Journal of Intelligent Information Systems, 2, 265–278 (1993)

On the Complexity of the Instance Checking Problem in Concept Languages with Existential Quantification

ANDREA SCHAERF aschaerf@assi.ing.uniromal.it
Dipartimento di Informatica e Sistemistica Università di Roma "la Sapienza" Via Salaria 113, 00198 Roma, Italia

Abstract. Most of the work regarding complexity results for concept languages consider subsumption as the prototypical inference. However, when concept languages are used for building knowledge bases including assertions on individuals, the basic deductive service of the knowledge base is the so-called instance checking, which is the problem of checking if an individual is an instance of a given concept. We consider a particular concept language, called $\mathcal{ALE}$ and we address the question of whether instance checking can be really solved by means of subsumption algorithms in this language. Therefore, we indirectly ask whether considering subsumption as the prototypical inference is justified. Our analysis, carried out considering two different measure of complexity, shows that in $\mathcal{ALE}$ instance checking is strictly harder than subsumption. This result singles out a new source of complexity in concept languages, which does not show up when checking subsumption between concepts.

Keywords: concept languages, terminological languages, description logics, computational complexity, query answering

1. Introduction

Concept description languages (also called terminological languages or *concept languages* have been introduced with the aim of providing a simple and well-established first order semantics to capture the meaning of the most popular features of the structured representations of knowledge (see for example (Levesque and Brachman, 1987; Nebel, 1990a)).

In concept languages, concepts are used to represent classes as sets of individuals, and roles are binary relations used to specify their properties or attributes.

It is a common opinion that subsumption checking (i.e., checking whether a concept represent necessarily a subset of the other) is the central reasoning task in concept languages. This has motivated a large body of research on the problem of subsumption checking in different concept languages (e.g., (Brachman and Levesque, 1984; Donini et al., 1991: Nebel, 1988; Schmidt-Schauß, 1989;

This work has been supported by the ESPRIT Basic Research Action N.6810-COMPULOG 2 and by the Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo of the CNR (Italian Research Council).

Schmidt-Schauß and Smolka, 1991)). However, if concept languages are to be used for building knowledge bases including assertions on individuals, the basic deductive service of the knowledge base is the so-called *instance checking*, which is the problem of checking whether a set of assertions implies that a given individual is an instance of a given concept.

In this paper we address the question of whether instance checking can be easily solved by means of subsumption algorithms and whether considering subsumption as the prototypical inference is justified.

The outcome of the analysis is that the answer to the question in the general case is *no* and that there are cases where instance checking is strictly harder than subsumption. In fact, the instance checking problem for the considered language ($\mathcal{ALE}$) turns out to be of higher complexity than the subsumption problem. This result is all the more interesting since $\mathcal{ALE}$ is *not* some artificial language (just defined for the purpose of showing that the complexity of subsumption may differ from the one of instance checking) but a rather natural language which has been investigated before.

This result singles out a new source of complexity in concept languages, which does not show up when checking subsumption between concepts, and is due to the presence of individuals in the knowledge base. A practical implication of this fact is that any actual deduction procedure for reasoning on structured knowledge representation systems cannot be based solely on subsumption checking, but has to embed some reasoning mechanisms that are not easily reducible to subsumption. This fact must be considered when designing the deductive services of a system for the development of applications based on concept languages. In particular, this may have an impact on the design of the so-called hybrid architectures whose purpose is to provide complex deduction capabilities as a result of the interaction between a terminological and an assertional component. In fact, it may happen that this kind of organization leads to implementations that are computationally more demanding.

In our analysis we have assumed we are dealing with complete reasoners. On the contrary, most of the existing systems (e.g., LOOM (MacGregor, 1991), CLASSIC (Patel-Schneider et al, 1991). BACK (Peltason, 1991)), use incomplete reasoners[1] (except for $\mathcal{KRIS}$ (Baader and Hollunder, 1991)) and in these systems a careful analysis of the relationship between subsumption and instance checking is lacking. Anyway, in our opinion (supported also by other researchers, e.g., see (Schmidt-Schauß and Smolka, 1991)), the study of complete methods gives more insight on the structure of the problem and it is useful also for the development of better incomplete reasoners.

Furthermore, we measure the complexity of instance checking by the measures suggested in (Vardi, 1982): *Data complexity* (i.e., the complexity with respect to the size of the knowledge base) and *combined complexity* (i.e., with respect to both the size of the knowledge base and the size of the concept representing the query).[2] Another result of our analysis is that there are cases where the complexity obtained using the two measures differs substantially. This result

proves that the distinction is necessary and must be taken into account in order to understand the behavior of the system in practical cases.

The paper is organized as follows. In Section 2 we provide some preliminaries on concept languages, in particular on the language $\mathcal{ALE}$. In Section 3, we deal with the problem of instance checking in $\mathcal{ALE}$. Discussion and conclusions are drawn in Section 4.

2. Preliminaries

In this section we present the basic notions regarding the concept language $\mathcal{ALE}$, knowledge bases built up using $\mathcal{ALE}$, the instance checking problem and the complexity measures used in the paper.

2.1. The Language $\mathcal{ALE}$

We consider the language $\mathcal{ALE}$ (Donini et al, 1992a; Schmidt-Schauß and Smolka, 1991) which is an extension of the basic concept language $\mathcal{FL}^-$ introduced in (Brachman and Levesque, 1984). Besides the constructors of $\mathcal{FL}^-$, $\mathcal{ALE}$ includes qualified existential quantification on roles and complementation of primitive concepts.

Given an alphabet of primitive concept symbols $\mathcal{A}$ and an alphabet of role symbols $\mathcal{R}$ and two special concept symbols $\top$ and $\bot$, *$\mathcal{ALE}$-concepts* (denoted by the letters C and D) are built up by means of the following syntax rule (where A denotes a primitive concept and R denotes a *role*, that in $\mathcal{ALE}$ is always primitive)

$$
\begin{array}{rcrl}
C, D & \longrightarrow & A & \mid \text{(primitive concept)} \\
 & & \top & \mid \text{(top)} \\
 & & \bot & \mid \text{(bottom)} \\
 & & \neg A & \mid \text{(primitive complement)} \\
 & & C \sqcap D & \mid \text{(intersection)} \\
 & & \forall R.C & \mid \text{(universal quantification)} \\
 & & \exists R.C & \text{(qualified existential quantification)}
\end{array}
$$

We have chosen the language $\mathcal{ALE}$ for several reasons. On one hand, it is rich enough to express a significant class of assertions, as we will see in the examples. In addition, it is suitable to express concepts representing meaningful queries. In particular, due to the qualified existential quantification it is possible to express queries requiring some form of navigation in the knowledge base through the assertions on roles. For example, the query: "find the individuals who have a

friend whose child is a doctor" can be expressed as ∃FRIEND.∃CHILD.Doctor.

On the other hand, $\mathcal{ALE}$ avoids other expressive constructors (such as disjunction of concepts) which introduce other sources of complexity and whose treatment is out of the scope of this paper. In addition, $\mathcal{ALE}$ is a sublanguage of all the cited existing systems (except of CLASSIC). In fact, all its constructors can be expressed in such languages.

An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of the set $\Delta^{\mathcal{I}}$ (the *domain* of $\mathcal{I}$) and a function $\cdot^{\mathcal{I}}$ (the *interpretation function* of $\mathcal{I}$) that maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following equations are satisfied:

$$\begin{aligned}
\top^{\mathcal{I}} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}} &= \emptyset \\
(\neg A)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \backslash A^{\mathcal{I}} \\
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow d_2 \in C^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{d_1 \mid \exists d_2 : (d_1, d_2) \in R^{\mathcal{I}} \wedge d_2 \in C^{\mathcal{I}}\}
\end{aligned}$$

An interpretation $\mathcal{I}$ is a *model* for a concept C if $C^{\mathcal{I}}$ is nonempty. A concept is *satisfiable* if it has a model and *unsatisfiable* otherwise.

We say C is *subsumed* by D, written as $C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$, and C is *equivalent* to D, written $C \equiv D$, if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every interpretation $\mathcal{I}$.

Example 2.1. Consider the following two $\mathcal{ALE}$-concepts

$$\exists \texttt{CHILD}.(\texttt{Female} \sqcap \texttt{Graduate})$$
$$(\exists \texttt{CHILD}.\texttt{Female}) \sqcap (\forall \texttt{CHILD}.\texttt{Graduate}).$$

The first one denotes the individuals having at least one graduate daughter. The second one denotes the individuals having at least one daughter and having only graduate children. It is easy to see that they are both satisfiable. Moreover the first one subsumes the second, in fact the second requires more conditions to be satisfied than the first, in particular, besides the existence of a graduate daughter, it requires all the other possible children to be graduates.

The following concept is instead not satisfiable and it is therefore trivially subsumed by both the others.

$$(\exists \texttt{CHILD}.\neg\texttt{Female}) \sqcap (\forall \texttt{CHILD}.\texttt{Female})$$

2.2. *Knowledge bases*

The construction of knowledge bases using concept languages is realized by permitting concept and role expressions to be used in assertions on individuals.

Let $\mathcal{O}$ be an alphabet of symbols denoting individuals, an *$\mathcal{ALE}$-membership assertion* (or simply *assertion*) is a statement of one of the forms:

$$C(a) \quad \text{or} \quad R(a, b)$$

where C is an $\mathcal{ALE}$-concept, R is a role, and a, b are individuals in $\mathcal{O}$.

In order to assign a meaning to the assertions, the interpretation function $\cdot^{\mathcal{I}}$ is extended to individuals in such a way that $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for each individual $a \in \mathcal{O}$ and $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$ (Unique Name Assumption). The meaning of the above assertions is now straightforward: if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is an interpretation, $C(a)$ is satisfied by $\mathcal{I}$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and $R(a, b)$ is satisfied by $\mathcal{I}$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

A set Σ of $\mathcal{ALE}$-assertions is called an *$\mathcal{ALE}$-knowledge base*. An interpretation $\mathcal{I}$ is said to be a *model* of Σ if every assertion of Σ is satisfied by $\mathcal{I}$. Σ is said to be *satisfiable* if it admits a model. We say that Σ *logically implies* an assertion α (written $\Sigma \models \alpha$) if α is satisfied by every model of Σ.

In the so-called terminological systems, the knowledge base also includes an intentional part, called *terminology*, expressed in terms of concept definitions. However, almost all implemented systems assume that such definitions are acyclic, i.e., in the definition of concept C no reference (direct or indirect) to C itself may occur (see Nebel, 1991,) for a discussion on the semantics of terminological cycles). It is well known that any reasoning process over knowledge bases comprising an acyclic terminology can be reduced to a reasoning process over a knowledge base with an empty terminology, in particular by substituting in the assertions every concept name with the corresponding definition. In Nebel (1990b) the complexity of that reduction is analyzed. The result obtained is that, even though the reduction is not polynomial in the worst case, it can be done in polynomial time under reasonable assumptions. For this reason, in our analysis we do not take into account terminologies and, therefore, we conceive a knowledge base as just a set of $\mathcal{ALE}$-assertions.

DEFINITION 2.1. We call *instance checking in $\mathcal{ALE}$* the following problem: given an $\mathcal{ALE}$-knowledge base Σ, an $\mathcal{ALE}$-concept D, and an individual a in $\mathcal{O}$, check if $\Sigma \models D(a)$ holds.

Notice that instance checking is a basic tool for more complex services. For example, using instance checking it is possible to solve the so-called *retrieval* problem: given a knowledge base Σ and a concept D, find all the individuals that are instances of D, i.e., find the set $\{a \in \mathcal{O} | \Sigma \models D(a)\}$. Retrieval can be performed simply by iterating the instance checking problem for all the individuals in Σ.

Example 2.2. Let Σ_1 be the following $\mathcal{ALE}$-knowledge base:

$$\begin{aligned}\Sigma_1 = \{&\texttt{CHILD(john, mary)},\\ &\texttt{(}\forall\texttt{CHILD.}\neg\texttt{Graduate)(john)},\\ &\texttt{(Female}\sqcap\exists\texttt{CHILD.Graduate)(mary)}\}\end{aligned}$$

Σ_1 states that Mary is a child of John, John has only nongraduate children and Mary is a female and she has a graduate child. It is easy to verify that Σ_1 is satisfiable. Notice also that the addition of the assertion `Graduate(mary)` to Σ_1 would make the knowledge base unsatisfiable; in fact, due to the first two assertions, $\Sigma_1 \models \neg$`Graduate(mary)`.

2.3. Complexity measures

In next section we will provide some complexity result for instance checking in $\mathcal{ALE}$. For this reason we introduce here the complexity measures we are going to use in next section.

DEFINITION 2.2. Given an $\mathcal{ALE}$-knowledge base Σ, an $\mathcal{ALE}$-concept D, an individual a in $\mathcal{O}$, and denoting with $|\Sigma|$ the size of Σ and with $|D|$ the size of D, we call:

- *data complexity* of instance checking in $\mathcal{ALE}$ the complexity of checking if $\Sigma \models D(a)$ with respect to $|\Sigma|$
- *expression complexity* of instance checking in $\mathcal{ALE}$ the complexity of checking if $\Sigma \models D(a)$ with respect to $|D|$
- *combined complexity* of instance checking in $\mathcal{ALE}$ the complexity of checking if $\Sigma \models D(a)$ with respect to $|\Sigma|$ and $|D|$

The combined complexity is the one usually considered in the literature in concept languages. It is a good measure of the actual cost of instance checking in the cases in which the knowledge base and the query are comparable in size.

On the other hand, the data complexity is important whether the size of the query is neglectable with respect to the size of the knowledge base. It is the measure widely accepted in the database community. However, in our case the assumption that the size of the query is neglectable is not always reasonable. In fact, there are cases in which the size of the concepts, after the elimination of the terminology, can reach the size of several lines of code.

Finally, the expression complexity should be taken into account whether the size of the knowledge base is neglectable with respect to the size of the query. Since this assumption generally does not reflect the reality, expression complexity is less important than the other two measures and it will be not considered in this paper.

It is worthwhile to notice that the combined complexity is always higher or equal than both the other two (in the worst-case analysis). In fact, the complexity with respect to $|\Sigma|$ and $|D|$ obviously includes as particular cases the possibilities that either $|\Sigma|$ or $|D|$ is small with respect to the other.

3. Instance checking in $\mathcal{ALE}$

In (Donini et al., 1992a) it is shown that checking the subsumption relation between two $\mathcal{ALE}$-concepts is an NP-complete problem and that checking the satisfiability of an $\mathcal{ALE}$-concept is coNP-complete. With regard to instance checking, we know that it is at least as hard as subsumption. In fact, subsumption can be reduced to instance checking in the following way: given two concepts C and D, the subsumption test $C \sqsubseteq D$ is performed by checking whether the knowledge base composed by the single assertion $C(a)$ (where a is any individual) logically implies $D(a)$, i.e., checking if $\{C(a)\} \models D(a)$ holds.

The above result holds for instance checking with respect to the combined complexity. In fact, the complexity of subsumption is measured with respect to the size of both the candidate subsumee and the candidate subsumer, which becomes respectively part of the knowledge base and part of the query.

With regard to the data complexity, we know that a concept C is unsatisfiable if and only if the knowledge base $\{C(a)\}$ implies every assertion, i.e., if $\{C(a)\} \models B(a)$ (where B is any concept). It follows that concept unsatisfiability can be reduced to instance checking with respect to data complexity (the size of B can be obviously fixed). Since concept satisfiability is coNP-complete, concept unsatisfiability is NP-complete. We can therefore conclude that instance checking is NP-hard even with respect to the data complexity. Notice that this result implies the previous one. In fact, as we pointed out at the end of Section 2, the combined complexity is always higher or equal to the data complexity.

In the two sequent subsections we give a more precise characterization of the complexity of instance checking. In Section 3.1 with respect to the data complexity and in Section 3.2 with respect to combined complexity.

3.1. Data complexity of instance checking in $\mathcal{ALE}$

In this section we give a lower and an upper bound for the data complexity of instance checking in $\mathcal{ALE}$. As shown above, instance checking in $\mathcal{ALE}$ is NP-hard. We now prove that it is coNP-hard too. Since subsumption in $\mathcal{ALE}$ is NP-complete, a consequence of this result is that (assuming NP $\neq$ coNP) instance checking for $\mathcal{ALE}$ is strictly harder than subsumption.

This unexpected result shows that the instance checking problem in $\mathcal{ALE}$ suffers from a new source of complexity, which does not show up when checking subsumption between $\mathcal{ALE}$-concepts. This new source of complexity is related to the use of qualified existential quantification in the concept representing the query that makes the behavior of the individuals heavily dependent on the other individuals in the knowledge base. The following example enlights this point.

Example 3.1. Let Σ_2 be the following $\mathcal{ALE}$-knowledge base:

Σ_2 = {FRIEND(john, susan), FRIEND(john, peter),
LOVES(susan, peter), LOVES(peter, mary),
Graduate(susan), ¬Graduate(mary)}

Consider now the following assertion

β = ∃FRIEND.(Graduate ⊓ ∃LOVES.¬Graduate)(john).

Asking whether $\Sigma_2 \models \beta$ means asking whether John has a graduate friend who loves a not graduate person. At the first glance, since Susan and Peter are the only known friends of John, it seems that the answer the query is to be found by checking whether either $\Sigma_2 \models$Graduate⊓∃LOVES.¬Graduate(susan) or $\Sigma_2 \models$Graduate⊓∃LOVES.¬Graduate(peter) is true.

Since $\Sigma_2 \not\models$Graduate(peter) it follows that $\Sigma_2 \not\models$Graduate⊓∃LOVES.¬Graduate(peter), and since $\Sigma_2 \not\models$ ∃LOVES.¬Graduate(susan) it follows that $\Sigma_2 \not\models$Graduate⊓∃LOVES.¬Graduate(susan).

Reasoning in this way would lead to the answer NO. On the contrary, the correct answer to the query is YES, and in order to find it, one needs to reason by *case analysis*. In fact, the query asks if in every model M of Σ_2 there is an individual, say a, such that FRIEND(john, a), Graduate(a) and ∃LOVES.¬Graduate(a) are true in M. Obviously, in every model M of Σ_2, either Graduate(peter) or ¬Graduate(peter) is true. In the first case, it is easy to see that a is simply peter (and the not graduate person he loves is mary), while in the second case a is susan (and the not graduate person she loves is just peter). Therefore, such an individual a exists in every model of Σ_2, and the query gets the answer YES.

Therefore, even if none of the individuals related to the individual john through the role FRIEND is in the condition requested by the query, it happens that the combination of the assertions on the individuals (susan and peter) in the knowledge base is such that in every model one or the other is in that condition.

The previous example shows that, in order to answer to a query involving qualified existential quantification, a sort of case analysis is required. We now show that this kind of reasoning makes instance checking in $\mathcal{ALE}$ coNP-hard with respect to the data complexity.

The proof is based on a reduction from a suitable variation of the propositional satisfiability problem (SAT) to instance checking in $\mathcal{ALE}$. We define 2+2-CNF formula on an alphabet P, a CNF formula F such that each clause of F has exactly four literals: two positive and two negative ones, where the propositional letters are elements of $P \cup \{true, false\}$. Furthermore, we call 2+2-SAT the problem of checking whether a 2+2-CNF formula is satisfiable.

THEOREM 3.1. 2+2-SAT is NP-complete.

Proof. The proof is obtained by a reduction from 3-SAT. Given a 3-CNF formula F we obtain a 2+2-CNF formula F' transforming each clause C of F according

to the following rules:

$$a \vee b \vee c \Longrightarrow (a \vee b \vee \neg d \vee \neg \mathit{true}) \wedge (c \vee d \vee \neg \mathit{true} \vee \neg \mathit{true})$$
$$a \vee b \vee \neg c \Longrightarrow a \vee b \vee \neg c \vee \neg \mathit{true}$$
$$a \vee \neg b \vee \neg c \Longrightarrow a \vee \mathit{false} \vee \neg b \vee \neg c$$
$$\neg a \vee \neg b \vee \neg c \Longrightarrow (\mathit{false} \vee d \vee \neg a \vee \neg b) \wedge (\mathit{false} \vee \mathit{false} \vee \neg c \vee \neg d)$$

where for each clause C of F, d is a new letter in P not appearing in F. It is easy to see that F is satisfiable iff F' is satisfiable.

Given a 2+2-CNF formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where $C_i = L^i_{1+} \vee L^i_{2+} \vee \neg L^i_{1-} \vee \neg L^i_{2-}$, we associate with it an $\mathcal{ALE}$-knowledge base Σ_F and a concept Q as follows. Σ_F has one individual l for each letter L in F, one individual c_i for each clause C_i, one individual f for the whole formula F, plus two individuals *true* and *false* for the corresponding propositional constants. The roles of Σ_F are Cl, P_1, P_2, N_1, N_2, and the only primitive concept is A.

$$\begin{aligned}\Sigma_F = \{&A(\mathit{true}), \neg A(\mathit{false}),\\ &Cl(f, c_1), Cl(f, c_2), \ldots, Cl(f, c_n),\\ &P_1(c_1, l^1_{1+}), P_2(c_1, l^1_{2+}), N_1(c_1, l^1_{1-}), N_2(c_1, l^1_{2-}),\\ &\ldots\\ &P_1(c_n, l^n_{1+}), P_2(c_n, l^n_{2+}), N_1(c_n, l^n_{1-}), N_2(c_n, l^n_{2-})\},\\ Q = \exists Cl.(&\exists P_1.\neg A \sqcap \exists P_2.\neg A \sqcap \exists N_1.A \sqcap \exists N_2.A).\end{aligned}$$

Intuitively, the membership to the extension of A or $\neg A$ corresponds to the truth values *true* and *false* respectively and checking if $\Sigma_F \models Q(f)$ corresponds to checking if in every truth assignment for F there exists a clause whose positive literals are interpreted as false, and whose negative literals are interpreted as true, i.e. a clause that is not satisfied.

LEMMA 3.1. A 2+2-CNF formula F is unsatisfiable if and only if $\Sigma_F \models Q(f)$.

Proof. "$\Rightarrow$" Suppose F is unsatisfiable. Notice first that Σ_F is always satisfiable independently of F. Consider a model $\mathcal{I}$ of Σ_F (which always exists), and let $\delta_{\mathcal{I}}$ be the truth assignment for F such that $\delta_{\mathcal{I}}(l) = \mathit{true}$ if and only if $l^{\mathcal{I}} \notin A^{\mathcal{I}}$, for every letter l. Since F is unsatisfiable, there exists a clause C_i that is not satisfied by $\delta_{\mathcal{I}}$. It follows that the individuals related to c_i through the roles P_1, P_2 are in the extension of $(\neg A)^{\mathcal{I}}$ and the individuals related to c_i through the roles N_1, N_2 are in the extension of $A^{\mathcal{I}}$. Thus $c_i^{\mathcal{I}} \in (\exists P_1.\neg A \sqcap \exists P_2.\neg A \sqcap \exists N_1.A \sqcap \exists N_2.A)^{\mathcal{I}}$, and consequently $f^{\mathcal{I}} \in Q^{\mathcal{I}}$. Therefore, since this argument holds for every model $\mathcal{I}$ of Σ_F, we can conclude that $\Sigma_F \models Q(f)$.

"$\Leftarrow$" Suppose F is satisfiable, and let δ be a truth assignment satisfying F. Let $\mathcal{I}_\delta$ be the interpretation for Σ_F defined as follows:

- $A^{\mathcal{I}_\delta} = \{l^{\mathcal{I}_\delta} \mid \delta(l) = \mathit{true}\}$.

- $\rho^{\mathcal{I}_\delta} = \{(a^{\mathcal{I}_\delta}, b^{\mathcal{I}_\delta}) \mid \rho(a, b) \in \Sigma_F\}$ for $\rho = Cl, P_1, P_2, N_1, N_2$.

By definition of $\rho^{\mathcal{I}_\delta}$ (for $\rho = Cl, P_1, P_2, N_1, N_2$) we see that $\mathcal{I}_\delta$ is a model of Σ_F. On the other hand, since F is satisfiable, for every clause in F there exists either a positive literal interpreted as *true* or a negative one interpreted as *false*. It follows that, for every individual c_i, there exists either a role (P_1 or P_2) such that the object related to c_i by means of that role is in the extension of A or there exists a role (N_1 or N_2) such that the object related to c_i by means of that role is in the extension of $\neg A$; hence $f^{\mathcal{I}_\delta} \notin Q^{\mathcal{I}_\delta}$. Therefore, we can conclude that $\Sigma_F \not\models Q(f)$.

THEOREM 3.2. Instance checking in $\mathcal{ALE}$ is coNP-hard in the size of the knowledge base.

Proof. The claim follows from Theorem 3.1 and Lemma 3.1 and from the fact that, given a 2+2-CNF formula F, Q is fixed independently of F and Σ_F can be computed in polynomial time with respect to the size of F.

The above reduction shows that the coNP-hardness arises even if the knowledge base is expressed using a simple language. This implies that, in order to obtain intractability, it is sufficient to enrich only the query language with the qualified existential quantification, keeping a simple and tractable assertional language. This result is in contrast with the result reported in (Lenzerini and Schaerf, 1991a): In that paper, a polynomial instance checking algorithm is provided for a knowledge base in the language $\mathcal{AL}$ ($\mathcal{AL}$ extends $\mathcal{FL}^-$ with primitive complements) using the query language $\mathcal{QL}$ (which is an extension of $\mathcal{ALE}$). Unfortunately, as pointed out in (Lenzerini and Schaerf, 1991b), while that algorithm is sound, it is in fact not complete. In particular, it answers NO the query β to the knowledge base Σ_2 of Example 3.1.

Since the language $\mathcal{ALE}$ involves primitive complements and the above reduction makes use of them it may seem that the coNP-hardness arises from the interaction of qualified existential quantification with the primitive complements. On the contrary, we are able to show that the coNP-hardness is caused by the qualified existential quantification alone. In fact, if we consider the language $\mathcal{FLE}^-$ ($\mathcal{FL}^-$ + qualified existential quantification), we are able to prove that instance checking in $\mathcal{FLE}^-$ is coNP-hard too. The intuition is that in $\mathcal{FLE}^-$ it is possible to require a reasoning by case analysis too. In particular, this is done by considering two $\mathcal{FLE}^-$-concepts of the form $\exists R.\top$ and $\forall R.C$ (instead of A and $\neg A$ in $\mathcal{ALE}$), and exploiting the fact that their interpretation covers the entire domain, i.e., $\exists R.\top \sqcup \forall R.C \equiv \top$. More in detail, the same reduction used to prove the coNP-hardness in $\mathcal{ALE}$, can be used for $\mathcal{FLE}^-$ by replacing Q with the $\mathcal{FLE}^-$-concept Q' obtained substituting any occurence of A with $\exists R.\top$ and each occurrence of $\neg A$ with $\forall R.C$. The proof that any 2+2-CNF formula F is unsatisfiable if and only if $\Sigma_F \models Q'(f)$ can be obtained following the same line

of the proof of Lemma 3.1.

In the following we give an upper bound to the data complexity of instance checking in $\mathcal{ALE}$. In particular, we prove that it is in the class Π_2^p. The class Π_2^p, also denoted by $coNP^{NP}$, consists of the problems whose complement can be solved by a nondeterministic polynomial time algorithm exploiting a nondeterministic polynomial oracle. For a discussion on the classes Σ_k^p, Π_k^p, and Δ_k^p see for example (Garey and Johnson, 1979).

LEMMA 3.2. Let Σ be a satisfiable $\mathcal{ALE}$-knowledge base, a be an individual, and D be a $\mathcal{ALE}$-concept. Then checking if $\Sigma \not\models D(a)$ can be done by a nondeterministic algorithm, which runs in polynomial time with respect to $|\Sigma|$ and exploits a nondeterministic polynomial time oracle.

Proof. We know that $\Sigma \models D(a)$ if and only if the knowledge base $\Sigma \cup \{\neg D(a)\}$ is unsatisfiable. In general, $\neg D$ is not an $\mathcal{ALE}$-concept, since it contains the complement of a nonprimitive concept. Therefore, $\Sigma \cup \{\neg D(a)\}$ is not an $\mathcal{ALE}$-knowledge base, but a knowledge base of a more expressive language called $\mathcal{ALC}$ (see (Schmidt-Schauß and Smolka, 1991)), which extend $\mathcal{ALE}$ with general complements. It (Baader and Hollunder, 1991) an algorithm is presented for checking the satisfiability of an $\mathcal{ALC}$-knowledge base. That algorithm consists of a NPTIME procedure which exploits a PSPACE subprocedure. In particular, the task of the subprocedure is to check the satisfiability of an $\mathcal{ALC}$-concept C. However in our case only the assertion $\neg D(a)$ may contain complements of non-primitive concepts and we are interested in the complexity with respect to Σ and not with respect to D, i.e., the size of D can be considered fixed. For this reason, any time we use the subprocedure it is called to check the satisfiability of a concept such that its part containing nonprimitive concepts has fixed size. It follows, according to the algorithm given in (Baader and Hollunder, 1991), that it can be computed in polynomial time by an alternating Turing Machine (TM) such that the alternation between $\wedge$-nodes and $\vee$-nodes exists only down to a fixed depth and below that level only $\vee$-nodes are present. An alternating TM respecting this limitation has a fixed number of $\wedge$-nodes and therefore can be simulated by a non-deterministic TM which works in polynomial time too.

THEOREM 3.3. Instance checking in $\mathcal{ALE}$ is in Π_2^p with respect to the data complexity.

Proof. Easily follows from Lemma 3.2.

Notice that the complexity of instance checking with respect to the data complexity is not completely placed in the complexity hierarchy. We conjecture that it is Π_2^p-complete.

3.2. *Combined complexity of instance checking in $\mathcal{ALE}$*

In this section we show that instance checking in $\mathcal{ALE}$ is PSPACE-complete with respect to the combined complexity. This result is based on the following lemma which states the PSPACE-hardness of the problem and which proof can be found in (Donini et al., 1992c).

LEMMA 3.3. Instance checking in $\mathcal{ALE}$ is PSPACE-hard with respect to the combined complexity.

THEOREM 3.4. Instance checking in $\mathcal{ALE}$ is PSPACE-complete with respect to the combined complexity.

Proof. Lemma 3.3 states the PSPACE-hardness. Since instance checking in $\mathcal{ALE}$ is obviously easier than the PSPACE-complete problem of instance checking in $\mathcal{ALC}$ (see (Baader and Hollunder, 1991)), it follows that instance checking in $\mathcal{ALE}$ is also in PSPACE. Hence instance checking in $\mathcal{ALE}$ is PSPACE-complete.

4. Discussion and conclusions

Table 1 summarizes the complexity of instance checking in $\mathcal{ALE}$ with respect to both knowledge base complexity and combined complexity, together with the previous known result regarding subsumption.

Table 1. Complexity of subsumption and instance checking in $\mathcal{ALE}$.

subsumption	**instance checking data complexity**	**instance checking combined complexity**
NP-complete (Donini et al., 1992a)	NP-hard coNP-hard in Π_2^p	PSPACE-complete

These results single out several interesting properties:

1. Instance checking is *not* polynomially reducible to subsumption, in the general case. As a consequence, an algorithm for instance checking based on subsumption (and classification), should include some other complex reasoning mechanisms.
2. The data complexity and the combined complexity of instance checking in $\mathcal{ALE}$ are in different classes: one in Π_2^p and the other is PSPACE-complete. This fact highlights that, in order to have an actual complexity measure of

the performance of our systems, we must pay attention to which is the crucial data of the application.

3. Reasoning in $\mathcal{ALE}$ suffers from an additional source of complexity which does not show up when checking subsumption, even with respect to the data complexity. This new source of complexity is related to the use of qualified existential quantification in the concept representing the querry which requires some sort of reasoning by case analysis. In particular, due to this source of complexity the instance checking problem (unlike subsumption) cannot be solved by means of one single level of nondeterministic choice.
4. With respect to the combined complexity, $\mathcal{ALE}$ is in the same class (PSPACE) as more complex languages (e.g., $\mathcal{ALC}$, see (Baader and Hollunder, 1991)). Therefore, whenever the expressiveness of $\mathcal{ALE}$ is required, other constructors can be added without any increase of the (worst-case) computational complexity.
5. The source of complexity identified in this paper is not related to the possibility of nesting an arbitrary number of existential and universal quantifiers, unlike the one pointed out in (Donini et al., 1992a) for subsumption in $\mathcal{ALE}$. In fact, it leads to intractability even using only two nested existential quantifiers and no universal ones. This is an important observation, since long chains of nested quantifier seem to do not appear frequently in "real" examples.

With regard to point 3, one may object that if the reasoning process underlying this source of complexity is not in the intuition of the user, as pointed out in Example 3.1, it must be ruled out by the reasoner (and by the semantics). On the contrary, there are cases in which this kind of reasoning seems to agree with the intuition and therefore it must be taken into account. In (Donini et al., 1992b) it is addressed the issue of the appropriate semantics for queries involving existential quantifications. It is shown how, by means of an epistemic operator in the query language, it is possible to achieve a more sophisticated control on the semantics. In particular, the system provides a mechanism for choosing the appropriate one for the particular query.

Acknowledgments

I would like to thank Francesco Donini, Maurizio Lenzerini, and Daniele Nardi for discussion that contributed to the paper and Franz Baader, Enrico Franconi, and Marco Schaerf for many helpful comments on earlier drafts. I also acknowledge Yoav Shoham for his hospitality at the Computer Science Department of the Stanford University, where part of this research has been developed.

Notes

1. CLASSIC is actually complete, but with respect of a non standard semantics.
2. In (Vardi, 1982) it is also considered the so-called *expression complexity* (i.e., the complexity with respect to the size of the concept representing the query).

References

Baader, F. & Hollunder, B. (1991). A Terminological Knowledge Representation System with Complete Inference Algorithm. In *Proc. Workshop on Processing Declarative Knowledge, PDK-19.* Lecture Notes in Artificial Intelligence. Springer-Verlag: New York.

Brachman, R.J. & Levesque, H.J. (1984). The Tractability of Subsumption in Frame-Based Description Languages. In *Proc. 4th Nat. Conf. on Artificial Intelligence AAAI-84.*

Donini, F.M., Hollunder, B., Lenzerini, M., Marchetti Spaccamela, A., Nardi, D. & Nutt, W.. (1992a). The Complexity of Existential Quantification in Concept Languages, *Artificial Intelligence*, 2–3, 309–327.

Donini, F.M., Lenzerini, M., Nardi, D. & Nutt, W. (1991). The Complexity of Concept Languages. In James Allen, Richard Fikes, and Erik Sandewall (Eds.), *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning KR-91*, pages 151–162, Morgan Kaufmann.

Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W. & Schaerf, A. (1992b). Adding Epistemic Operators to Concept Languages. In *Proc. 3rd Int. Conf. On Principles of Knowledge Representation and Reasoning KR-92*, pages 342–353.

Donini, F.M., Lenzerini, M., Nardi, D. & Schaerf, A. (1992c). From Subsumption to Instance Checking, Technical Report 15.92, Dipartimento di Informatica e Sistemistica, Universita di Roma "La Sapienza."

Garey, M.R. & Johnson, D.S. (1979). *Computers and Intractability—A guide to NP-completeness*, Freeman: San Francisco.

Lenzerini, M. & Schaerf, A. (1991a). Concept Languages as Query Languages. In *Proc. 9th Nat. Conf. on Artificial Intelligence AAAI-91.*

Lenzerini, M. & Schaerf, A. (1991b). Querying Concept-Based Knowledge Bases. In *Proc. Workshop on Processing Declarative Knowledge, PDK-91,* Lecture Notes in Artificial Intelligence. Springer-Verlag: New York.

Levesque, H.J. & Brachman, R.J. (1987). Expressiveness and Tractability in Knowledge Representation and Reasoning, *Computational Intelligence*, 3, 78–93.

MacGregor, R. (1991). Inside the LOOM Description Classifier, *SIGART Bulletin,* 2(3); 88–92.

Nebel, B. (1988). Computational Complexity of Terminological Reasoning in BACK, *Artificial Intelligence*, 34(3), 371–383.

Nebel, B. (1990a) *Reasoning and Revision in Hybrid Representation Systems*, Lecture Notes in Artificial Intelligence. Springer-Verlag: New York.

Nebel, B. (1990b). Terminological Reasoning is Inherently Intractable. *Artificial Intelligence*, 43, 235–249.

Nebel, B. (1991). Terminological Cycles: Semantics and Computational Properties. In John F. Sowa (ed.), *Principles of Semantic Networks,* pages 331–361. Morgan Kaufmann.

Patel-Schneider, P.F., McGuiness, D.L., Brachman, R.J., Alperin Resnick, L. & Borgida, A. (1991). The CLASSIC Knowledge Representation System: Guiding Principles and Implementation Rationale. *SIGART Bulletin*, 2(3); 108–113.

Peltason, C. (1991). The BACK System—An Overview, *SIGART Bulletin*, 2(3); 114–119.

Schmidt-Schauß, M. (1989). Subsumption in KL-ONE Is Undecidable. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter Ed, *Proc. 1st Int. Conf. on Principles of Knowledge Representation and Reasoning KR-89.* Morgan Kaufmann.

Schmidt-Schauß, M. & Smolka, G. (1991). Attributive Concept Descriptions with Complements, *Artificial Intelligence*, 48(1), 1–26.

Vardi, M. (1982). The Complexity of Relational Query Languages.In *14th ACM Symp. on Theory of Computing,* pages 137–146.