To appear in B. Nebel, C. Rich, W. Swartout, editors, *Principles of Knowledge Representation and Reasoning – Proceedings of the 3rd International Conference*, October 1992, Cambridge, MA.

# An Empirical Analysis of Optimization Techniques for Terminological Representation Systems

or: Making KRIS get a move on

**Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich**
German Research Center for AI (DFKI)
Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany
e-mail: ⟨*last name*⟩@dfki.uni-sb.de

**Enrico Franconi**
Istituto per la Ricerca
Scientifica e Tecnologica (IRST)
38050 Povo TN, Italy
e-mail: franconi@irst.it

## Abstract

We consider different methods of optimizing the classification process of terminological representation systems, and evaluate their effect on three different types of test data. Though these techniques can probably be found in many existing systems, until now there has been no coherent description of these techniques and their impact on the performance of a system. One goal of this paper is to make such a description available for future implementors of terminological systems. Building the optimizations that came off best into the KRIS system greatly enhanced its efficiency.

## 1 INTRODUCTION

Terminological representation systems can be used to represent the taxonomic and conceptual knowledge of a problem domain in a structured and well-formed way. To describe this kind of knowledge, one starts with atomic concepts (unary predicates) and roles (binary predicates), and defines more complex concepts and roles using the operations provided by the concept language of the particular formalism. In addition to this concept description formalism, most terminological representation systems also have an assertional component, which can be used to express facts about a concrete world.

Of course, it is not enough to have a system that just stores concept definitions and assertional facts. The system must also be able to reason about this knowledge. An important inference capability of a terminological representation system is *classification*. The classifier computes all *subsumption* relationships between concepts, i.e., the subconcept-superconcept relationships induced by the concept definitions. In this paper we consider only optimizations for the classification process. We do not take into account problems that are specific to assertional reasoning. This concentration on the terminological component is partially justified by the fact that this is the part that partakes in most reasoning activities of almost all systems—which means that the efficiency of this reasoning component is crucial for the overall behavior of the system.

The first terminological representation system, KL-ONE [Brachman and Schmolze,1985], was an implementation of Brachman's work on structured inheritance networks [Brachman,1977]. In the last decade many knowledge representation systems based on these ideas have been built, for example BACK [Peltason,1991], CLASSIC [Patel-Schneider *et al.*,1991], KANDOR [Patel-Schneider,1984], KL-TWO [Vilain,1985], K-Rep [Mays *et al.*,1991], KRYPTON [Brachman *et al.*,1985], KRIS [Baader and Hollunder,1991], LOOM [MacGregor,1991], MESON [Edelmann and Owsnicki,1986], NIKL [Schmolze and Mark,1991], SB-ONE [Kobsa,1991], and YAK [Cattoni and Franconi,1990]. Moreover, formal aspects of terminological representation languages have been thoroughly investigated, with the highest emphasis having been placed on the decidability and complexity of the subsumption problem (see, e.g., [Levesque and Brachman,1987; Nebel,1988; Schmidt-Schauß,1989; Patel-Schneider,1989; Nebel,1990b; Schmidt-Schauß and Smolka,1991; Donini *et al.*,1991a; 1991b]). As a result of these investigations, it is known that subsumption determination is at least NP-hard or even undecidable for reasonably expressive languages. The developers of terminological representation systems usually have reacted to this problem in one of the following two ways. On the one hand, there are systems such as CLASSIC which support only a very limited terminological language, but employ almost complete reasoning methods. On the other hand, systems such as LOOM provide for a very powerful language, but the reasoning is incomplete, which means that not all existing subsumption relationships are detected.

The only system that does not make this compromise, i.e., that provides complete algorithms for a very expressive concept description language, is KRIS. Obviously, this means that KRIS will need exponential time for worst case examples which, on the one hand, are not expressible in the less expressive systems, and which are, on the other hand, treated more efficiently, but less completely, by systems with fast and incomplete algorithms. However, it is not *a priori* clear whether this also implies that KRIS has to be less efficient for "typical" knowledge bases. In particular, it might at least be fast in cases where its full expressive power is not used, or where incomplete algorithms are still complete. The empirical analysis of terminological representation systems described in [Heinsohn *et al.*,1992] seems to preclude this possibility, though. KRIS turned out to be much slower than, for example, CLASSIC, even for knowledge bases that are in the scope of CLASSIC's concept language, and for which CLASSIC's subsumption algorithm is complete.

One aim of the present paper is to demonstrate that this bad performance of KRIS is not mainly due to the use of complete subsumption algorithms, but instead to the fact that the tested version was the first implementation of an experimental system where efficiency considerations only played a minor role. For this purpose we shall consider possible optimizations of the classification process on three different levels. The optimizations on the *highest level* are independent of the fact that what we are comparing are concepts defined by a terminological language. On this level, classification is considered as the abstract order-theoretic problem of computing a complete representation of a partial ordering (in our case the subsumption hierarchy) by making as few as possible explicit comparisons (in our case calls of the subsumption algorithm) between elements of the underlying set (in our case the set of all concepts occurring in the terminology). Optimizations on the *next level* still leave the subsumption algorithm unchanged, but they do employ the fact that we are not comparing abstract objects but instead structured concepts. At this level subsumption relationships that are obvious consequences of this structure can be derived without invoking the subsumption algorithm. On the *third level*, the actual subsumption algorithm is changed so that it can benefit from the information provided by subsumption relationships which have previously been computed.

The effects these optimizations have on the classification process are evaluated on three different sets of test data. As in [Heinsohn *et al.*,1992] we consider both existing knowledge bases used in other projects and randomly generated knowledge bases whose structure resembles those of the real knowledge bases. Since the first level of optimizations can be done in an abstract order-theoretic setting, these optimizations are also evaluated on randomly generated partial orderings (see [Winkler,1985] for a description of the generation process we have used).

It should be noted that we do not claim that all the presented optimizations are novel. Most optimizations can probably be found in many of the existing systems (see e.g. [Lipkis,1982; MacGregor,1988; Peltason *et al.*,1989; Woods,1991; Ellis,1991]). However, until now it was not possible to find a coherent description of all of them, and there were no empirical studies on their exact effects. A second motivation for this work is to make such a description available for future implementors of terminological representation systems.

## 2 COMPUTING THE SUBSUMPTION HIERARCHY

In the first level of optimizations we are concerned with computing the concept hierarchy induced by the subsumption relation. More abstractly, this task can be viewed as computing the representation of a partial ordering. For a given partial ordering[1] $\leq$ on some set $P$, $\prec$ shall denote the *precedence relation* of $\leq$, i.e., $\prec$ is the smallest relation such that its reflexive, transitive closure is identical with $\leq$. Obviously, $x \prec y$ iff $x \leq y$ and there is no $z$ different from $x$ and $y$ such that $x \leq z \leq y$. If $x \leq y$, we say that $x$ is a *successor* of $y$ and $y$ is a *predecessor* of $x$. Similarly, if $x \prec y$, we say that $x$ is an *immediate successor* of $y$ and $y$ is an *immediate predecessor* of $x$.

Given a set $X$ and a partial ordering $\leq$ on $X$, computing the representation of this ordering on $X$ amounts to identifying $\prec$ on $X$. (If $\leq$ is a *total* ordering, this task is usually called sorting.) The basic assumption here is that the partial ordering is given via a comparison procedure, and that the comparison operation is rather expensive. For this reason, the complexity of different methods to compute the precedence relation is measured by counting the number of comparisons. Of course, the number of other operations should not be too high as well.

In our case, $X$ is the set of concepts defined in a terminological knowledge base, and $\leq$ is the subsumption relation between these concepts. The assumption that the subsumption test is the most expensive operation is justified by the known complexity results for the subsumption problem [Donini *et al.*,1991a]. To be more precise, the subsumption relation is only a quasi-ordering, i.e., it need not be antisymmetric. For the following discussion, this is mostly irrelevant, however. There is only one place in the algorithms where this fact has to be taken into account.

The worst case complexity of computing the representation of a partial ordering on a set with $n$ elements is obviously $O(n^2)$ because it takes $n \times (n-1)$ comparisons to verify that a set of $n$ incomparable elements is indeed a flat partial order. Since subsumption hierarchies typically do not have such a "pathological"

[1] A partial ordering is a transitive, reflexive, and antisymmetric relation.

structure, considerably less than $n \times (n-1)$ comparisons will almost always suffice.

Below, we describe and analyze four different methods to identify the representation of a partial ordering, namely, the *brute force* method, the *simple traversal* method, the *enhanced traversal* method, and the *chain inserting* method. Average case analyses of these methods seem to be out of reach since one does not know enough about the structure of "typical" terminological knowledge bases, and since it is not even known how many different partial orders exist for a given number of elements [Aigner,1988]. For this reason, the different methods are compared empirically.

All methods we describe are incremental, i.e., assuming that we have identified the precedence relation $\prec_i$ for $X_i \subseteq X$, the methods compute for some element $c \in X - X_i$ the precedence relation $\prec_{i+1}$ on $X_{i+1} = X_i \cup \{c\}$. The two most important parts of this task are the *top search* and the *bottom search*. The top search identifies the *set of immediate predecessors* in $X_i$ for a given element $c$, i.e., the set $X_i{\downarrow}c := \{x \in X_i \mid c \prec x\}$. Symmetrically, the bottom search identifies the *set of immediate successors* of $c$, denoted by $X_i{\uparrow}c$.

To be more precise, the procedures for top search that we will describe below compute the set $\{x \in X_i \mid c \leq x \text{ and } c \not\leq y \text{ for all } y \prec_i x\}$, which in most cases is the set $X_i{\downarrow}c$. Because the subsumption relation is only a quasi-ordering, there is one exception. The concept $c$ can be equivalent to an element $x$ of $X_i$, i.e., $c \leq x$ and $x \leq c$. In this case, the top search procedures will yield $\{x\}$ instead of $X_i{\downarrow}c$. To take care of this case, we test $x \leq c$ whenever the top search procedure yields a singleton set $\{x\}$. If this test is positive, $c$ is equivalent to $x$, and we know that $X_i{\downarrow}c = X_i{\downarrow}x$, and $X_i{\uparrow}c = X_i{\uparrow}x$, which means that we don't need the bottom search phase. Otherwise, the result of the top search procedure is in fact $X_i{\downarrow}c$.

Given the set $X_i{\downarrow}c$, $X_i{\uparrow}c$, and $\prec_i$, it is possible to compute the precedence relation $\prec_{i+1}$ on $X_{i+1} = X_i \cup \{c\}$ in linear time. In fact, one just has to add $\prec$-links between $c$ and each element of $X_i{\downarrow}c$, and between each element of $X_i{\uparrow}c$ and $c$. In addition, all $\prec$-links between elements of $X_i{\uparrow}c$ and $X_i{\downarrow}c$ have to be erased.

### 2.1 THE BRUTE FORCE METHOD

The top search part of the brute force method can be described as follows:

1. Test $c \leq x$ for all $x \in X_i$.
2. $X_i{\downarrow}c$ is the set of all $x \in X_i$ such that the test succeeded and for all $y \prec_i x$ the test failed.

The bottom search is done in the dual way.

This method obviously uses $2 \times |X_i|$ comparisons for the step of inserting $c$ in $X_i$. Summing over all steps leads to $n \times (n-1)$ comparison operations to compute the representation of a partial ordering for $n$ elements. Further, this is not only the worst-case, but also the best-case complexity of this method.

### 2.2 THE SIMPLE TRAVERSAL METHOD

It is obvious that many of the comparison operations in the brute force method can be avoided. Instead of testing the new element $c$ blindly with all elements in $X_i$, in the top search phase the partial ordering can be traversed top-down and in the bottom search phase bottom-up, stopping when immediate predecessors or successors have been found. This leads us to the specification of the simple traversal method.

The top search starts at the top[2] of the already computed hierarchy. For each concept $x \in X_i$ under consideration it determines whether $x$ has an immediate successor $y$ satisfying $c \leq y$. If there are such successors, they are considered as well. Otherwise, $x$ is added to the result list of the top search.

In order to avoid multiple visits of elements of $X_i$ and multiple comparisons of the same element with $c$, the top search algorithm described in Figure 1 employs one label to indicate whether a concept has been "visited" or not and another label to indicate whether the subsumption test was "positive," "negative," or has not yet been made. The procedure *top-search* gets two concepts as input: the concept $c$, which has to be inserted, and an element $x$ of $X_i$, which is currently under consideration. For this concept $x$ we already know that $c \leq x$, and *top-search* looks at its direct successors with respect to $\prec_i$. Initially, the procedure is called with $x = \top$. For each direct successor $y$ of $x$ we have to check whether it subsumes $c$. This is done in the procedure *simple-top-subs?*. Since our hierarchy need not be a tree, $y$ may already have been checked before, in which case we have memorized the result of the test, and thus need not invoke the expensive subsumption procedure *subs?*. The direct successors for which the test was positive are collected in a list *Pos-Succ*. If this list remains empty, $x$ is added to the result list; otherwise *top-search* is called for each positive successor, but only if this concept has not been visited before along another path.

The bottom search can be done again in the dual way. It is interesting to note that this top search is in principle the same as the one described by Lipkis [Lipkis,1982], who implemented the first classification algorithm for KL-ONE. The bottom search described by Lipkis, however, is more efficient than the one given here.

[2] We assume that our concept hierarchies always contain a top element $\top$ and a bottom element $\perp$.

```
top-search(c,x) =
    mark(x,"visited")
    for all y with y ≺_i x do
        if simple-top-subs?(y,c)
          then Pos-Succ ← Pos-Succ ∪ {y}
        fi
    od
    if Pos-Succ is empty
      then Result ← {x}
      else for all y ∈ Pos-Succ do
              if not marked?(y,"visited")
                then Result ← Result ∪ top-search(c,y)
              fi
           od
    fi
```

```
simple-top-subs?(y,c) =
    if marked?(y,"positive")
      then Result ← true
      elsif marked?(y,"negative")
          then Result ← false
          elsif subs?(y,c)
              then mark(y,"positive")
                   Result ← true
              else mark(y,"negative")
                   Result ← false
          fi
      fi
    fi
```

Figure 1: Top search phase of the simple traversal method

### 2.3 THE ENHANCED TRAVERSAL METHOD

Although the simple traversal method is a big advantage compared with the brute force method (see Figure 5 (a)), it still does not exploit all possible information. First, during the top search phase, we can take advantage of tests that have already been performed. Second, in the bottom search phase, we can use the information gained during the top search as well.

Of course, a dual strategy is also possible, i.e., performing the bottom search before the top search and exploiting the information gathered during the bottom search phase. Analyzing Figure 5, it becomes quickly obvious that this strategy would be less efficient, however. In fact, for the simple traversal method—where the top and bottom phase are done in a symmetric way—the top search phase turns out to be a lot faster. Thus it is better to start with this phase because the information gained thereby can then be used to speed up the slower bottom search phase.

When trying to take advantage of tests that have already been performed during top search one can either concentrate on negative information (i.e., that a subsumption test did not succeed) or on positive information (i.e., that a subsumption test was successful).

To *use negative information* during the top search phase one has to check whether for some predecessor $z$ of $y$ the test $c \leq z$ has failed. In this case, we can conclude that $c \not\leq y$ without performing the expensive subsumption test [MacGregor,1988]. In order to gain maximum advantage, all predecessors of $y$ should have been tested before the test is performed on $y$. This can be achieved by using a modified breadth-first search where the already computed hierarchy is traversed in topological order, as described by Ellis [1991]. Alternatively, one can make a recursive call whenever there is a predecessor that has not yet been tested. This is what the procedure *enhanced-top-subs?* described in Figure 2 does. If we replace the call of *simple-top-subs?* in *top-search* by a call of *enhanced-top-subs?*, we get the top search part of the enhanced traversal method.

```
enhanced-top-subs?(y,c) =
    if marked?(y,"positive")
      then Result ← true
      elsif marked?(y,"negative")
          then Result ← false
          elsif for all z with y ≺_i z
                    enhanced-top-subs?(z,c)
              and   subs?(y,c)
              then mark(y,"positive")
                   Result ← true
              else mark(y,"negative")
                   Result ← false
          fi
      fi
    fi
```

Figure 2: Top search phase of the enhanced traversal method. The procedure *top-search* is adopted from the simple traversal method, but instead of *simple-top-subs?* it calls *enhanced-top-subs?*

The enhanced top search procedure just described makes maximum use of failed tests. Alternatively, it is possible to *use positive information.* Before checking $c \leq y$, one can look for successors $z$ of $y$ that have passed the test $c \leq z$ [MacGregor,1988]. If there exists such a successor, one can conclude that $c \leq y$ without performing an actual subsumption test. Although we are only interested in minimizing the number of comparison operations, it should be noted that instead of searching for a successor that has passed the test it is more efficient to propagate positive information up through the subsumption hierarchy. This can be achieved by an easy modification of the procedure *simple-top-subs?*. When the call "subs?($y$,$c$)" yields *true*, not only $y$ is marked "positive," but so are all of $y$'s predecessors. Obviously, this technique cannot be combined with the enhanced top search described in Figure 2 since it reduces the number of subsumption tests only if there are predecessors which have not yet been tested, and enhanced top search tests all predecessors before making a subsumption test.

Neither of these alternatives is uniformly better than the other one, which can be seen by considering the examples described in Figure 3 and 4.

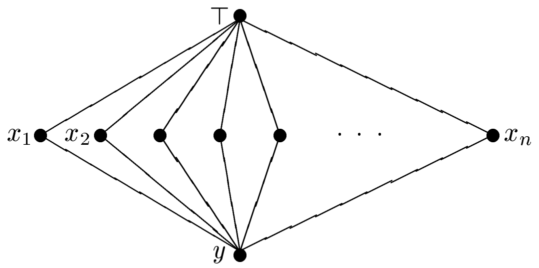



Figure 3: The new element $c$ is a direct successor of $y$

In the first example, the top-search using negative information makes $n+1$ tests: it first tests $x_1$, then goes to $y$, but before testing it, it tests all its direct predecessors, i.e., $x_2, \ldots, x_n$. The top search using positive information makes two tests: first $x_1$ and then $y$; the positive result of this second test is propagated to $x_2, \ldots, x_n$.

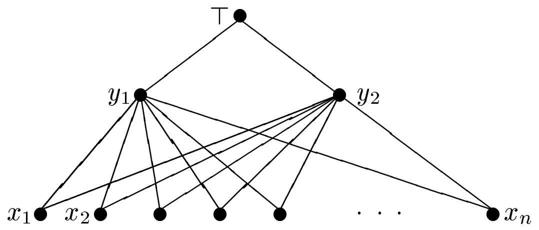



Figure 4: The new element $c$ is a direct successor of $y_1$, but not a successor of $y_2$, $x_1, \ldots, x_n$

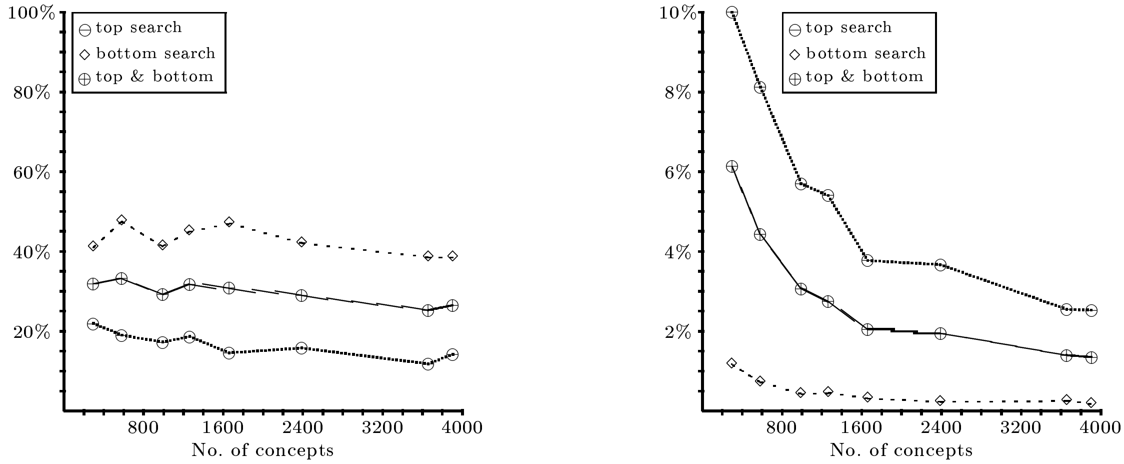
In the second example, the top search using negative information needs only two tests: first it tests $y_1$, then goes to $x_1$, but before testing $x_1$ its direct predecessor $y_2$ is tested. The negative result of this test prevents $x_1, \ldots, x_n$ from being tested. The top search using positive information tests $n+2$ nodes: first $y_1$, then all its successors $x_1, \ldots, x_n$, and finally $y_2$.

However, we have observed significant performance differences for the two different top search strategies. For the random knowledge bases, the method using positive information was only slightly better than the simple traversal method (less than 5%). For this reason, we have also considered a "hybrid method" which propagates positive information up, and negative information down the hierarchy (but does not test all predecessors before testing a node). Propagating negative information down is again achieved by an easy modification of *simple-top-subs?*. When the call of "subs?($y$,$c$)" yields *false*, not only $y$ is marked "negative," but all of $y$'s successors. The hybrid method turned out to be a lot better than just propagating positive information, but it still needed slightly more tests (approx. 5%–10%) than the enhanced top search for all but one of the random knowledge bases. On five of the six realistic knowledge bases the "hybrid method" was insignificantly faster than the enhanced top search (less than 1%). On the remaining realistic KB, the hybrid method needed 10% more comparisons. Although these results do not seem to be conclusive in favor of the "hybrid method" or the enhanced top search, it is obvious that the use of negative information leads to a significantly greater reduction of comparisons than the use of positive information.

Now we turn to the *bottom search* phase of the enhanced traversal method. Of course, optimizations dual to the ones described for the top search can be employed here. In addition, the set $X_i{\downarrow}c$ can be used to severely cut down the number of comparisons in the bottom search phase. As mentioned by Lipkis [1982], the search for immediate successors of $c$ can be restricted to the set of successors of $X_i{\downarrow}c$. In fact, the set of candidates for $X_i{\uparrow}c$ is even more constrained. Only elements that are successors of *all* $x \in X_i{\downarrow}c$ can be immediate successors of $c$ [Ellis,1991]. This optimization is achieved by an easy modification of the procedure *enhanced-bottom-search* (which is dual to *enhanced-top-search*): the test "marked?($y$,"negative")" is augmented to "marked?($y$,"negative") or $y$ is not a successor of all $x \in X_i{\downarrow}c$." The remaining problem is how to implement the second part of this test. One possibility is to mark the successors of the elements of $X_i{\downarrow}c$ in an appropriate way, and then test these labels. Another possibility, which we have used in our tests, is to equip each concept in $X_i$ with a list of all its predecessors in $X_i$, and test whether $X_i{\downarrow}c$ is contained in the list of predecessors of $y$.

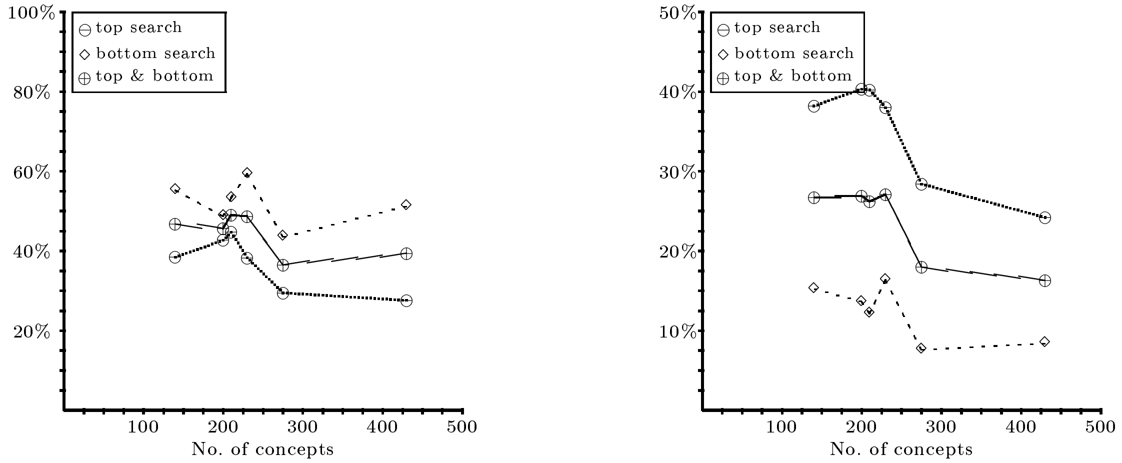
As a result of this optimization, the number of necessary comparison operations can be cut down to a fraction compared with the simple bottom search strategy. Interestingly, we observed a further reduction of comparison operations in case of the real knowledge bases when searching top-down starting at $X_i{\downarrow}c$ instead of searching bottom-up. For the random knowledge bases, no such difference was observed, however.

The effects of the simple and enhanced traversal




(a) Simple traversal method (b) Enhanced traversal method

Figure 5: Number of comparison operations relative to *brute force* method for random knowledge bases




(a) Simple traversal method (b) Enhanced traversal method

Figure 6: Number of comparison operations relative to *brute force* method for realistic knowledge bases

method for the random knowledge bases and the realisitc knowledge base as test data are displayed in Figures 5 and 6. These graphs present the number of necessary comparisons *relative* to the brute force method for the top search and the bottom search phase, as well as for the entire classification process.

### 2.4 THE CHAIN INSERTING METHOD

Sorting a set of elements that is linearly ordered can be either done by incrementally searching the already ordered sequence linearly or by using binary search. In the former case, we inevitably end up with quadratic complexity, while in the latter case $O(n \times \log n)$ is a possibility. Of course, it seems attractive to transfer the latter technique to our problem, an idea that leads to the chain inserting method.

In order to specify the method, we first define the notion of a chain covering of a partial ordering. A *chain covering* is a partition of a partial ordering into *chains*, i.e., totally ordered subsets. Provided we have a chain covering of the set $X_i$, it is possible to identify the sets $X_i{\downarrow}c$ and $X_i{\uparrow}c$ by binary search in all chains. For a given chain $C_j$ of the covering $X_i = C_1 \cup \cdots \cup C_m$, binary search is used to find the least predecessor and the greatest successor of $c$ in $C_j$. Since the underlying ordering $\leq$ is only a partial ordering on $X$, the new element $c$ to be inserted into the chain $C_j$ need not be comparable with all elements of $C_j$. For this reason one needs two binary search phases for each chain. The first one asks $c \leq x$, and treats negative answers as if they would mean $c > x$. This phase yields the least predecessor of $c$ in $C_j$. The other phase is dual, and yields the greatest successor of $c$ in $C_j$. The set of these least predecessor (resp. greatest successors) for all chains of the covering yields a superset of $X_i{\downarrow}c$ (resp. $X_i{\uparrow}c$). The set $X_i{\downarrow}c$ (resp. $X_i{\uparrow}c$) is obtained by eliminating the elements which are not minimal (resp. maximal) with respect to $\leq_i$. As a further optimization, propagation of positive and negative information of successful and of failed tests in the existing subsumption hierarchy is used to make some of the explicit subsumption tests during binary search superfluous, after one or more chains have already been searched through.

We have also considered a "hybrid" method that employs chain inserting for long chains and enhanced traversal afterwards. The idea here is that by binary search in long chains one gets rather quickly into the "center" of the partial ordering, from which propagation of positive and negative information should have the greatest effect.

It is, of course, advisable to use chain coverings with a minimal number of chains. Unfortunately, the computation of minimal chain coverings is nontrivial and takes more than quadratic time [Jungnickel,1990]. Nevertheless, *simple heuristics* permit the incremental construction of chain coverings that are almost optimal. The heuristic we have used to update the chain covering when a new element $c$ is inserted proceeds as follows. After the sets $X_i{\downarrow}c$ and $X_i{\uparrow}c$ have been computed, $c$ is inserted in the longest chain satisfying one of the following conditions:

1. Binary search has yielded both a least predecessor and greatest successor in the chain, and they are successive elements of the chain. In this case, $c$ is inserted between these two elements in the chain.
2. Binary search has yielded a least predecessor (or greatest successor) in the chain, and it is the least (resp. greatest) element of the chain. In this case, $c$ is inserted below (resp. above) this element in the chain.

If there is no chain satisfying one of these conditions, a new chain consisting of $c$ is created. In our experiments, the chain coverings obtained this way were less than 10% suboptimal.

The empirical results concerning the performance of the chain inserting method are not conclusive. To our surprise, the chain inserting method turned out to be less efficient than the enhanced traversal method on the random and real knowledge bases, even though it is still a lot better than the simple traversal method. The "hybrid" version using chain inserting for long chains and enhanced traversal afterwards also turned out to be less efficient than the pure enhanced traversal method. On the other hand, for tests on randomly generated partial orders the chain inserting method in some cases showed a much better performance than the enhanced traversal method. A reason for this behavior could be that, compared to the realistic knowledge bases we used in our tests, the randomly generated partial orders have a much higher connectivity (which means that propagation of positive and negative information has more effect) and permit longer chains (which makes binary search more important). The chain inserting method may thus become more interesting for knowledge bases defining a relatively deep hierarchy. Additionally, it seems possible to exploit the chain covering in order to implement storage compression techniques as described by Jagadish [1989].

## 3 OBVIOUS SUBSUMPTION RELATIONSHIPS

In this section we describe some further techniques for avoiding subsumption tests by exploiting relations which are obvious when looking at the syntactic structure of concept definitions.[3] These pre-tests require only little effort but can speed up the classification process significantly. We consider three different optimizations which apply to different stages of the classification process.

The first technique can be used prior to the top search. It applies when the description of the concept $c$ that we want to insert is conjunctive (which is the case for the majority of concepts, in particular if we consider the existing real knowledge bases). If this description mentions $x$ explicitly as a conjunct, then it is obviously the case that $c \leq x$. We call such concepts *$x$ told subsumers* of $c$. Of course, if $x$ is also a conjunctively defined concept, it may have told subsumers as well, and these (and their told subsumers, etc.) can be included into the list of told subsumers of $c$. It is rather easy to compile this list while reading in the concept definitions. The information that $c$ is subsumed by its told subsumers can be propagated through the existing hierarchy $(X_i, \prec_i)$ prior to the top search, e.g., by pre-setting the markers used in the traversal method to "positive" for the told subsumers and all their predecessors. A prerequisite for this optimization technique to be effective is that the told subsumers of $c$ are already contained in $X_i$. This can be achieved by inserting concepts following the so-called "definition-order." This order can be formally defined as follows: We say that a concept $x$ directly uses a concept $y$ iff $y$ occurs in the definition of $x$. Let "uses" be the transitive closure of "directly uses." Then $x$ comes in the definition-order after $y$ if $x$ uses $y$.

Assuming that concepts are inserted in the subsumption hierarchy following the definition-order, another optimization can be applied. The bottom search phase can be completely avoided if a *primitive concept* (i.e., a concept that is described by giving only necessary

[3] These techniques are probably used in all systems, see, e.g. [Peltason *et al.*,1989].

conditions) has to be classified. In fact, such a concept $c$ can only subsume the bottom concept and concepts whose definitions use $c$. Since the second type of possible subsumees is not yet present in the actual hierarchy when inserting along the definition-order, the result of the bottom search is just the bottom concept $\perp$. Considering the fact that in realistic KBs the majority of concepts (60%-90%) are primitive, this optimization can save most of the subsumption calls during the bottom search phase. Combining the two optimization techniques led to a saving of 10% to 20% with respect to the pure enhanced traversal method for the realistic knowledge bases. In case of the random knowledge bases, the savings where even greater, as can be seen from Figure 7.

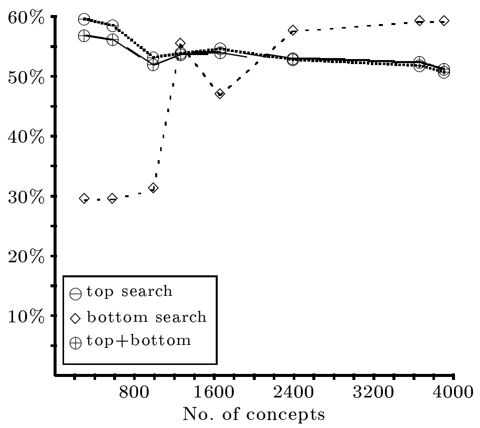



Figure 7: Number of necessary comparisons when exploiting obvious subsumption relations relative to pure *enhanced traversal* method for random KBs

A final optimization technique can be used as a pretest before calling the subsumption algorithm. It makes use of the fact that a concept $c$ is subsumed by a *primitive concept* $x$ if, and only if, the completely expanded form of $c$ contains the "primitive component of $x$" (see [Nebel,1990a, p. 54–56]) as a conjunct. By extracting and caching the "primitive components" of all concepts, it becomes possible to check whether a subsumption relation is possible by comparing the sets of primitive components. If this test gives a negative result, the subsumption algorithm need not be called. Although such a test overlaps with computations the subsumption algorithm does, it is much faster than the subsumption test. For this reason, this pretest pays off if most of the subsumption calls can be avoided, which was indeed the case for our test data. Our experiments indicate that the number of calls of the subsumption algorithm can be again reduced by 50%-60%, if this technique is applied.

# 4 THE SUBSUMPTION ALGORITHM

In this section we consider two possible optimizations of the subsumption algorithm, and describe the effects they have on the performance of classification for our test knowledge bases. Let us first reconsider the two types of subsumption algorithms usually implemented in terminological systems.

In almost all terminological representation systems other than KRIS *structural subsumption algorithms* are employed (e.g. CLASSIC, LOOM, BACK). Such algorithms basically proceed as follows. First, the concepts are normalized, i.e., they are transformed into equivalent normal forms. Subsumption between normalized concepts is a kind of structural comparison where each subexpression of the first concept must have a counterpart in the other concept. This algorithmic technique allows one to develop efficient subsumption algorithms which are easily shown to be sound. However, for expressive terminological languages these algorithms are usually not complete, and it is not clear how the technique could be extended in order to build complete structural subsumption algorithms.

Using a different paradigm, in the past years sound and *complete* subsumption algorithms for a large class of terminological languages have been developed (e.g. [Schmidt-Schauß and Smolka,1991; Hollunder *et al.*,1990]). Most of these algorithms are designed as *satisfiability checking algorithms*. These algorithms are model generation procedures, and are similar to first-order tableaux calculus, with the main difference that the specific structure of concept descriptions allows one to impose an appropriate control that ensures termination. Since a concept $A$ subsumes a concept $B$ if, and only if, $\neg A \sqcap B$ is not satisfiable, i.e., there does not exist an interpretation which interprets $\neg A \sqcap B$ as a non-empty set, a satisfiability algorithm in fact can be used to solve the subsumption problem. In order to check whether a given concept $C$ is satisfiable, the tableaux-based algorithm tries to generate a finite interpretation in which $C$ is interpreted as a non-empty set. This generation process is complete in the sense that if it fails, i.e., an obvious contradiction occurs, we can conclude that $C$ is not satisfiable; otherwise $C$ is satisfiable. An obvious contradiction in the model generating process occurs, for example, if some element is constrained to be both instance of a "primitive component" and its complement—which is impossible.

It is well-known that subsumption of concepts defined in a cycle-free terminology can be reduced easily to subsumption of concept terms which do not refer to other concept definitions of the terminology (so-called *expanded concept terms*) [Nebel,1990a]. For conceptual simplicity both types of subsumption algorithms are usually described in the literature as taking expanded concept terms as arguments, which precludes the exploitation of previously computed subsumption relationships.

## 4.1 THE OPTIMIZATIONS

However, almost all terminological representation systems take advantage of previously computed subsumption relationships. To illustrate how this can be done for a structural subsumption algorithm, suppose that $C$ and $D$ are normalized concept descriptions. As mentioned above, structural subsumption between $C$ and $D$ means to find for each subexpression $C'$ of $C$ a corresponding subexpression $D'$ of $D$. Often, in turn, these subexpressions have to be tested for subsumption. In case $C'$ and $D'$ are concept names of possibly defined concepts, and we already know a subsumption relationship between $C'$ and $D'$, it is not necessary to call the subsumption algorithm recursively for (the expanded form of) $C'$ and $D'$. Thus, it is rather natural and straightforward to incorporate the use of already computed subsumption relations into a structural subsumption algorithm. It should be noted that it is an essential requirement *not* to completely expand the concept definitions before checking subsumption. Further, it is necessary to classify the concepts according to the "definition-order" mentioned in the previous section.

In contrast to other terminological systems, KRIS employs a satisfiability algorithm to determine subsumption relationships between concepts. Since a satisfiability algorithm does not recursively call subsumption algorithms but satisfiability algorithms, it is not obvious how to exploit previously computed subsumption relationships. A closer look, however, reveals that a satisfiability algorithm may detect a contradiction earlier during model generation if previously computed subsumption relationships are taken into account. To see this, suppose that we already know that a defined concept $A$ subsumes a defined concept $B$. If during the model generation an element is constrained to be both instance of $\neg A$ and $B$, a contradiction can be detected without expanding the definitions of $A$ and $B$. Again, this approach only works if the concept definitions are not expanded before starting to check satisfiability.

If expansion is done "by need" during the satisfiability test, one has to decide in which order to expand the concept names. It is easy to see that this order may have considerable impact on the runtime behavior. For example, assume that we are testing $A \sqcap B$ for satisfiability where in the TBox $A$ is defined by a very large concept description and $B$ is defined to be $\neg A \sqcap C$. If $B$ is expanded first, the contradiction between $A$ and $\neg A$ is detected at once. On the other hand, if $A$ is expanded first, detecting the contradiction between the large descriptions associated with $A$ and its negation may be rather time-consuming, depending on the structure of the description.

One way of avoiding this problem is to expand concept names according to the inverse of their definition-order, which in the above example would mean that we expand $B$ before $A$, because the definition of $B$ refers to $A$. Of course, this means that for each expansion operation one has to go through the list of all expandable names, and look for a maximal one with respect to the definition-order. For our tests we have used another solution, which avoids searching for a maximal name, but may use more space. Here one expands in arbitrary order, but when a name is expanded it is not removed, but just marked as expanded. If, in our example, $A$ is expanded before $B$, we then still have the name $A$, and as soon as $B$ is expanded it yields the contradiction with $\neg A$.

In order to gain experience in how to optimize the satisfiability algorithm to be employed in KRIS, we implemented the following three versions.

1. The first one takes *completely expanded* concept descriptions as input. Since these descriptions do not contain names of defined concepts, obvious contradictions can only be detected between "primitive components," i.e., concept names which are not defined in the TBox.
2. The second one *successively expands* the concept descriptions during model generation, but keeps the names, as described above. This allows the algorithm to detect obvious contradictions not only between primitive components but also between names of defined concepts.
3. The third version is a refinement of the second one in that already computed subsumption relationships are taken into account when looking for obvious contradictions.

## 4.2 EMPIRICAL RESULTS AND ANALYSIS

It turns out that the first version is significantly slower than the second one, a result we did expect. The main reason for this behavior is that the number of recursive calls of the satisfiability algorithm is reduced due to obvious contradictions detected between names of defined concepts. As a consequence, the runtime of the second version is reduced by 40-60% relative to the first version (see Figure 8, which displays the results for the random knowledge bases).

A result we did *not* expect is that the behavior of the third version is no better than of the second, which means that trying to exploit already computed subsumption relationships does not pay off. The reason for this behavior seems to be that—at least for the test data—only a few contradictions are detected by using already computed subsumption relationships. This is indicated by the fact that the number of recursive calls of the satisfiability algorithm does not significantly de-

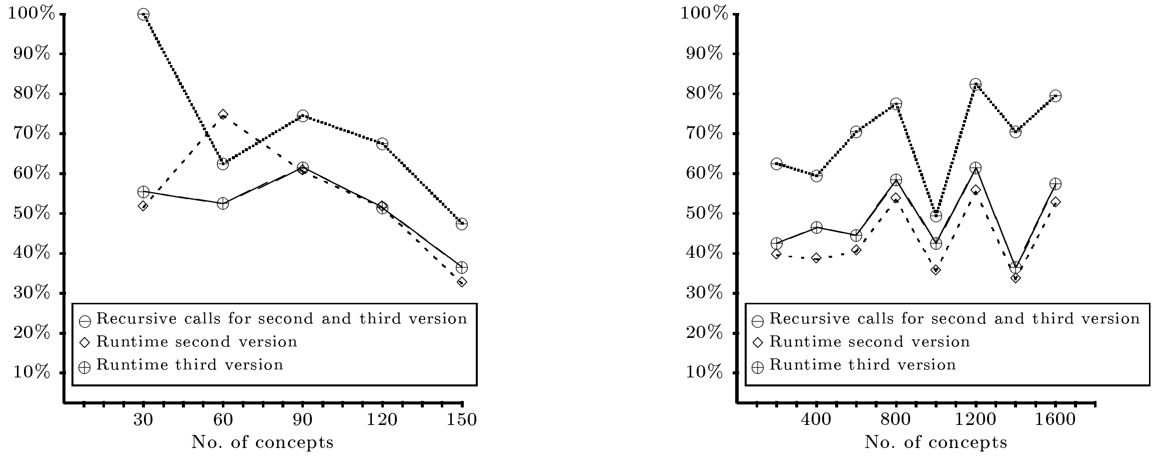



Figure 8: Runtime and number of recursive calls of the second and third version's satisfiability algorithm relative to the algorithm taking completely expanded concept terms as input (first version)

crease when going from the second to the third version. However, the test of whether a set of negated and unnegated concept names is contradictory w.r.t. already computed subsumption relationships is more complex than just searching for complementary names, which explains that the third version's runtime behavior is even slightly worse than the second one's (cf. Figure 8).

This result is all the more surprising since using computed subsumption relationships during classification is an optimization technique employed by most terminological systems. The reason why it may pay off for other systems could be that these systems first normalize, and during this normalization phase auxiliary concepts may be introduced. For example, assume that $C$ is defined by the description $\forall R.A \sqcap \forall R.B$, and $D$ by $\forall R.A$. The normalization procedure may introduce a new concept name $E$, define it as $A \sqcap B$, and modify the definition of $C$ to $\forall R.E$. Now the subsumption relationship between $A$ and the auxiliary concept $E$—which is found first if the terminology is classified according to the definition-order—immediately entails that $D$ subsumes $C$. Thus classification of the terminology with the auxiliary concepts allows one to exploit previously computed subsumption relationships more often. On the other hand, it has the disadvantage that in general a lot more concepts have to be classified.

Another interesting behavior we observed is due to the interaction between different optimization techniques. The optimizations described in the previous two sections try to avoid subsumption tests, whereas the present section is concerned with speeding up the subsumption test. Ideally, one could expect that these optimizations are independent. This means that the overall speedup factor is the product of the speedup factors of the individual optimizations. This can only be true if the optimizations apply uniformly to all situations, however.

If the optimizations apply to special cases only, subsumption avoidance optimizations and subsumption test optimization may aim at similar special cases and lead to the situation that subsumption tests are avoided which have neglectable computational costs in any case.

If we take the second or third version's satisfiability algorithm, the exploitation of obvious subsumption relationships caused by *conjunctive definitions,* i.e., the first optimization technique mentioned in Section 3, does no longer speed up the classification process significantly. This is due to the fact that such subsumption relationships can now be easily detected by the satisfiability algorithms. For example, let $C$ be a concept that is defined to be the conjunction of $C_1, \ldots, C_m$, where the $C_i$ are defined concepts as well. The obvious subsumption relationship between $C_i$ and $C$ is immediately detected by the second and third version of the satisfiability algorithm, due to an obvious contradiction between $C_i$ and $\neg C_i$.

## 5 CONCLUSION

We have described and analyzed different optimization techniques for the classification process in terminological representation systems. Interestingly, two of the most promising techniques, namely, the chain inserting method for computing the representation of a partial order and the exploitation of already computed subsumption relations in the subsumption algorithm, did not lead to the expected performance increase in case of realistic knowledge bases.

As a result of our empirical analysis, the optimization techniques that came off best were incorporated in the KRIS system. Whereas the unoptimized version was orders of magnitude slower than the fastest system tested in [Heinsohn *et al.*,1992], the new version has now a runtime behavior similar to that of the other systems on the test data used there.

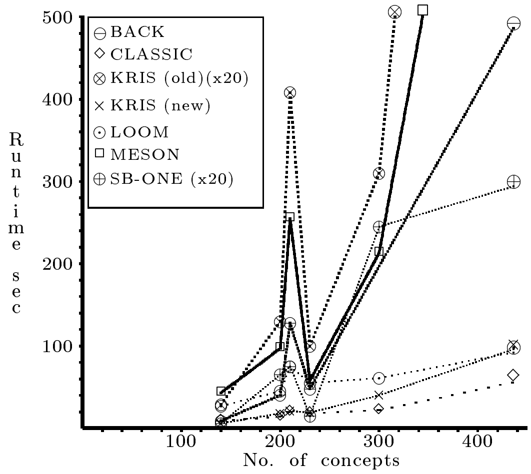



Figure 9: Runtime performance for existing knowledge bases

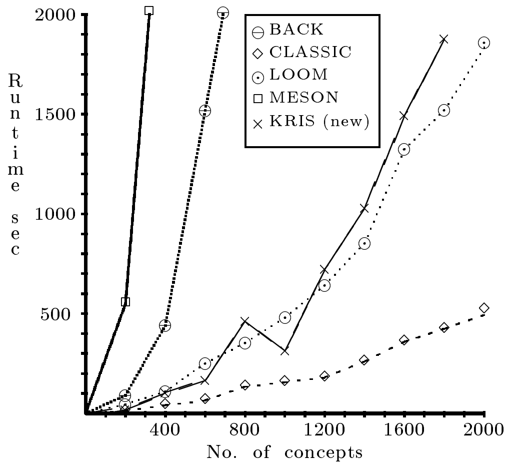



Figure 10: Runtime performance for large random knowledge bases

Figure 9 displays the runtime of the new KRIS version for the realistic knowledge bases and contrasts them with the runtime figures given in [Heinsohn *et al.*,1992]. Figure 10 gives the results for large random knowledge bases.[4]

It should be noted, however, that all the knowledge bases used in the test are formulated using quite limited terminological languages. An interesting open problem is the development of further optimization techniques for more powerful terminological languages containing also disjunction and negation and of specific optimization techniques for assertional reasoning.

[4]The description of the runtime behavior of the systems in [Heinsohn *et al.*,1992] refers to system versions as of 1990 and does not necessarily reflect the performance of more recent versions.


### Acknowledgements

We would like to thank Uwe Utsch for implementing the different subsumption strategies, Hans-Jürgen Bürckert, Jochen Heinsohn, Armin Laux, and Werner Nutt for helpful discussions concerning the topics described in this paper, and Alex Borgida and Peter Patel-Schneider for helpful comments on an earlier version of this paper.

This work has been supported by the German Ministry for Research and Technology (BMFT) under research contracts ITW 8901 8 and ITW 8903 0 and by the Italian National Research Council (CNR), project “Sistemi Informatici e Calcolo Parallelo.”


### References


Martin Aigner (1988). *Combinatorical Search.* Teubner, Stuttgart, Germany.

Franz Baader and Bernhard Hollunder (1991). KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14.

Ronald J. Brachman and James G. Schmolze (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216.

Ronald J. Brachman, Victoria Pigman Gilbert, and Hector J. Levesque (1985). An essential hybrid reasoning system: Knowledge and symbol level accounts in KRYPTON. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 532–539, Los Angeles, CA.

Ronald J. Brachman (1977). *A Structural Paradigm for Representing Knowledge.* PhD thesis, Havard University.

Roldano Cattoni and Enrico Franconi (1990). Walking through the semantics of frame-based description languages: A case study. In *Proceedings of the Fifth International Symposium on Methodologies for Intelligent systems*, Knoxville, TN. North-Holland.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt (1991a). The complexity of concept languages. In James A. Allen, Richard Fikes, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pages 151–162, Cambridge, MA. Morgan Kaufmann.

Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Werner Nutt (1991b). Tractable concept languages. In John Mylopoulos and Ray Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 458–465, Sydney, Australia. Morgan Kaufmann.

Jürgen Edelmann and Bernd Owsnicki (1986). Data models in knowledge representation systems: A case study. In Claus-Rainer Rollinger and Werner Horn, editors, *GWAI-86 und 2. Österreichische Artificial-Intelligence-Tagung*, pages 69–74, Ottenstein, Austria. Springer-Verlag.

Gerard Ellis (1991). Compiled hierarchical retrieval. In *6th Annual Conceptual Graphs Workshop*.

Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich (1992). An empirical analysis of terminological representation systems. In *Proceedings of the 10th National Conference of the American Association for Artificial Intelligence*, pages 767–773, San Jose, CA. MIT Press. An extended version of this paper is available as DFKI Research Report RR-92-16.

Bernhard Hollunder, Werner Nutt, and Manfred Schmidt-Schauß (1990). Subsumption algorithms for concept description languages. In L. C. Aiello, editor, *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 348–353, Stockholm, Sweden. Pitman.

H. V. Jagadish (1989). A compressed transitive closure technique for efficient fixed-point query processing. In L. Kerschberg, editor, *Expert Database Systems—Proceedings From the 2nd International Conference*, pages 423–446, Menlo Park, CA. Benjamin/Cummings.

Dieter Jungnickel (1990). *Graphen, Netzwerke und Algorithmen*. BI Wissenschaftsverlag, Mannheim, Germany, 2nd edition.

Alfred Kobsa (1991). First experiences with the SB-ONE knowledge representation workbench in natural-language applications. *SIGART Bulletin*, 2(3):70–76.

Hector J. Levesque and Ronald J. Brachman (1987). Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93.

Thomas Lipkis (1982). A KL-ONE classifier. In J. G. Schmolze and R. J. Brachman, editors, *Proceedings of the 1981 KL-ONE Workshop*, pages 128–145, Cambridge, MA. The proceedings have been published as BBN Report No. 4842 and Fairchild Technical Report No. 618.

Robert MacGregor (1988). A deductive pattern matcher. In *Proceedings of the 7th National Conference of the American Association for Artificial Intelligence*, pages 403–408, Saint Paul, MI.

Robert MacGregor (1991). Inside the LOOM description classifier. *SIGART Bulletin*, 2(3):88–92.

Eric Mays, Robert Dionne, and Robert Weida (1991). K-Rep system overview. *SIGART Bulletin*, 2(3):93–97.

Bernhard Nebel (1988). Computational complexity of terminological reasoning in BACK. *Artificial Intelligence*, 34(3):371–383.

Bernhard Nebel (1990a). *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, New York.

Bernhard Nebel (1990b). Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43:235–249.

Peter F. Patel-Schneider, Deborah L. McGuinness, Ronald J. Brachman, Lori Alperin Resnick, and Alex Borgida (1991). The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bulletin*, 2(3):108–113.

Peter F. Patel-Schneider (1984). Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, pages 11–16, Denver, Colo..

Peter F. Patel-Schneider (1989). Undecidability of subsumption in NIKL. *Artificial Intelligence*, 39(2):263–272.

Christof Peltason, Albrecht Schmiedel, Carsten Kindermann, and Joachim Quantz (1989). The BACK system revisited. KIT Report 75, Department of Computer Science, Technische Universität Berlin, Berlin, Germany.

Christof Peltason (1991). The BACK system – an overview. *SIGART Bulletin*, 2(3):114–119.

Manfred Schmidt-Schauß and Gert Smolka (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26.

Manfred Schmidt-Schauß (1989). Subsumption in KL-ONE is undecidable. In Ron J. Brachman, Hector J. Levesque, and Ray Reiter, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference*, pages 421–431, Toronto, ON. Morgan Kaufmann.

James G. Schmolze and William S. Mark (1991). The NIKL experience. *Computational Intelligence*, 6:48–69.

Marc B. Vilain (1985). The restricted language architecture of a hybrid representation system. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 547–551, Los Angeles, CA.

P. Winkler (1985). Random orders. *Order*, 1:317–331.

William A. Woods (1991). Understanding subsumption and taxonomy: A framework for progress. In John F. Sowa, editor, *Principles of Semantic Networks*, pages 45–94. Morgan Kaufmann, San Mateo, CA.