Evaluating a modular Abox algorithm

Sergio Tessaris, Ian Horrocks and Graham Gough
Department of Computer Science
University of Manchester
Manchester, UK
{tessaris|horrocks|gough}@cs.man.ac.uk

Abstract

This work constitutes an advance in the direction of the development of Description logic systems providing efficient and powerful reasoning in presence of individuals. We present the empirical evaluation of an algorithm for checking the satisfiability of Description logic knowledge bases.

The experiments we performed show that a modular algorithm, which separates terminological and hybrid reasoning, can provide as good performance as an hybrid Description logic reasoner. We are experimenting with the expressive Description logic $\mathcal{SH}f$, which extends the standard Description logic $\mathcal{ALC}$ with transitive roles, role hierarchy, and attributes.

1 Introduction

A description logic (DL) knowledge base (KB) is made up of two parts, a terminological part (the Tbox) and an assertional part (the Abox), each part consisting of a set of axioms. The Tbox asserts facts about *concepts* (sets of objects) and *roles* (binary relations), usually in the form of inclusion axioms, while the Abox asserts facts about *individuals* (single objects), usually in the form of instantiation axioms. For example, a Tbox might contain an axiom asserting that `Human` is subsumed by `Mammal`, while an Abox might contain axioms asserting that `John`, `Peter` and `Bill` are instances of the concept `Human` and that the pairs $\langle$`John`, `Peter`$\rangle$ and $\langle$`Peter`, `Bill`$\rangle$ are instances of the role `Brother`.

Transitive roles play an important part in knowledge representation, and they have been identified as a requirement in application domains concerned with complex physically composed objects (e.g. medical or engineering domains, see Artale et al. [1996], Rector and Horrocks [1997], Sattler [1995]). Several previous papers have also shown the efficacy of using a logic that includes a set of transitive roles, without incurring the cost entailed by the introduction of the transitive closure operator (see Horrocks and Gough [1997], Horrocks and Sattler [1998]).

There has been a great deal of work on the development of reasoning algorithms for expressive DLs (see Baader [1991], Horrocks and Sattler [1999], De Giacomo and Massacci [2000]), but most of this effort has been devoted to terminological reasoning (i.e. with an empty Abox). Important theoretical results have been published on the problem of reasoning with individuals as well as terminology (see for example Buchheit et al. [1993] or De Giacomo and Lenzerini [1996]). However, only a few works, like Horrocks et al. [2000] and the RACE system (see Haarslev and Möller [2000]), tackle the problem of developing practical Abox algorithms. This can be explained by the fact that for many applications Tbox reasoning can be enough, and realistic Aboxes tend to be far bigger than Tboxes. The size problem could lead to practical intractability, given the high complexity of reasoning in expressive DLs.

This paper presents an empirical evaluation of an algorithm which extends the above mentioned work for reasoning within a knowledge representation system which also includes ABox assertions. The approach used is based on an extension of the so–called *precompletion* technique (see Hollunder [1994], Donini et al. [1994]) for reducing the KB satisfiability problem to concept satisfiability. The cited works focused on DL knowledge bases with empty terminologies, and languages without transitive roles, while we generalise the precompletion technique for KBs with general inclusion axioms, transitive roles, and functional roles.

The main idea behind this technique consists of a correctness–preserving process which eliminates specific information regarding dependencies between individuals, while maintaining the consequences of such information. Once these dependencies are eliminated, the assertions

about a single individual can be independently verified, ignoring the fact that an individual is involved. The precompletion, or elimination of the dependencies, is performed by adding new assertions using a set of nondeterministic syntactic rules. Because of the nondeterminism of the rules, many different precompletions can be derived from a single knowledge base, which is satisfiable if and only if at least one of these precompletions is satisfiable.

The algorithm is completely independent from the terminological reasoner which is used for the concept satisfiability test. In this way, an Abox can be easily added to most existing DL systems without re-implementing the whole system. Moreover, optimisation strategies implemented at the terminological level do not adversely interact with the precompletion algorithm.

A further advantage of precompletion is that it confines the exponential blow up of the complexity to terminological reasoning. In fact, the size of precompletions is polynomial in the size of the KB. This may indicate that a system based on precompletion can run with a smaller memory footprint compared to a system based on an extension of Tableaux methods.[1] This, together with the fact that precompletion exploits the connectivity of the knowledge base, suggests that the algorithm can perform well in large and loosely connected KBs.

Due to space restrictions, correctness and completeness proofs for the algorithm are not included in this paper; they can be found in Tessaris [2001].

1.1 $\mathcal{SHf}$ knowledge bases

The DL $\mathcal{SHf}$ extends $\mathcal{ALC}$ with transitively closed primitive roles, role hierarchy and functional restrictions on roles. Valid concept expressions are defined by the syntax:
$C ::= \top \mid \bot \mid A \mid \neg A \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C$
where A is a concept name chosen from a set $\mathcal{CN}$, and R is a role name chosen from a set $\mathcal{RN}$. We consider two disjoint subsets of $\mathcal{RN}$: the set of transitive role names $\mathcal{TRN}$, and the set of functional role names $\mathcal{FRN}$. In addition, we assume a set of individual names $\mathcal{O}$. A standard Tarski style model theoretic semantics provides the meaning for the expressions by means of interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ (see Schmidt-Schauss and Smolka [1991]). The set $\Delta^{\mathcal{I}}$ is the domain, and $\cdot^{\mathcal{I}}$ is an interpretation function which maps each concept name in $\mathcal{CN}$ to a subset of $\Delta^{\mathcal{I}}$, each role name in $\mathcal{RN}$ to a binary relation over $\Delta^{\mathcal{I}}$, and each individual name in $\mathcal{O}$ to a distinct element of $\Delta^{\mathcal{I}}$.[2] In addition, relations corresponding to transitive roles must be transitive, and the interpretation of functional roles must be a partial function. In the rest of the paper capital letters C or D indicate concepts, R or S roles, and small letters o, a or b individual names.

A DL knowledge base is a pair $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a *TBox* and $\mathcal{A}$ an *ABox*. The TBox, or terminology, contains concept axioms of the form $C \sqsubseteq D$ and role axioms of the form $S \sqsubseteq R$, while the ABox contains assertions about a set of individual names $\mathcal{O}$. These assertions are of the form $a{:}C$ or $\langle a, b \rangle{:}R$, where a,b are individual names in $\mathcal{O}$. We say that an interpretation satisfies an axiom $C \sqsubseteq D$ if the interpretation of D includes that of C, and analogously for inclusion between roles. An assertion $a{:}C$ is satisfied if the interpretation of a is an element of the interpretation of C; role assertions like $\langle a, b \rangle{:}R$ talk about the membership of the pair to the relation. A knowledge base Σ is said to be satisfiable if and only if there exists at least one interpretation which satisfies all the assertions in the knowledge base (i.e. the inclusion and membership relations represented in the KB are satisfied).

Role inclusion axioms contain only role names, and if there are cyclical definitions (e.g. $S \sqsubseteq R$ and $R \sqsubseteq S$), all the names involved in the cycle must correspond to the same binary relation in every interpretation (satisfying the axioms). For these reasons, we assume that role axioms are summarized by a partial order $\preceq$ defined over the set of role names.

2 KB satisfiability via precompletion

Let us consider for example a very simple Abox containing only the assertions:

$$\mathcal{A} = \{a{:}\forall R.C, \langle a, b \rangle{:}R, b{:}\neg C\} .$$

The two first assertions can be used to derive the new assertion $b{:}C$; it is easy to realise that $\mathcal{A}$ is satisfiable iff $\mathcal{A}' = \mathcal{A} \cup \{b{:}C\}$ is satisfiable as well (which is not the case, because b cannot be in C and $\neg C$ at the same time). The interesting point is that when we check the satisfiability of $\mathcal{A}'$ we do not need to consider the role assertion, because its effects have been made "explicit" in the new assertion. Therefore we can verify the KB satisfiability by checking the satisfiability of the concepts $\forall R.C$ and $C \sqcap \neg C$ separately.

Intuitively, the precompletion algorithm uses a set of transformation rules to propagate restrictions imposed by universal expressions along the role assertions in the Abox; in the example C is propagated to b because of $\forall R.C$. In the meantime, the propositional parts of the assertions (conjunctions and disjunctions) are expanded. Modal parts of the assertions (e.g. $\exists R.C$) are ignored and left to the terminological reasoner.

[1] This actually depends on the kind of optimisation implemented in the DL system. As we are going to suggest later on, the precompletion technique can be used as a testbed for improving strategies for Tableaux based systems.

[2] We use the unique name assumption for individuals.

In $\mathcal{SH}f$ the choice of whether an universal expression applies to a role assertion is complicated by the interaction between functional roles and role hierachy. For example let us consider a functional role F and two roles R, S included in F (i.e. $R \preceq F$ and $S \preceq F$). From the set of assertions $\{\langle a, b\rangle{:}R, a{:}\exists S.C\}$ we can derive $b{:}C$, because the functionality restriction on F forces any element related to a through R and S to be the same. This phenomenon is extended to "chains" of roles as well; e.g. like $R \preceq F_1$, $S \preceq F_1$, $S \preceq F_2$, and $T \preceq F_2$, with F_1 and F_2 functional roles. Precompletion rules are designed to take account of this interaction and to simplify their structure we introduce a set of binary operators $\cdot \approx_o \cdot$ (one for each individual name o in $\mathcal{O}$). Roughly speaking, two role names R and S are $\cdot \approx_o \cdot$ related if they are functional, and the Abox assertions force the R and S successors of the individual name o to be the same element (see Tessaris [2001] for its formal definition).

$\mathcal{A} \rightarrow_{\sqsubseteq} \{o{:}C\} \cup \mathcal{A}$
if o is in $\mathcal{O}$, $\top \sqsubseteq C$ is in $\mathcal{T}$
and $o{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\sqcup} \{o{:}D\} \cup \mathcal{A}$
if $o{:}C_1 \sqcup C_2$ is in $\mathcal{A}$,
and $D = C_1$ or $D = C_2$
and $o{:}D$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\exists^1} \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\exists R.C$ and $\langle o, o'\rangle{:}S$ are in $\mathcal{A}$,
$R \approx_o S$, and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\forall^+} \{o'{:}\forall R.C\} \cup \mathcal{A}$
if $o{:}\forall T.C$ in $\mathcal{A}$, $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$,
and there is $R \in \mathcal{TRN}$ such that $S \preceq R \preceq T$,
and $o'{:}\forall R.C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\sqcap} \{o{:}C_1,\ o{:}C_2\} \cup \mathcal{A}$
if $o{:}C_1 \sqcap C_2$ is in $\mathcal{A}$,
and either $o{:}C_1$ or $o{:}C_2$ if not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\forall^1} \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\forall R.C$ and $\langle o, o'\rangle{:}S$ are in $\mathcal{A}$,
there is $R' \preceq R$ s.t. $R' \approx_o S$
and $o'{:}C$ is not in $\mathcal{A}$.

$\mathcal{A} \rightarrow_{\forall} \{o'{:}C\} \cup \mathcal{A}$
if $o{:}\forall R.C$ is in $\mathcal{A}$, and $\langle o, o'\rangle{:}S$ is in $\mathcal{A}$,
and $S \preceq R$, and $o'{:}C$ is not in $\mathcal{A}$.

Figure 1: Precompletion rules for $\mathcal{SH}f$

The input to the algorithm consists of a set containing the set of assertions in the Abox, which in this context are called *constraints*. The set of all concepts appearing in constraints like $o{:}C$, and associated to a single individual name is called the label of this name.

The rules in Figure 1 are repeatedly applied to the initial set until either no rule is applicable or a contradictory combination of constraints is detected (a so-called *clash*). The $\sqcup$-rule generates several alternative branches, which are exhaustively explored by means of *backtrack points* where the algorithm restarts in case of a clash detection.

Intuitively, a clash corresponds to a trivially unsatisfiable combination of constraints in the constraint set. When the precompletion process encounters a clash there is no need to continue adding new constraints, the precompletion will be unsatisfiable anyway; therefore the algorithm backtracks to the most recent nondeterministic choice and tries to explore the other possibilities. In $\mathcal{SH}f$ there are two kind of clashes. The first one occurs when two contradictory concepts are in the label of the same individual (e.g. a constraint system containing both $o{:}A$ and $o{:}\neg A$). The second one involves the functional roles, which may force two different individual names to be interpreted as the same element of the domain. This contradicts the unique name assumption, and the precompletion would be unsatisfiable. Due to the interaction between role hierarchy and functionality, the relation $\cdot \approx_o \cdot$ must be taken into account: two constraints $\langle o, a\rangle{:}R$, $\langle o, b\rangle{:}R'$ constitute a clash if $a \neq b$ and $R \approx_o R'$.

If none of the rules is applicable and there are no clashes, then a precompletion has been generated and must be checked for its satisfiability. This is done by gathering all the concept assertions concerning an individual. The concepts are then conjoined, generating a new single concept associated to each single individual. Each of these concepts is then verified by using an external call to a terminological reasoner; if they are all satisfiable then the algorithm terminates with success, otherwise it backtracks as in the case of clash detection. If the process fails to find a satisfiable precompletion after all the nondeterministic branches have been explored, then it terminates with failure.

As presented here the algorithm sounds rather naive and prone to inefficient exploration of the search space. In fact, we need a few "tricks" to avoid a hopelessly slow algorithm. In the following section we will show how to improve the algorithm to obtain acceptable behaviour in typical cases. Note that the theoretical worst case complexity of reasoning is EXPTIME-complete, therefore there can be pathological KBs manifesting this complexity.

3 Optimising the algorithm

The experience with previous DL systems shows that the direct implementation of the tableaux–based satisfiability algorithms provides very poor performance, unacceptable for any real application. Even though the experiments with other DL systems have been mainly at the terminological level, we can try to extract some lessons from those experiences (see Haarslev and Möller [1999], Horrocks and Patel-Schneider [1999]). The precompletion phase is completely separated from the terminological reasoning, so optimisation techniques implemented at the two levels do not

interact adversely. Moreover, the terminological reasoner we are using (FaCT) is already optimised, therefore we focus on the optimisation we can perform during the precompletion.

Section 3.1 introduces some of the well known optimization techniques adopted by most of the state of the art DL reasoners, while Section 3.2 and Section 3.3 describe techniques developed in conjunction with the precompletion algorithm.

3.1 Standard techniques

Axiom absorption and lazy expansion General axioms are one of the major sources of nondeterminism; in fact, in every axiom $C \sqsubseteq D$ is hidden the disjunctive formula $D \sqcup \neg C$ applied to every individual name. One of the most effective ways of reducing the effects of axioms is the so called *absorption* technique used in conjunction with lazy expansion of concept names (see Horrocks and Tobies [2000]).

Roughly speaking, the idea behind this technique is to transform a general axiom into the special form $A \sqsubseteq C$ where A is a concept name. Then the axiom is treated as a sort of definition for the name A and ignored until a concept constraint of the form $o{:}A$ is examined; at this point the "definition" C of A is added to the label of o (i.e. the new constraint $o{:}C$). This basic idea can be extended to negated concept names as well; i.e. having definitions of the form $\neg A \sqsubseteq C$. However, their combination must be used carefully to avoid incorrect results. We implemented the absorption algorithm described in Horrocks and Tobies [2000].

Lexical normalisation Concept expressions are normalised and encoded according to the transformation rules described in Horrocks and Patel-Schneider [1999]. In the normal form, concept expressions can be concept names, conjunctions of normal form concepts, universal quantification constructors, and the negation of a normal form. Conjunctions are represented as sets, so the order of the conjuncts does not affect the syntactic equivalence of different expressions; in addition, nested conjunctions are flattened (e.g. expressions like $((C_1 \sqcap C_2) \sqcap D)$ are transformed into $(C_1 \sqcap C_2 \sqcap D)$).

Backjumping Inherent unsatisfiability concealed in subproblems can lead to large amounts of unproductive backtracking search known as thrashing. Consider for example the set of constraints

$$\left\{ \begin{array}{l} a{:}(C_1 \sqcup D_1), \ldots, a{:}(C_n \sqcup D_n), a{:}\forall S.\forall R.\neg C, \\ b{:}\exists R.(C \sqcap D), \langle a, b\rangle{:}S \end{array} \right\},$$

which may cause the exploration of 2^n alternative combinations of constraints on a deriving from the concepts $(C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n)$, while the true cause for the failure is related to the individual b. For example, this will happen if the rule application strategy forces the evaluation of all the constraints associated with an individual before considering a different individual.

This problem is addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems (see Baker [1995]). Backjumping works by labeling concept constraints with a dependency set indicating the branching points on which they depend. A concept constraint $a{:}C$ depends on a branching point if $a{:}C$ was added to the label by the $\sqcup$–rule generating the branching point or if $a{:}C$ was generated by a different rule and the concept constraints involved in the rule depend on the branching point.[3]

When a clash is discovered, the dependency sets of the clashing concept constraints can be used to identify the most recent branching point where exploring the other branch might alleviate the cause of the clash. The algorithm can then jump back over intervening branching points without exploring alternative branches.

Note that when a contradiction is discovered by the verification of a label, there is no way for our algorithm to know the source of the contradiction; therefore, the union of the dependency set of all constraints of the label is returned. As we are going to show in Section 3.2, we can modify the algorithm in order to make the backjumping more effective in such cases.

With respect to the given example, the precompletion algorithm generates the first precompletion

$$\left\{ \begin{array}{l} a{:}(C_1 \sqcup D_1), \ldots, a{:}(C_n \sqcup D_n), a{:}\forall S.\forall R.\neg C, \\ a{:}C_1, \ldots, a{:}C_n \\ b{:}\exists R.(C \sqcap D), b{:}\forall R.\neg C, \langle a, b\rangle{:}S \end{array} \right\},$$

by choosing the first concept for each constraint containing the disjunction, and applying the $\forall$–rule to the constraints $a{:}\forall S.\forall R.\neg C$ and $\langle a, b\rangle{:}S$. In this process, the algorithm generates n different branching points, and every constraint $a{:}C_1, \ldots, a{:}C_n$ is labelled with a different branching point. The constraints associated with b have no branching point associated with them (this indicate a dependency with the top level), so when the terminological reasoner discovers the inconsistency of the label associated with b there is no reason to explore the alternative options for the individual a.

[3] Since new role assertions are never introduced, the dependency is only related to concept expressions.

3.2 Precompletion techniques

In the algorithm described above, the terminological reasoner is used only when a precompletion is generated. However, it can be used in different phases of the algorithm in a more sophisticated way.

Result caching As pointed out in Donini et al. [1996], caching the satisfiability results for the tested concept expressions is essential for maintaining the complexity of the actual algorithm in the EXPTIME theoretical complexity. In addition, in our case it can allow us to avoid the overhead of converting a concept expression from the internal representation to a format suitable for the terminological reasoner.

Early inconsistency detection This is based on the observation that if the constraints applied to an individual are inconsistent they will be inconsistent in any generated precompletion. Therefore the label of an individual can be verified even before an actual precompletion is generated. If an individual is found to have an inconsistent label, the algorithm stops exploring the current search branch and backtracks to the appropriate saved state of the precompletion process.

The label is verified when all the constraints associated to an individual have been considered, before considering a different individual. If during subsequent precompletion of the rest of the KB no new constraints are added to an individual which as been already precompleted, there is no reason to verify its label again.

Modal verification Backjumping only works properly with a precise knowledge of the constraints which generate a clash. Our problem is that in case of a clash detected by the terminological reasoner, we cannot get the information about the constraints responsible for the clash. The assumption that any concept in the label can be the cause of the clash makes the backjumping technique much less effective.

For example, the label $\{C, \neg D, \exists S.C, \exists R.C, \forall R.D\}$ is inconsistent if the terminology contains an axiom like $C \sqsubseteq \neg D$. However, the inconsistency can only be detected by the terminological reasoner. Once the unsatisfiability of the label is discovered, the system cannot tell which element of the label caused the inconsistency.

We can improve the algorithm by observing that if during the precompletion a clash has not been detected, the only possible cause for an inconsistency must be associated with an "anonymous" element whose existence is enforced by an existential quantification.[4] This is because the tree-like model property of the logic (see Vardi [1997], Tessaris [2001]) guarantees that no constraints can be "pushed back" from these "anonymous" elements, and contradictions generated in the propositional part of the labels (e.g. like C and $\neg C$) are detected during the precompletion process.

When there is the necessity of verifying the satisfiability of a label, the standard algorithm builds a new concept by conjoining all the concepts in the label. With the modal verification technique, the only concepts considered are the universal and existential quantifications (i.e. $\exists R.C$ and $\forall R.C$). Moreover, the algorithm selects the smallest subsets of the label which can be independently verified without compromising the completeness of the algorithm. In the previous example the terminological reasoner is used to verify the two concepts $(\exists R.C \sqcap \forall R.D)$ and $\exists S.C$ independently.

The basic idea for selecting these subsets is to consider each existential concept in a different subset, and adding the universal restrictions which can apply to it (i.e. having the same or a more general role name). However, this simple approach does not work with $\mathcal{SH}f$, because of the interaction between functional role and role hierachy. In fact, the operators $\cdot \approx_o \cdot$ introduced in Section 2 are used to group the relevant concepts of the label.

Using this technique, in case of unsatisfiability we can narrow the set of involved constraints. Therefore it can be extremely useful in conjunction with backjumping, because it enables a more precise identification of the backtrack point responsible for the clash.

Let us consider the following constraints

$$\left\{\begin{array}{l} a{:}(C_1 \sqcup C_2), \ldots, a{:}(C_{2n-1} \sqcup C_{2n}), \\ a{:}\forall R.\neg C, a{:}\exists R.(C \sqcap D) \end{array}\right\}.$$

In this case backjumping alone would not provide any improvement because the generated precompletions are of the form

$$\left\{\begin{array}{l} a{:}(C_1 \sqcup C_2), \ldots, a{:}(C_{2n-1} \sqcup C_{2n}), \\ a{:}C_{i_1}, \ldots, a{:}C_{i_n} \\ a{:}\forall R.\neg C, a{:}\exists R.(C \sqcap D) \end{array}\right\},$$

and the terminological reasoner is invoked to verify the concept

$$C_{i_1} \sqcap \ldots \sqcap C_{i_n} \sqcap \forall R.\neg C \sqcap \exists R.(C \sqcap D).$$

The algorithm is unable to detect which constraints are the cause of the unsatisfiability. By checking the modal part only, the terminological reasoner is invoked with the concept $\forall R.\neg C \sqcap \exists R.(C \sqcap D)$. When the unsatisfiability is reported, the algorithm knows that the clash is generated by one of the two constraints $a{:}\forall R.\neg C, a{:}\exists R.(C \sqcap D)$ and

[4] A constraint like $a{:}\exists R.C$ implies that there is an element related to individual name a through role R. The precompletion process would not investigate its existence since it is a task left to the terminological reasoner.

the backjumping is more effective.

3.3 Instance checking by subsumption

Instance checking is the problem of verifying whether the interpretation of an individual name belongs to a certain concept in every interpretation. This is a fundamental service for any Abox reasoner and it is usually performed by a reduction to KB satisfiability (see Schaerf [1994]). In fact, an individual name a is a member of a concept C in every interpretation satisfying a KB iff adding of the Abox assertion $a{:}\neg C$ makes the KB unsatisfiable.

The terminological reasoner can sometimes be used to avoid precompletion in case of instance checking, by using an approximation of the Most Specific Concept technique (see Era and Donini [1992], Küsters and Molitor [2001]). The idea is to build a concept expression which is guaranteed to contain the individual, and is as specific as possible. The trivial way of doing it is by conjoining all the concepts in the label of the individual, but the role constraints can be considered as well to narrow the expression by using existential quantification.

For example, from the set of constraints $\{a{:}C, \langle a, b\rangle{:}R, b{:}D\}$ we know that the individual a must be contained in the expression $(C \sqcap \exists R.D)$. This process can be carried on for different levels of role assertions (e.g. b can be related to a third individual c or even to a itself); we decided to build the MSC by stopping when a cycle is detected. Note that the result is an approximation of the Most Specific Concept for a given individual name; however, it is good enough for our purpose.

Now, let us assume that we calculated the MSC associated to the element a and we call it MSC_a. If we are interested in verifying whether the individual a is in the extension of the concept C in every interpretation, we can first check if MSC_a is subsumed by C or by $\neg C$ before performing any precompletion. In fact, if MSC_a is subsumed by C then a is an instance of C, while if it is subsumed by $\neg C$ we know for sure that a cannot be an instance of C in any interpretation satisfying the KB. If neither of the tests succeed, we cannot say anything about the membership of the individual to the given concept, and a standard precompletion process is performed.

4 Empirical evaluation

The main purpose of the experiments we performed with the system was to verify the feasability of the approach in terms of performance. In particular, whether optimisations made at the precompletion level make a real difference, and which ones produce the best results. We compared our implementation with RACE (see Haarslev and Möller [2000]) because it is the fastest available Abox rasoner, and it provides a superset of $\mathcal{SH}f$ (it has unqualified number restrictions as well). The comparison of RACE with other DL systems can be found in Franconi et al. [1998]. In this paper we show a summary of the experiments, for a full description refer to Tessaris [2001].

4.1 DL benchmark suite

The main problem in performing experiments with an Abox reasoner is the lack of real data (i.e. KBs). For our experiments we used the *synthetic Abox tests* of the DL benchmark suite,[5] a collection of tests for DL systems which has been introduced in the DL'98 workshop (see Horrocks and Patel-Schneider [1998]). Unfortunately, the test suite is mainly oriented towards terminological reasoning, because the tests were originated in the modal logic community.

There are 9 classes of tests, each one containing different instances of problems of increasing difficulty. Each instance is automatically generated according to a schema related to the class, and it consists of a Tbox, an Abox and a set of instance checking queries. Each instance is evaluated by loading the knowledge base (i.e. Tbox and Abox), then the KB satisfiability is checked and all the Abox queries are evaluated with a fixed CPU time limit of 500 seconds.[6] For each class is recorded the latest instance the system has been able to evaluate within the time limit.

Abox test instances derive from the concepts generated for the *Concept Satisfiability Tests* (see Horrocks and Patel-Schneider [1998]). The Tbox is generated by naming all the sub-concepts appearing in the test concept, while the Abox is generated from the model generated by a satisfiability test over the concept itself. This approach has the advantage of being well defined, and the results for each Abox query is known in advance; this is crucial for verifying the correctness of the DL system. On the other hand, it has several disadvantages that make the tests unsatisfactory as examples of realistic problems. Firstly, the originating concepts contain a single role name, and this is very unlikely in any real KB. The second problem is the terminology. Since all the sub–concepts are recursively named the resulting Tbox is acyclic, and this means that it can be considered as an empty terminology. The last problem of the synthetic Abox tests is the fact that Aboxes are built starting from a model generated by a tableaux based DL system for a concept. The kind of models these systems generate

[5] It is available at the URL `http://kogs-www.informatik.uni-hamburg.de/~moeller/dl-benchmark-suite.html`.

[6] Experience shows that, due to the exponential nature of the problem, instance problems not solved within this time limit are unlikely to be solved in a reasonable amount of time.

are not unstructured, but they are always tree shaped and mostly fully connected. Obviously they cannot be taken as a representative sample for generic KBs.

As we are going to show in the next section, the inadequacy of the data tests is highlighted by some of the results of our experiments. Unfortunately, these synthetic Abox tests are the only coherent suite for testing DL systems, so we must make the most of them. However, the results must be considered in the light of their limitations. We do not think that the available Abox tests reflect realistic KBs, so they cannot give an estimate of the size/type of KB our system can handle.

4.2 Measurements

Our objective is to etablish a correlation between the performance of the system on different tests and the behaviour of the optimization techniques. We assumed that each class of tests presents a different pattern, therefore we took separate measurements for each class. The results are summarised in the following table:

	MSC	Default	Backjumping only	None	RACE
k_branch_n	2	2	2	1	3
k_d4_n	2	2	2	1	2
k_dum_n	16	8	2	1	13
k_grz_n	8	7	7	2	10
k_lin_n	4	4	4	3	4
k_path_n	3	4	3	1	3
k_ph_n	6	6	6	6	5
k_poly_n	4	4	4	4	4
k_t4p_n	5	2	2	0	2

Results of the experiments are grouped by the class of tests, and each column shows the last instance of the test which has been solved within the given timeout of 500 seconds. Different columns contain the results with different configuration for the optimizations described in Section 3. As a reference, the last column shows the results of the RACE system (version 1.2).[7] All the experiments have been performed on a machine equipped with a Celeron 450 MHz processor, with 256 Mb of main memory and running Linux with the kernel version 2.2.17. Both the systems are written in Common Lisp and run using Allegro Common Lisp version 5.0.1.

In the first configuration (MSC) all the previously described optimizations are active; however, almost all the queries are solved by a single subsumption test; therefore precompletion is not used. The second column is the default settings with all optimization active with the exception of the *instance checking by subsumption* technique, because it prevents the precompletion, so other optimizations are not exercised. In the fourth column only backjumping is active, and *modal verification* is turned off. Finally, the fifth column shows the results for the system without any optimization apart from the basic *lexical normalisation* and *axiom absoption*.

5 Conclusions

We noticed that most of the instance checking tests can be solved by terminological reasoning only, without precompleting the KB. In all classes except **k_d4_n** and **k_grz_n** all the instance checks are solved without precompleting the Abox, and even in those two classes over 98% of the instance checks are solved by using the MSC. This is a clear indication that the tests are not adequate for fully evaluating an Abox reasoner.

Note that using the MSC is not always the best way (in terms of efficiency) of performing the instance checking. Although, in some cases it provides remarkably good results (e.g. **k_dum_n** and **k_t4p_n**), in others it actually manifests a loss of performance (e.g. **k_path_n**). In these cases the algorithm is able to discover clashes in early stages of the precompletion and backjumping is very effective, while the overhead of building and testing the MSC does not pay off.

Given the fact that during the precompletion no new individuals are created, nondeterminism is definitely the main source of slow down of the system, and optimisations that are directed at pruning the search space (such as backjumping) seem to provide uniformly better results. This is clearly indicated by the fact that the performances degraded significantly in the last two configurations. Moreover, as predicted in Section 3.1, backjumping provides better results if associated with the *modal verification* technique.

With the Abox tests from the benchmark suite, we really did not expect our system to be faster than the RACE system; even in a few classes. In fact, we know that RACE is no slower than FaCT as a terminological reasoner,[8] and it uses the very same engine for both Abox and terminological reasoning. If you look closely at the precompletion technique, it uses transformation rules which are very similar to the ones implemented in RACE; the difference lies the fact that our system made a sharp distinction between rules working with the Abox (the precompletion rules) and rules working at the terminological level (delegated to the terminological reasoner FaCT).

A system without this separation (like RACE) has the pos-

[7] This was the version of the RACE system available at that time.

[8] In fact, w.r.t. $\mathcal{SH}f$ DL both RACE and FaCT use almost the same algorithm and optimizations, and difference in their performance is mainly due to implementation details.

sibility to maximise the effect of optimisations like backjumping because it has better information about constraints causing contradictions (something we tried to simulate by verifying the modal part of the label only). We think that with classes like **k_branch_n** it is this fact that makes the real difference in performance. In addition, RACE is a more mature system and it has more optimisation techniques implemented (for example model merging, see Haarslev and Möller [2000]).

For these reasons, we did not expect to outperform RACE for any test having a small and almost fully connected Abox, where partitioning and memory occupation do not make any difference. Obviously, it could be the case that some of the results exhibited by our system depend on the peculiar structure of the test problems. However, if this behaviour is exhibited even with general KBs, it means that there is some unnecessary overhead in the hybrid reasoning in RACE which can be reduced by separating the treatment of Abox and "terminological" constraints. This would suggest that the performance of tableaux–based algorithms, such as the one implemented in RACE, might be improved by using an evaluation strategy more similar to the precompletion strategy (e.g. work on individuals first).

References

A. Artale, E. Franconi, N. Guarino, and L. Pazzi. Part-whole relations in object-centered systems: An overview. *Data and Knowledge Engineering (DKE)*, 20: 347–383, 1996.

F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 446–451. Morgan Kaufmann, 1991.

A. B. Baker. *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. PhD thesis, University of Oregon, 1995.

M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In *Proceedings of the Fifth International Conference on the Principles of Knowledge Representation and Reasoning (KR-96)*, pages 316–327. Morgan Kaufmann Publishers, 1996.

G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for Converse-PDL. *Information and Computation*, (162): 117–137, 2000.

F. M. Donini, G. De Giacomo, and F. Massacci. EXPTIME tableaux for $\mathcal{ALC}$. In L. Padgham, E. Franconi, M. Gehrke, D. L. McGuinness, and P. F. Patel-Schneider, editors, *Collected Papers from the International Description Logics Workshop (DL'96)*, number WS-96-05 in AAAI Technical Report, pages 107–110. AAAI Press, Menlo Park, California, 1996.

F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Deduction in concept languages: From subsumption to instance checking. *Journal of Logic and Computation*, 4 (4):423–452, 1994.

A. Era and F. Donini. Most specific concepts for knowledge bases with incomplete information. In *Int. Conf. on Information and Knowledge Management, Baltimore*, 1992.

E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, and C. A. Welty, editors. *Proceeding of the 1998 International Workshop on Description Logics (DL'98)*, 1998. CEUR Pubblication.

V. Haarslev and R. Möller. An empirical evaluation of optimization strategies for abox reasoning in expressive description logics. In P. Lambrix, A. Borgida, M. Lenzerini, R. Möller, and P. F. Patel-Schneider, editors, *Proceeding of the 1999 International Workshop on Description Logics (DL'99)*, pages 115–119. CEUR Pubblication, 1999.

V. Haarslev and R. Möller. Consistency testing: The RACE experience. In *Proceedings of TABLEAUX 2000*, volume 1847 of *Lecture Notes in Computer Science*, pages 57–61. Springer, 2000.

B. Hollunder. *Algorithmic Foundations of Terminological Knowledge Representation Systems*. PhD thesis, Universität des Saarlandes, 1994.

I. Horrocks and G. Gough. Description logics with transitive roles. In *Proceedings of the International Workshop on Description Logics, DL'97*. LRI, Universitè PARIS-SUD, Cente d'Orsay, 1997.

I. Horrocks and P. F. Patel-Schneider. DL system comparison. In Franconi et al. [1998].

I. Horrocks and P. F. Patel-Schneider. Optimising description logic subsumption. *Journal of Logic and Computation*, 9(3):267–293, 1999.

I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. In Franconi et al. [1998].

I. Horrocks and U. Sattler. A description logic with transitive and inverse roles and role hierarchies. *Journal of Logic and Computation*, 9(3):385–410, 1999.

I. Horrocks, U. Sattler, and S. Tobies. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David

MacAllester, editor, *Proc. of the 17th Int. Conf. on Automated Deduction (CADE-17)*, number 1831 in Lecture Notes In Artificial Intelligence, pages 482–496. Springer-Verlag, 2000.

I. Horrocks and S. Tobies. Reasoning with axioms: Theory and practice. In *Proc. of the 7th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'2000)*, pages 285–296, 2000.

R. Küsters and R. Molitor. Approximating Most Specific Concepts in Description Logics with Existential Restrictions. In F. Baader, editor, *Proceedings of the Joint German/Austrian Conference on Artificial Intelligence, 24th German / 9th Austrian Conference on Artificial Intelligence (KI 2001)*, volume 2174 of *Lecture Notes in Artificial Intelligence*, Vienna, Austria, 2001. Springer–Verlag.

A. Rector and I. Horrocks. Experience building a large, reusable medical ontology using a description logic with transitivity and concept inclusions. In *Proceedings of the Workshop on Ontological Engineering, AAAI Spring Symposium (AAAI'97)*. AAAI Press, Menlo Park, California, 1997.

U. Sattler. A concept language for an engeneering application with part-whole relations. In A. Borgida, M. Lenzerini, D. Nardi, and B. Nebel, editors, *Proceedings of the International Workshop on Description Logics*, pages 119–123, Rome, 1995.

A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

M. Schmidt-Schauss and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

S. Tessaris. *Questions and answers: reasoning and querying in Description Logic*. PhD thesis, University of Manchester, 2001.

M. Y. Vardi. Why is modal logic so robustly decidable? Technical Report TR97-274, Rice University, 1997.