existential assertion $o : \exists r.C \in \Sigma$. In other words, if $o : \exists r_1.C_1$, $o : \exists r_2.C_2$, ..., $o : \exists r_k.C_k$, with $k \geq 0$, are the existential assertions for o, the expansion process applies rule ($\exists$-r) k times, creating k distinct variables v_1, v_2, ..., v_k, each one associated with one existential assertion. After this operation, o is an *expanded object*.

Note that an object without existential assertions can also be expanded. In this case, no new variable (and no assertion) is generated.

Now we present the second step of the algorithm that computes the completions. Its output is called a "complete set of partial completions" and is obtained by resuming the completion process with every clash-free precompletion.

Let Σ be an $\mathcal{ALC}$-knowledge base and $\{\Sigma_1, \ldots, \Sigma_n\}$ be the set of clash-free precompletions of Σ, for some $n \geq 1$. A *complete set of partial completions* of Σ is a set $\Theta = \{\overline{\Sigma_1}, \ldots, \overline{\Sigma_n}\}$ of $\mathcal{ALC}_{var}$-knowledge bases, called *partial completions* of Σ, obtained in the following way:

> Select every precompletion Σ_j, one at a time,
> and apply the completion rules under the policy that follows,
> until none of these rules can be applied.
> The knowledge base obtained is $\overline{\Sigma_j}$ (for every $1 \leq j \leq n$).

The policy consists of the following two principles:

1. Expand an object o if and only if:
   (a) no rule but possibly ($\exists$-r) is applicable;
   (b) Γ_o is clash-free; and
   (c) if o is a variable, there is no expanded variable v equivalent to o (either in the knowledge base that is being processed or in any other already computed).
2. Apply rule ($\exists$-r) only when expanding an object.

Note that every individual i occurs both in every precompletion and in every partial completion. However, in general, the corresponding sets of assertions

containing i are not all equal. Fot this reason, we shall consider that different "instances" of an individual are different objects.

It is easy to verify that no $\mathcal{ALC}$-assertion is generated in this second step. As an immediate consequence, different precompletions originate different partial completions, and the set of clash-free precompletions has the same cardinality as any complete set of partial completions.

Notice also that this algorithm is not deterministic. For instance, the order in which precompletions are selected was not specified and it can be shown that it is not irrelevant. Nevertheless (see Proposition 6, below), any output is equivalent from the point of view of the automaton we want to build.

Lemma 4. *Let Σ be an $\mathcal{ALC}$-knowledge base. The algorithm that computes a complete set of partial completions of Σ always terminates.*

Once more, the algorithm blindly generated the alternatives offered by rule ($\sqcup_v$-r) and some of the resulting combinations of assertions are contradictory, which means that they do not correspond to any element of any model. So, our next step is to "prune" complete sets of partial completions.

Let $\{\overline{\Sigma_1}, \ldots, \overline{\Sigma_n}\}$ be a complete set of partial completions of an $\mathcal{ALC}$-knowledge base Σ, with $n \geq 1$, v be a variable, and $o : \exists r.C$ be an existential assertion of $\overline{\Sigma_j}$, for some $1 \leq j \leq n$. We say that v was *generated* by $o : \exists r.C$ if:

- v was created by applying rule ($\exists$-r) to $o : \exists r.C$; or
- v was created by applying rule ($\sqcup_v$-r) and the variable v' that gave rise to v was generated by $o : \exists r.C$.

An object o' occurring in $\overline{\Sigma_k}$, for some $1 \leq k \leq n$, is *contradictory* if:

- $\Gamma_{o'}$ has a clash; or
- o' is an expanded object and there is an existential concept $\exists r.C \in \Gamma_{o'}$ such that all variables generated by $o' : \exists r.C$ are contradictory; or
- o' is a variable equivalent to some contradictory variable.

A partial completion is *admissible* if no individual is contradictory.

The next lemma shows that the admissibility of a partial completion does not depend on the complete set of partial completions that is considered. Actually, it depends on the satisfiability of the precompletion on which the partial completion was based.

Lemma 5. *Let Σ be an $\mathcal{ALC}$-knowledge base, $\{\Sigma_1, \ldots, \Sigma_n\}$ be the set of clash-free precompletions of Σ, for some $n \geq 1$, and $\{\overline{\Sigma_1}, \ldots, \overline{\Sigma_n}\}$ be a complete set of partial completions of Σ. Then, for every $1 \leq j \leq n$, Σ_j is satisfiable if and only if $\overline{\Sigma_j}$ is admissible.*

By Lemma 3 (2.) and Lemma 5, if no partial completion is admissible, one should stop because the initial knowledge base Σ is unsatisfiable. Otherwise, we build an automaton for every satisfiable precompletion Σ_j of Σ. Intuitively, this automaton reflects the "structure" common to all models of Σ_j, which comprises the common existential elements, their primitive properties, and how they relate to each other.

States are sets of expanded and not contradictory objects because, on the one hand, every noncontradictory object that was not expanded is equivalent to some expanded object and, on the other hand, contradictory objects do not correspond to any element of any model. A state $\Omega = \{o_1, \ldots, o_m\}$, with $m \geq 1$, intends to characterize an element, existing in all models of Σ_j, that belongs to all concepts of some Γ_{o_k}, with $1 \leq k \leq m$. So, we associate with Ω the set of primitive and negated primitive concepts that belong to all Γ_{o_l} (for $1 \leq l \leq m$).

The transition relation is based on existential assertions. Given an assertion $o : \exists r.C$, with o an expanded and not contradictory object, there is a nonempty set of variables v generated by $o : \exists r.C$ and not contradictory. Intuitively, any model with an element a that belongs to all concepts of Γ_o, has also an element b, related to a by r, that belongs to all concepts of Γ_v, for v one of those variables. However, we are interested not in these variables v but in the equivalent expanded variables. The next definions formalize these notions.

Let Θ be a complete set of partial completions of a satisfiable $\mathcal{ALC}$-knowledge base. An object o occurring in some partial completion of Θ is called a *good object* if o was expanded and is not contradictory. The set of good objects is denoted by GoodObj.

Besides, let $\exists$-Assertions be the set of existential assertions that concern good objects, i.e., elements of $\exists$-Assertions are of the form $o : \exists r.C$, with $o \in$ GoodObj. The function ρ is defined as follows, where $\wp(\cdot)$ denotes the powerset:

$$\rho : \exists\text{-Assertions} \longrightarrow \wp(\text{GoodObj}) \setminus \emptyset$$
$$\rho(o : \exists r.C) = \{v \in \text{GoodObj} \mid (\exists v')\ v' \text{ was generated by } o : \exists r.C,\ v' \equiv v\}.$$

Let Σ be a satisfiable $\mathcal{ALC}$-knowledge base, Σ_j be a satisfiable precompletion of Σ, and Θ be a complete set of partial completions of Σ. Let us assume that every individual $i \in$ Ind is represented by the object i^j in the admissible partial completion of Θ that was based on Σ_j. Consider the finite automaton $A = (Z, z, T, \delta, F)$, where:

$Z = \wp(\text{GoodObj})$; % Set of states.
$z = \emptyset$; % Initial state.
$F = Z \setminus \{z\}$; % Set of final states.
$T = \text{Ind} \cup \text{Rel}^{\diamond}$; and % Input alphabet.
δ is the least relation such that: % Transition relation.

- $(z, i, \{i^j\}) \in \delta$ if $i \in$ Ind;
- $(\{i_1^j\}, r, \{i_2^j\}) \in \delta$ if $i_1, i_2 \in$ Ind and $\Sigma \vdash r(i_1, i_2)$; *and*
- $(\{o_1, \ldots, o_m\}, r, \bigcup_{1 \leq k \leq m} \rho(o_k : \exists r_k.C_k)) \in \delta$
  if $m \geq 1$ and $(\forall 1 \leq k \leq m)\ \exists r_k.C_k \in \Gamma_{o_k}$, $r \subseteq r_k$,

and the following function π, which associates a set of primitive and negated primitive concepts with every final state of A:

$$\pi(\Omega) = \{C \in \textstyle\bigcap_{o \in \Omega} \Gamma_o \mid C \text{ is of the form } P \text{ or } \neg P, \text{ for some } P \in \text{Prim}\}.$$

The pair $\mathcal{A} = (A, \pi)$ is an *automaton* for Σ_j, and $\pi(\Omega)$ is the *property* of Ω.

Our next step is to build an automaton for the initial knowledge base Σ. Basically, this is done by "intersecting" the automata for the satisfiable precompletions of Σ. Since an automata for a satisfiable precompletion of Σ can be viewed as capturing the "structure" common to all models of the precompletion and, by Lemma 3 (1.), any model of Σ is a model of some of its satisfiable precompletions, the intersection of all those automata gives the "structure" common to all models of Σ.

Let Σ be a satisfiable $\mathcal{ALC}$-knowledge base, $\Sigma_1, \ldots, \Sigma_n$ be the satisfiable precompletions of Σ, for some $n \geq 1$, and $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be automata for $\Sigma_1, \ldots, \Sigma_n$, respectively. Suppose $\mathcal{A}_j = ((Z_j, z_j, T, \delta_j, F_j), \pi_j)$, for every $1 \leq j \leq n$. Consider the finite automaton $A = (Z_1 \times \cdots \times Z_n, <z_1, \ldots, z_n>, T, \delta, F_1 \times \cdots \times F_n)$, where

$$\delta = \{(<\Omega_1, \ldots, \Omega_n>, \tau, <\Omega'_1, \ldots, \Omega'_n>) \mid (\forall 1 \leq j \leq n)\ (\Omega_j, \tau, \Omega'_j) \in \delta_j\},$$

and the function $\pi(<\Omega_1, \ldots, \Omega_n>) = \bigcap_{1 \leq j \leq n} \pi_j(\Omega_j)$, defined for every final state of A. The pair $\mathcal{A} = (A, \pi)$ is called an *automaton* for Σ, and $\pi(\Omega)$ is the *property* of state Ω.

Proposition 6. *There is a unique automaton (up to isomorphism) for every satisfiable $\mathcal{ALC}$-knowledge base, which can be computed in finite time.*

This construction gives rise to automata that may not be in "reduced" form, in the sense that they may have useless states and transitions. We have already developed an algorithm for reducing this type of automata, but it will not be described in this paper because of space constraints.

6.2 The Query-Answering Algorithm

Let Σ be a satisfiable knowledge base, $\mathcal{A}$ be an automaton for Σ, and Q be a query. The query-answering algorithm performs two steps. First, we compute the "interpretation" of Q in $\mathcal{A}$, as if this structure was a kind of "least model", representing what is true in all models of Σ. In this context, the set of final states plays the role of the domain of $\mathcal{A}$, the interpretation of a primitive concept P is the set of final states whose property contains P, and the interpretation of a primitive relation s is the set of pairs (Ω_1, Ω_2) such that $(\Omega_1, s, \Omega_2) \in \delta$. The main aspect in which this interpretations differs from the usual ones is in the way we interpret queries of the form $\neg P$. Instead of being the complement of the interpretation of P, it is the set of final states whose property contains $\neg P$. For this reason, there may be states that do not belong to the interpretation of P nor to the interpretation of $\neg P$ and, in this sense, $\mathcal{A}$ should be seen as a "partial interpretation".

Let us now explain the second step. Intuitively, a state Ω characterizes an individual i if $(z, i, \Omega) \in \delta$, and characterizes an element that is related to i through the relation $r_1 \circ \cdots \circ r_n$ (with $n \geq 1$) if there are states Ω_0, Ω_1, ..., Ω_{n-1} such that (z, i, Ω_0), $(\Omega_0, r_1, \Omega_1)$, ..., $(\Omega_{n-1}, r_n, \Omega) \in \delta$. Therefore, to obtain the answer, we simply need to compute the language accepted by the finite automaton when the set of final states is the one computed in the first phase.

Let $\mathcal{A} = ((Z, z, T, \delta, F), \pi)$ be an automaton and Q be a query. The set of *states satisfying* Q, denoted by $\text{States}_{\mathcal{A}}(Q)$, is defined by induction on the structure of Q in the following way:

$$\begin{array}{ll}
\text{States}_{\mathcal{A}}(\quad \top \quad) & = F \\
\text{States}_{\mathcal{A}}(\quad \bot \quad) & = \emptyset \\
\text{States}_{\mathcal{A}}(\quad P \quad) & = \{\Omega \in F \mid P \in \pi(\Omega)\} \\
\text{States}_{\mathcal{A}}(\quad \neg P \quad) & = \{\Omega \in F \mid \neg P \in \pi(\Omega)\} \\
\text{States}_{\mathcal{A}}(Q_1 \sqcap Q_2) & = \text{States}_{\mathcal{A}}(Q_1) \cap \text{States}_{\mathcal{A}}(Q_2) \\
\text{States}_{\mathcal{A}}(\quad \exists r.Q \quad) & = \bigcup_{\Omega' \in \text{States}_{\mathcal{A}}(Q)} \{\Omega \mid (\Omega, r, \Omega') \in \delta\}
\end{array}$$

Let $\mathcal{A} = ((Z, z, T, \delta, F), \pi)$ be an automaton and $X \subseteq F$ be a set of final states. The *language accepted* by X, represented by $\mathcal{L}_{\mathcal{A}}(X)$, is a regular expression that denotes the language accepted by the finite automaton (Z, z, T, δ, X).

Let Σ be a satisfiable $\mathcal{ALC}$-knowledge base, $\mathcal{A}$ be an automaton for Σ, and Q be a query. The *computed answer* to Q is

$$\text{ComputedAnswer}(\Sigma, Q) = \mathcal{L}_{\mathcal{A}}(\text{States}_{\mathcal{A}}(Q)).$$

Assuming, as it is usual, that two regular expressions are equal if they denote the same language, it follows from Proposition 6 that computed answers are unique.

7 Answers and Computed Answers

As we have already mentioned, our algorithm is not complete in the general case. In order to identify the situations in which the declarative and the operational semantics coincide, we present the following definition.

Let Σ be an $\mathcal{ALC}$-knowledge base. Individuals i and i' are *related through a path of length* $n \geq 1$, if there are n relations, $r_1, r_2, \ldots, r_n$, and $n-1$ individuals, $i_1, i_2, \ldots, i_{n-1}$, such that $\Sigma \vdash r_1(i, i_1)$, $\Sigma \vdash r_2(i_1, i_2)$, $\ldots$, $\Sigma \vdash r_n(i_{n-1}, i')$. In addition, any individual is related to itself through a path of length zero.

Proposition 7. *Let Σ be a satisfiable $\mathcal{ALC}$-knowledge base, and Q be a query.*

1. *(Soundness) ComputedAnswer$(\Sigma, Q) \subseteq$ Answer(Σ, Q).*
2. *(Completeness) If either there are no individuals i and i' related through two paths of different length, or there are no existential concepts in Σ, then Answer$(\Sigma, Q) \subseteq$ ComputedAnswer(Σ, Q).*

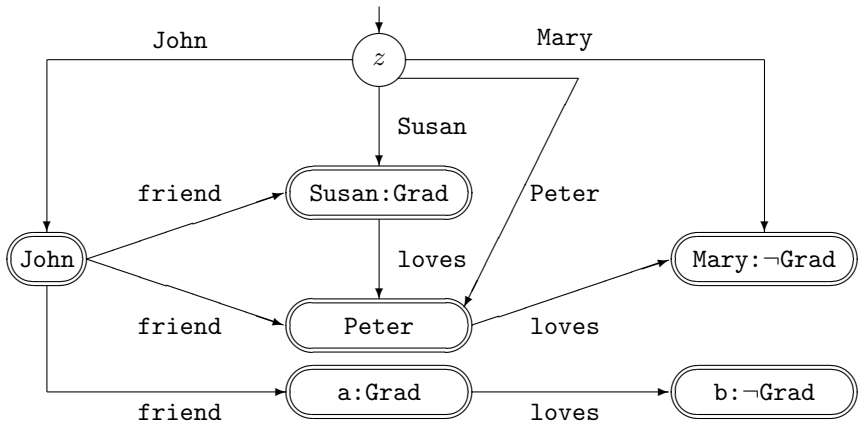
Let us see an example based on the knowledge base Σ_f, borrowed from [14]:

$\texttt{Susan} : \texttt{Grad}$	$\texttt{friend}(\texttt{Jonh}, \texttt{Susan})$	$\texttt{loves}(\texttt{Susan}, \texttt{Peter})$
$\texttt{Mary} : \neg\texttt{Grad}$	$\texttt{friend}(\texttt{Jonh}, \texttt{Peter})$	$\texttt{loves}(\texttt{Peter}, \texttt{Mary})$

Notice `John` and `Peter` are related through paths of length one and two, but since there are no existential concepts in Σ_f, our algorithm is complete. A (reduced) automaton for Σ_f is shown below. First, let us consider the query

$$Q_f^1 = \exists \texttt{friend}.(\texttt{Grad} \sqcap \exists \texttt{loves}.\neg\texttt{Grad}),$$

which asks for those elements having a graduate friend that loves someone who is not graduate. In this case, both the classical and our answers are {John}. Although the knowledge base does not specify if Peter is graduate or not, one of these situations holds in each model of Σ_f and the answer is justified by case analysis. If Peter is graduate, Jonh has a graduate friend (Peter) that loves Mary, a nongraduate. Otherwise, the graduate friend of John is Susan, who loves someone nongraduate (Peter).



Now suppose we asked for those elements that are graduate and love someone who is not graduate,

$$Q_f^2 = \texttt{Grad} \sqcap \exists \texttt{loves}.\neg\texttt{Grad}.$$

Our answer would be "a friend of John", i.e., John friend, whereas the classical answer would be the empty set because no (fixed) individual satisfies both constraints in all models.

Notice that disjunction and universal quantification cannot be treated in a similar way. If the interpretation of $\texttt{Grad} \sqcup \neg\texttt{Grad}$ in the automaton was the union of the interpretations of the disjuncts, the first step of the query-answering algorithm would return the set of states $\{\texttt{Susan}, \texttt{Mary}, a, b\}$ and, for instance, Jonh would not belong to the computed answer. Besides, interpreting universal concepts in the usual way, the computed answer to $\forall\texttt{loves}.\neg\texttt{Grad}$ would comprise elements like Peter and Mary, which do not belong to the answer.

8 Conclusion and Future Work

This work has shown that the definition of answer may be extended in order to refer to elements that have no explicit representation in the knowledge base or, in other words, that answers in terminological languages need not be restricted to the finite set of individuals only. Furthermore, since the query-answering algorithm works on a preprocessing of the knowledge base (a finite automaton), once this information has been collected and synthesized, answers can be efficiently computed.

Concerning future work, there are two topics we are interested in investigating immediately. First of all, in order to achieve completeness in the general case, we are trying to characterize an automaton that always captures all existential elements. Besides, we intend to study the complexity of our methods.

Another field we would like to address concerns the integration of default logic into terminological representation systems. In [4], Baader and Hollunder give examples proving that, when existential elements are treated by Skolemization, the consequences of an open default theory may depend on the syntax of the knowledge base. It turns out that for the equivalent knowledge bases presented in those examples, our (reduced) automata are equal – and correspond to our intuitions about the existential elements they represent.

References

1. F. Baader, H-J. Bürckert, B. Nebel, W. Nutt, G. Smolka. *On the Expressivity of Feature Logics with Negation, Functional Uncertainty, and Sort Equations.* DFKI Research Report RR-91-01, DFKI, Germany, 1991.
2. F. Baader, B. Hollunder. KRIS: Knowledge Representation and Inference System. *SIGART Bulletin*, 2(3):8–14, 1991.
3. F. Baader, B. Hollunder. A Terminological Knowledge Representation System with Complete Inference Algorithms. In *Proc. PDK'91*, LNAI 567, Springer, 1991.
4. F. Baader, B. Hollunder. Embedding Defaults into Terminological Knowledge Representation Formalisms. In *Proc. KR'92*, Morgan Kaufmann, 1992.
5. M. Buchheit, F. Donini, A. Schaerf. Decidable Reasoning in Terminological Knowledge Representation Systems. In *Proc. IJCAI'93*, Morgan Kaufmann, 1993.
6. W. Cohen, H. Hirsh. Learning the CLASSIC Description Logic: Theoretical and Experimental Results. In *Proc. KR'94*, Morgan Kaufmann, 1994.
7. F. Donini, M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt. Adding Epistemic Operators to Concept Languages. In *Proc. KR'92*, Morgan Kaufmann, 1992.
8. F. Donini, M. Lenzerini, D. Nardi, W. Nutt, A. Schaerf. Queries, Rules and Definitions as Epistemic Sentences in Concept Languages. In *Foundations of Knowledge Representation and Reasoning*, LNAI 810, Springer-Verlag, 1994.
9. M. Lenzerini, A. Schaerf. Querying Concept-based Knowledge Bases. In *Proc. PDK'91*, LNAI 567, Springer-Verlag, 1991.
10. M. Mamede, L. Monteiro. Towards Complete Answers in Concept Languages. In *Proc. EPIA'93*, LNAI 727, Springer-Verlag, 1993.
11. M. Mamede, L. Monteiro. $\mathcal{ALC}$rn: A Decidable Terminological Language with Role Negation. In *Proc. 6th Australian Joint Conference on AI*, World Scientific Publishing, 1993.
12. B. Nebel. *Reasoning and Revision in Hybrid Representation Systems.* LNAI 422, Springer-Verlag, 1990.
13. C. Peltason. The BACK System – An Overview. *SIGART Bulletin*, 2(3):114–119, 1991.
14. A. Schaerf. On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification. In *Proc. ISMIS'93*, LNAI 689, Springer-Verlag, 1993.
15. M. Schmidt-Schauß, G. Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

This article was processed using the LaTeX macro package with LLNCS style