# Recognition Algorithms for the Loom Classifier*

**Robert M. MacGregor** and **David Brill**
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
macgregor@isi.edu, brill@isi.edu

## Abstract

Most of today's terminological representation systems implement hybrid reasoning architectures wherein a concept classifier is employed to reason about concept definitions, and a separate *recognizer* is invoked to compute instantiation relations between concepts and instances. Whereas most of the existing recognizer algorithms designed to maximally exploit the reasoning supplied by the concept classifier, LOOM has experimented with recognition strategies that place less emphasis on the classifier, and rely more on the abilities of LOOM's backward chaining query facility. This paper presents the results of experiments that test the performance of the LOOM algorithms. These results suggest that, at least for some applications, the LOOM approach to recognition is likely to outperform the classical approach. They also indicate that for some applications, much better performance can be achieved by eliminating the recognizer entirely, in favor of a purely backward chaining architecture. We conclude that no single recognition algorithm or strategy is best for all applications, and that an architecture that offers a choice of inference modes is likely to be more useful than one that offers only a single style of reasoning.

## Introduction

LOOM (MacGregor and Burstein, 1991; MacGregor, 1991b) belongs to the family of terminological representation systems descended from the language KL-ONE (Brachman and Schmolze, 1985). A characteristic feature of these systems is their ability to reason with definitional and descriptive knowledge. A specialized reasoner called a *classifier* enables these systems to compute subsumption relationships between definitions (to determine when one definition implies another) and to test the internal consistency of definitions and constraints. The kind of reasoning obtainable with a classifier has proved to be useful in a wide variety of applications (Patel-Schneider *et al.*, 1990).

A *recognizer* is a reasoner that complements the abilities of a classifier. Recognition (also called "realization" (Nebel, 1990)) refers to a process that computes instantiation relationships between instances and concepts in a knowledge base. An instantiation relationship holds between an instance and a concept if the instance satisfies (belongs to) that concept. We have observed that LOOM applications use the recognizer much more than they do the classifier (i.e., they spend more of their time reasoning about instances than about definitions) and we believe that this trait extends as well to applications based on other terminological representation systems. Thus, it is of critical importance that the recognition process be as efficient as possible.

Most classifier-based representation systems treat recognition as a variation of concept classification, and implement recognizers that rely on the classifier to perform most of the necessary inferences. LOOM is perhaps the only terminological representation system to experiment with recognition algorithms that differ significantly from what has become the standard or classical approach to building a recognizer. In this paper, we present the results of experiments that suggest that, for at least some applications, the LOOM algorithms perform better than the classical algorithm. LOOM also supports an alternative mode for computing instantiation relationships that substitutes a backward chaining prover in place of the recognizer. This backward chaining mode has proven to be much more efficient for some applications. Thus, the adoption of more flexible (and more complex) recognition algorithms has lead to improved performance for some LOOM applications.

## The Classical Approach to Recognition

Terminological representation systems partition their knowledge bases into a terminological component (TBox) and an assertional component (ABox) (MacGregor, 1990; Nebel and von Luck, 1988). The TBox

*This research was sponsored by the Defense Advanced Research Projects Agency under contract MDA903-87-C-0641.

| *Feature* | *Interpretation* |
|---|---|
| `all`($R$,$C$) | $\lambda x.\, \forall y.\, R(x,y) \rightarrow C(y)$ |
| `atLeast`($k$,$R$) | $\lambda x.\, \exists\, k$ *distinct* $y_i\ \wedge_i R(x,y_i)$ |
| `atMost`($k$,$R$) | $\lambda x.\ \nexists\, k+1$ *distinct* $y_i\ \wedge_i R(x,y_i)$ |
| `filledBy`($R$,$f$) | $\lambda x.\, R(x,f)$ |
| `sameAs`($R_1$,$R_2$) | $\lambda x.\, \forall y.\, R_1(x,y) \leftrightarrow R_2(x,y)$ |

Table 1: Some LOOM Features

contains a set of *descriptions* (one-variable lambda expressions) organized into a subsumption hierarchy. A named description is called a *concept*. The ABox contains facts about individuals. In LOOM a fact is represented either as an assertion that an instance (individual) satisfies a particular description (e.g., "The individual Fred satisfies the concept Person.") or as an assertion that a role relates two individuals (e.g., "50 is a filler of the role 'age of Fred'."). A knowledge base can also contain constraint axioms of the form "*description1* implies *description2*" having the meaning "Instances that satisfy *description1* necessarily satisfy *description2*."

A description is either atomic or composite. We call atomic descriptions *features*—Table 1 lists some LOOM features. A composite description consists of a conjunction of two or more features.[1] An instance satisfies a description if and only if it satisfies each of its feature(s). The LOOM recognizer finds an instantiation relationship between an instance $I$ and a concept $C$ by proving that $I$ satisfies each of the features of $C$.

**Example 1.** Suppose we have concepts `A` and `B`, role relations `R` and `S`, and an individual `I` with definitions and assertions as follows:

```
concept A = atMost(1,S).
concept B = atLeast(1,R)
                and atMost(1,S).
assert A(I), R(I,3), R(I,4).
```

The fact `R(I,3)` implies that `I` satisfies the feature `atLeast(1,R)`, and the fact `A(I)` implies that `I` satisfies the feature `atMost(1,S)`. Hence, `I` is an instance of the concept `B`.

A classifier computes subsumption relationships between a new description and the already classified descriptions in a description hierarchy. The final step in the classification process is to link the newly-classified description into the hierarchy. A description hierarchy is constructed by starting with an empty network, and classifying descriptions one at a time until all of them have found their place relative to each other in the hierarchy. In Example 1 above, the subsumption test invoked by the classifier would determine that `A` subsumes (is more general than) `B`. Classifying `A` first and then `B` or vice-versa would result in the same description hierarchy, with `A` above `B`.

The classical approach to classifying (recognizing) instances utilizes an abstraction/classification (A/C) strategy that relies on the classifier machinery to compute instantiation relationships (Kindermann, 1990; Quantz and Kindermann, 1990; Nebel, 1990). Given an instance $I$, the A/C algorithm operates by (i) computing a description $A_I$ representing an abstraction of $I$, and (ii) classifying $A_I$. $I$ is an instance of each concept that subsumes $A_I$. For example, the description

```
atLeast(2,R) and filledBy(R,3)
    and filledBy(R,4) and atMost(1,S)
```

represents a possible abstraction of the individual $I$ in Example 1. This abstraction is subsumed by concepts `A` and `B`.

The abstract description of an instance has the potential to be much larger (take up more space) than the originally asserted facts about that instance. Some classical classifiers are turning to an incremental approach wherein successively more detailed *partial* abstractions of an instance are generated during a recognition cycle (Kindermann, 1991). However, because of concerns with possible performance problems, the LOOM choice is to implement a completely different recognition algorithm.

We see two potential areas where performance of the classic A/C algorithm may deteriorate:

(1a) Instances in a knowledge base are typically pairwise distinguishable. Hence, their abstractions, if sufficiently detailed, will also be pairwise distinct. Hence, the number of descriptions in a system that utilizes A/C recognition may grow to be proportional to the number of instances. We expect that for knowledge bases containing thousands or tens of thousands of individuals, the A/C strategy will cause the size of the TBox to become unmanageably large.

(1b) With the A/C algorithm, every update to an instance necessitates generating a new abstraction. Unless old abstractions are deleted, frequent updates may cause the cumulative number of abstractions in the net to be much larger than "just" the number of instances.[2]

(2) In the classical approach to recognition, all instantiation relationships are continuously cached and kept up to date. Thus, every update to an instance necessitates recomputing instantiation relationships between the instance and *every* description that it satisfies. As the size of a net grows, this computation becomes increasingly expensive.

To mitigate performance problems (1a) and (1b), LOOM has experimented with recognition strategies designed to reduce the number of additional abstract descriptions generated during the recognition process.

[1] LOOM descriptions can also contain disjunctive expressions, which this paper ignores for the sake of simplicity.

[2] A strategy that deletes each "old" abstraction risks having to generate that same abstraction over and over.

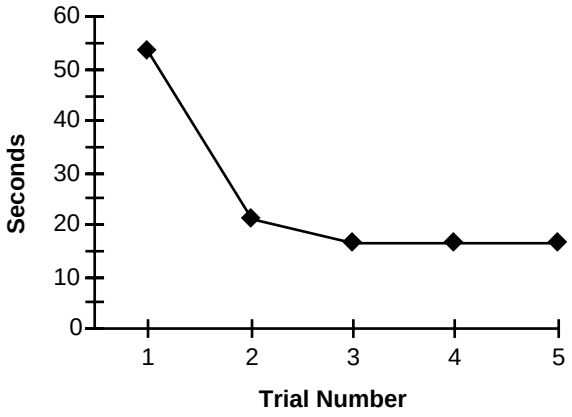


Figure 1: Recognition Speed over 5 Iterations

Section  presents LOOM's "query-based" recognition strategy, wherein calls to a backward chaining query facility are substituted in place of searches for features within an instance's abstract description. To cope with problem (2), LOOM supports an inference mode wherein instantiation relationships are computed only on demand; few or none of the instantiation relationships for an instance are cached. This is discussed in section .

Before looking at experiments that compare different recognition algorithms, we first present the results of an experiment performed using a single recognizer (in this case, the recognizer implemented for LOOM version 1.4). Figure 1 shows results the using a LOOM-based application called DRAMA (Harp *et al.*, 1991). DRAMA is a system that provides intelligent assistance to analysts of logistics databases. The portion of DRAMA that we used in our tests analyzes data for anomolies, such as inconsistencies and noteworthy changes, and logs and categorizes any data anomolies it encounters. In this experiment, the knowledge base was initially loaded with 672 concept definitions, 481 relation definitions, and 677 constraint axioms. We made five trials over the same sequence of instance creations and updates (25 instances were created), clearing the ABox portion of the knowledge base between trials.

Observe that the performance of recognizer improves significantly on the second and following trials. We attribute the improved performance of the 1.4 recognizer to the fact that the recognizer generates significantly fewer augmentations to the description network during the second and subsequent trials. We have observed this "learning" effect in a variety of applications.[3] Because of this effect, the performance figures we present in the sequel include timings for repeated trials over the same data, to provide indications of LOOM's behavior at both ends of the performance spectrum.

[3]However, we have also observed situations where an excess of descriptions, created by prior invocations of the recognizer, can cause system performance to degrade.

## Query-based Recognition

The LOOM recognition algorithm is designed to minimize the number of new abstractions generated as a by-product of the recognition cycle. To compare an instance $I$ with a description $D$, the A/C algorithm generates an abstract description $A_I$ of $I$, and then compares $A_I$ with $D$ using a standard subsumption algorithm. The LOOM strategy is to implement a specialized subsumption test that allows it to directly compare $I$ against $D$, thus avoiding the necessity for generating an abstract description. Further details of the LOOM recognition algorithm can be found in (MacGregor, 1988) and (MacGregor, 1991a).

**Query-based Recognition**—an algorithm for computing the set of concepts satisfied by an instance $I$:

(i) Inherit all features from concepts that $I$ satisfies by direct assertion, and inherit all features implied by those concepts via constraint axioms.

(ii) Compute a normalized set of features representing the unification of the set of inherited features.

(iii) Mark all directly asserted concepts; mark all features in the normalized set; and recursively mark the superiors of marked descriptions.

(iv) Classify $I$ substituting the query based satisfaction test described below in place of the classical subsumption test. Mark each newly satisfied description.

(v) Inherit all features implied by the most specific descriptions that $I$ has been proved to satisfy in step (iv).

(vi) Repeat steps (ii) through (v) until closure is achieved.

**Query-based Satisfaction Test** (assumes concepts and features have been marked prior to the test):

Instance $I$ satisfies a concept $C$ if $I$ satisfies all unmarked features of $C$. Iterate over the unmarked features, and execute a query for each one to test its satisfaction by $I$. For example, if the feature is `atLeast(k,R)`, then retrieve the fillers of the role "`R` of $I$", count them, and return true if the sum is at least `k`. If the feature is `atMost(k,R)`, then return true if the role "`R` of $I$" is closed and if the cardinality of the set of fillers of that role is at most `k`. If the feature is `all(R,B)`, then return true if the role "`R` of $I$" is closed and if each filler of that role satisfies the concept `B`.

Let us apply query-based recognition to the instance in Example 1. In step (i) instance `I` inherits the feature `atMost(1,S)` from the concept `A`. The normalized set computed in step (ii) is just the singleton set containing that feature. In step (iii) we mark the concept `A` and the feature `atMost(1,S)`. In step (iv) we visit the unmarked concept `B`, and test for satisfaction of its

unmarked features—in this case, we test the feature `atLeast(1,R)`. The feature satisfaction test involves retrieving the filler set ({3,4}) of the role "`R` of `I`", computing the cardinality (2 in this case), noting that 2 is at least 1, and returning true. We have proved that `I` satisfies `D` so we mark it. In step (iv) we inherit two features from B. Repeating steps (ii) through (v) reveals no new instantiation relationships for `I`, so the algorithm terminates.

A key difference between query-based recognition and the abstraction/classification strategy is that the former algorithm tends to generate fewer new features. For example, in performing recognition for instance `I` in Example 1, the A/C algorithm generates the feature `atLeast(2,R)` as a part of the abstraction of `I`, while the query-based algorithm generates no new features.

We now discuss an extension to the query-based recognition algorithm that was implemented starting with version 1.4 of LOOM. In versions up through LOOM 1.3, the set of instantiation relationships for each recognized instance was cached by recording a list of the most specific descriptions satisfied by that instance. This list is called the "type" of the instance. In Example 1, the type of `I` is the singleton list containing `B`, (concept `B` is more specific than concept `A`). Starting with version 1.4, we changed the recognition algorithm so that whenever the type of an instance (computed at the end of step (iv)) contains more than one description, a new description is created representing the conjunction of the descriptions in that instance's type. This new description then replaces the list of descriptions. Thus, the LOOM 1.4 recognizer occasionally generates new partial abstractions of instances, and hence bears a greater resemblance to the classical A/C recognizer than does its predecessor.

Why did we, as LOOM implementors, make this change, given that generating new abstractions tends to slow things down? In a LOOM-based parsing application (Kasper, 1989) we observed that unification operations over the same sets of descriptions were being performed repeatedly during step (ii) of the recognition cycle (see (MacGregor, 1991b) for a discussion of description unification). Creation of a new description $D$ representing the conjunction of descriptions in step (iv) has the effect of caching the unification operation for that set of descriptions, because the LOOM data structure representing $D$ stores within it the unification of all features inherited (or implied by) $D$. The creation of these additional conjunction descriptions can have the effect of increasing the likelihood of triggering an optimization in step (ii) wherein the unification operation is eliminated whenever all inherited features derive from a single description.

When we modified the 1.4 recognizer to generate these additional conjunctive descriptions, we observed speed-ups of up to 20 percent in the parsing application. Unfortunately, we observed an opposite effect when we tested the effects of this modification on the

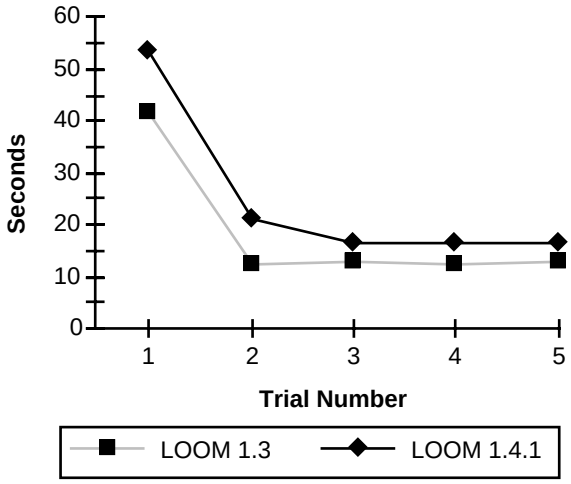


Figure 2: 1.3 Recognizer vs. 1.4 Recognizer

DRAMA application. As illustrated in Figure 2, the effect of generating additional descriptions was a degradation in performance of around 30 percent. This is due primarily to the fact that the 1.4 recognizer generates 83 new descriptions during the first trial, while the 1.3 recognizer generates only 6 new descriptions. However, the degradation persists in the second and subsequent trials, when neither version generates a significant number of new descriptions. In the DRAMA application we found a large variation across the different ABox instances created during the test run, causing the leverage derived from caching unifications to be low. Hence, the 1.4 recognizer ran more slowly because it had to wade through a larger number of descriptions in the hierarchy. We conjecture that the performance of DRAMA using a fully classical recognizer would be even worse than that exhibited by the version 1.4 recognizer.

## Backward Chaining Instantiation Tests

LOOM's query-based recognizer algorithm is designed to address performance problems (1a) and (1b) above, which are concerned with the creation of excessive numbers of descriptions during recognition. The results in Figure 2 indicate that for *some* applications, these concerns are real. However, the query-based recognition algorithm described above does not address performance problem (2). We addressed that problem by implementing a purely backward chaining algorithm for computing instantiation relationships between instances and descriptions. Because the query part of the LOOM recognition algorithm is itself a backward chainer, construction of the new algorithm was much simpler than would have been the case if we had originally implemented a classical recognizer.

Figure 3 shows the results when the DRAMA appli-

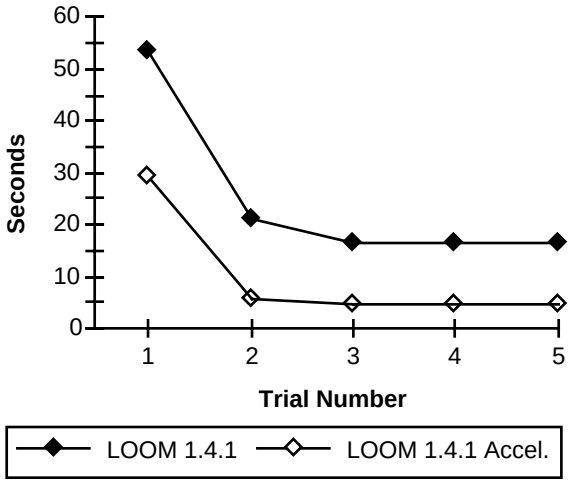


Figure 3: Accelerated vs. Non-accelerated Recognition

cation was run in an accelerated mode where the percentage of concepts for which instantiation relationships were computed by the recognizer was reduced from 79 percent to 36 percent. Computation of the other 64 percent of the instantiation relationships was performed (only on demand) by the backward chaining facility. We observed a decrease of between 46 and 72 percent in the total amount of time spent computing instantiation relationships. Future modifications to LOOM should enable us to completely eliminate the use of the recognizer for the DRAMA application, hopefully resulting in further improvements in that application's performance.

Given the positive results obtained with the accelerated mode, why don't we run all LOOM applications in that mode? The answer is that inference using the LOOM backward chainer is *strictly weaker* than inferences obtained using the LOOM recognizer, because the backward chainer does not implement an analogue of the constraint propagation (Nebel and von Luck, 1988; Nebel, 1990) process that is interleaved between invocations of the recognition cycle. Constraint propagation involves propagating the effects of features satisfied by an instance to adjacent instances in the ABox. For example, suppose a knowledge base contains the axiom

```
A implies all(R,B)
```

where `A` and `B` are concepts, and `R` is a role, and suppose the LOOM recognizer proves that an instance `I` satisfies `A`. `I` will then inherit the feature `all(R,B)`. During the next constraint propagation phase, the implication implicit in that feature is used as the basis for inferring that all fillers of the role "`R` of `I`" necessarily satisfy `B`, i.e., the constraint of satisfying `B` propagates to each of the filler instances.

A backward chaining procedure for evaluating the above axiom would look as follows: "To prove that an instance $x$ satisfies `B`, try to prove that some filler of the role '(inverse of `R`) of $x$' satisfies `A`." In our estimation, extending the capabilities of the backward chainer to include inferences of this sort would be likely to significantly degrade its performance, possibly negating its utility. Instead, LOOM has made the following architectural decision: "An axiom of the form '*description1* implies *description2*' can be applied during backward chaining inference only in the case that *description2* is a concept." In effect, during backward chaining LOOM utilizes only those axioms that are syntactic analogues of Horn clauses. Thus, for example, the axiom "`all(R,B) implies A`" would be applicable during backward chaining, but the axiom "`A implies all(R,B)`" would not.

Other classes of constraint propagation that are applied during recognition, but not during backward chaining, include merging pairs of instances that are inferred to be equivalent, and generating skolem individuals to fill roles in cases where a role filler is known to exist, but where the identify of the filler is not known. Some LOOM applications, notably, natural language parsing, explicitly depend on this kind of constraint propagation (Kasper, 1989)—these applications cannot profit from the accelerated mode. The DRAMA application provides an existence proof of a real application that can execute correctly (and more efficiently) using the weakened form of inference provided by LOOM's backward chaining facility.

## Discussion and Conclusions

One characteristic that differentiates the recognizer algorithms we have considered is the number of abstract descriptions that they generate as a side-effect of recognition. The 1.3 recognizer generates relatively few, the classical algorithm generates relatively many, and the 1.4 recognizer lies somewhere in between. The instrumentation we performed on our algorithms suggests that when 1.3 outperformed 1.4, the difference in speed was due to the fact that 1.3 generated fewer abstract descriptions. Hence, while the jury is still out as to whether 1.3 or 1.4 is the better all around performer, we interpret our tests as casting serious doubt as to the viability of a full-classical recognizer. However, the results of our experiments should be regarded as suggestive rather than as definitive. It is clear that we could have formulated a test application that would produce very different (i.e., opposite) results. The DRAMA application we used has the virtue that it is both real and non-trivial.

With respect to modes of inference, we observe that no single inference strategy can deliver acceptable performance across a spectrum of domain applications. This lesson is of course no surprise to AI practitioners. In many reasoning systems inference is controlled by explicitly marking rules as either forward or backward chaining. We consider embedding control information

within individual rules to be an ultimately bankrupt approach to performance enhancement, for at least two reasons: First, we believe that the task of tuning such rules will become increasingly untenable as the size of rule bases increases. Second, we believe that it will eventually become commonplace that a rule base will be shared by two or more applications that need to apply different modes of inference to that same set of rules. In LOOM, the application chooses which mode of inference is most suitable. Thus, for example, a LOOM-based diagnosis system might choose to run in backward chaining mode, while a natural language explanation system running against the same rule base might choose to invoke the recognizer to assist its own reasoning processes.

The most general conclusion indicated by our experiments is that complex, multi-modal classifier architectures appear to be faster than simple (more elegant) architectures, at least for uniprocessor-based systems. This is basically a negative result, since it increases our estimation of the difficulty involved in building a knowledge representation system that is both general purpose and efficient.

## Acknowledgement

The authors wish to thank Tom Russ for his help in producing the graphs used in this paper, and Craig Knoblock and Bill Swartout for their criticisms of an earlier draft of this paper.

## References

Brachman, R.J. and Schmolze, J.G. 1985. An overview of the KL-ONE knowledge representation system. *Cognitive Science* 171–216.

Harp, B.; Aberg, P.; Neches, R.; and Szekely, P. 1991. DRAMA: An application of a logistics shell. In *Proceedings of the Annual Conference on Artificial Intelligence Applications for Military Logistics*, Williamsburg, Virginia. American Defense Preparedness Association. 146–151.

Kasper, Robert 1989. Unification and classification: An experiment in information-based parsing. In *Proceedings of the International Workshop on Parsing Technologies*, Pittsburg, PA.

Kindermann, Carsten 1990. Class instances in a terminological framework—an experience report. In Marburger, H., editor 1990, *GWAI-90. 14th German Workshop on Artificial Intelligence*, Berlin, Germany. Springer-Verlag. 48–57.

Kindermann, Carsten 1991. Personal communication.

MacGregor, Robert and Burstein, Mark H. 1991. Using a description classifier to enhance knowledge representation. *IEEE Expert* 6(3):41–46.

MacGregor, Robert 1988. A deductive pattern matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*, St. Paul, MINN. AAAI. 403–408.

MacGregor, Robert 1990. The evolving technology of classification-based knowledge representation systems. In Sowa, John, editor 1990, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan-Kaufman. chapter 13.

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

MacGregor, Robert 1991b. Using a description classifier to enhance deductive inference. In *Proceeding Seventh IEEE Conference on AI Applications*, Miami, Florida. IEEE. 141–147.

Nebel, Bernhard and von Luck, Kai 1988. Hybrid reasoning in BACK. *Methodologies for Intelligent Systems* 3:260–269.

Nebel, Bernhard 1990. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Germany.

Patel-Schneider, P.F.; Owsnicki-Klewe, B.; Kobsa, A.; Guarino, N.; MacGregor, R.; Mark, W.S.; McGuiness, D; Nebel, B.; Schmiedel, A.; and Yen, J. 1990. Term subsumption languages in knowledge representation. *The AI Magazine* 11(2):16–23.

Quantz, Joachim and Kindermann, Carsten 1990. Implementation of the BACK system version 4. KIT Report 78, Department of Computer Science, Technische Universitaat Berlin, Berlin, Germany.