Optimising Description Logic Subsumption

Ian Horrocks
Department of Computer Science,
University of Manchester[1]

and

Peter F. Patel-Schneider
Bell Labs Research,
Murray Hill, NJ, U.S.A.

November, 1998

[1]Part of this work was carried out while being a guest at IRST, Trento.

Abstract

Effective optimisation techniques can make a dramatic difference in the performance of knowledge representation systems based on expressive description logics. With currently-available desktop computers, systems that incorporate these techniques can effectively reason in description logics with intractable inference. Because of the correspondence between description logics and propositional modal logic, difficult problems in propositional modal logic can be effectively solved using the same techniques.

1 Introduction

Description logics are a logical formalism for the representation of knowledge about individuals and descriptions of individuals [8]. Description logics represent and reason with descriptions similar to "all people whose friends are both doctors and lawyers" or "all people whose children are doctors or lawyers or who have a child who has a spouse".

The computations performed by systems that implement description logics are based around determining whether one description is more general than (subsumes) another. There have been various schemes for computing this subsumption relationship, depending on the expressive power of the description logic and the degree of completeness of the system. As description logic systems perform numerous subsumption checks in the course of their operations, they need to have a highly-optimised subsumption checker.

Schild [44] has shown that determining subsumption in expressive description logics is equivalent to determining satisfiability of formulae in propositional modal or dynamic logics. Several description logic systems have been built for such description logics, and thus include what is essentially a satisfiability checker for some propositional modal logic; examples of such systems include KRIS [5] and CRACK [10]. These two systems have incorporated a number of optimisations to achieve better performance of their subsumption checkers.

Description logic systems are also optimised in other ways. In particular, their operations are arranged so as to avoid potentially-costly subsumption checks whenever possible. There are also optimisations that are particular to description logics, having to do with the nature of the representation of knowledge in a description logic [2], but these have little or nothing to do with optimising subsumption in general.

Two systems that explore the subsumption optimisations required to build an expressive description logic system are FaCT [29], a full description logic system, and DLP [37], an experimental system providing only a limited description logic interface.[1] These two systems incorporate a range of known, adapted, and novel optimisation techniques in their subsumption checkers. The optimisation techniques include: lexical normalisation, semantic branching search, simplification, dependency directed backtracking, heuristic guided search and caching.

These optimisation techniques make a dramatic difference to the performance of a description logic system. As evidence, KRIS is not able to load a modified version of the GALEN knowledge base because it gets stuck trying to determine one of the thousands of subsumptions required to load the knowledge base. With their higher levels of optimisation, FaCT and DLP are able to quickly load this knowledge base, classifying over two thousand definitions in about two hundred seconds for FaCT and under 100 seconds for DLP.

[1] DLP is designed to facilitate the investigation of new and improved optimisation techniques: it is written in a functional language and is highly configurable.

We have also performed experiments with both FaCT and DLP on several test suites of propositional modal formulae.[2] The optimisations built into the two systems qualitatively change their behaviour on the test suites, indicating that the optimisations have considerable utility simply taken as optimisations for reasoning in propositional modal logics.

2 Background

FaCT and DLP are designed to build and maintain taxonomies of named concepts. Given a collection of definitions of named concepts and statements about these concepts, they determine the subsumption partial order for the named concepts. To do this, FaCT and DLP have to determine subsumption relationships between descriptions in a description logic. Both FaCT and DLP implement expressive description logics, with subsumption problems that are known to be highly intractable in the worst case.

FaCT implements a superset of the description logic $\mathcal{ALC}_{R+}$ [43], an extension of $\mathcal{ALC}$ [45] that distinguishes the set of transitive roles, $\mathbf{R}_+$; in FaCT, this set is defined by axioms of the form $R \in \mathbf{R}_+$. To this logic, FaCT adds role and concept inclusion axioms [29]. Role inclusion axioms are of the form $R \sqsubseteq S$, where R and S are role names, and can be used to define a primitive role hierarchy. Concept inclusion axioms are of the form $C \sqsubseteq D$, where C and D are concept expressions, and can be used to assert arbitrary subsumption relationships.[3]

A standard Tarski style model theoretic semantics is used to interpret concepts and roles, and to justify subsumption inferences [47, 3]. The meaning of concepts and roles is given by an interpretation $\mathcal{I}$, which is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain (a set) and $\cdot^{\mathcal{I}}$ is an interpretation function. The interpretation function maps each concept to a subset of $\Delta^{\mathcal{I}}$ and each role to a binary relation (or equivalently a set valued function): $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ ($R^{\mathcal{I}} : \Delta^{\mathcal{I}} \longrightarrow 2^{\Delta^{\mathcal{I}}}$). The syntax and semantics of FaCT's concepts, roles and axioms is given in Table 1. In this table A is an atomic concept, C and D are concept expressions, and R and S are roles.

The Tarski style semantics is a simple transformation of the possible world semantics for propositional modal logics. In this transformation elements of the domain correspond to possible worlds, atomic concepts correspond to propositional variables, and roles correspond to modalities. This transformation shows that fragments of FaCT (and $\mathcal{ALC}_{R+}$) correspond to $\mathbf{K_{(m)}}$ and $\mathbf{K4_{(m)}}$, with transitive roles in FaCT being used for $\mathbf{K4_{(m)}}$ and non-transitive roles for $\mathbf{K_{(m)}}$. FaCT can also express formulae in $\mathbf{KT_{(m)}}$ and $\mathbf{S4_{(m)}}$ via the usual encoding that maps $\forall R.C$ into $C \sqcap \forall R.C$ and $\exists R.C$ into $C \sqcup \exists R.C$.

DLP implements two description logics of differing expressive power. The less expressive logic is equivalent to the logic implemented by FaCT. Here DLP is essentially a reimplementation (with improved data structures and some additional optimisation) of FaCT's subsumption reasoner. The more expressive logic implemented by DLP includes propositional dynamic logic (PDL) [16], augmenting PDL with number restrictions on atomic roles. The syntax and semantics of the more expressive logic is given in Table 2. In this table A is

[2] We augmented FaCT and DLP with an interface that allows these systems to perform directly as reasoners for various propositional modal logics.

[3] Concept equality, written $C \doteq D$, can be asserted using a symmetrical pair of inclusion axioms $C \sqsubseteq D$ and $D \sqsubseteq C$.

	Syntax	Semantics
Concepts	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	$\top$	$\Delta^{\mathcal{I}}$
	$\bot$	$\emptyset$
	$\neg C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
	$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$
Roles	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Axioms	$R \in \mathbf{R}_{+}$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Table 1: Syntax and semantics of the FaCT description logic

an atomic concept, C and D are concept expressions, P is an atomic role, and R and S are arbitrary roles.

A simple transformation of the semantics demonstrates that this more expressive logic contains propositional dynamic logic as a subset [44]. It is also easy to see that this description logic is a superset of the logic used in FaCT, as both transitive roles and role inclusions can be simulated using role expressions. For example, a transitive role R can be simulated by replacing R with R^{+} wherever it appears in a concept expression, while a role inclusion axiom $R \sqsubseteq S$ can be simulated by replacing S with $(R \sqcup S)$, wherever it appears in a concept expression.

Determining subsumption in $\mathcal{ALC}_{R^+}$ (and equivalently in $\mathbf{S4}$) is PSPACE-complete [43, 24]. Adding either role or concept inclusion axioms results in EXPTIME-complete subsumption[4] [38, 48, 29]. Determining subsumption in propositional dynamic logic is also EXPTIME-complete [39]. These and related complexity problems have lead some developers of description logic systems to use less-expressive description logics [9]. However, although the theoretical complexity results are discouraging, empirical analyses of real applications have shown that the kinds of construct that lead to worst case intractability rarely occur in practice [35, 25, 46, 28], and it has proved possible to build practical description logic systems based on expressive description logics [5, 10, 29].

Systems that are based on description logics like these generally determine whether a subsumption holds by transforming the subsumption problem into a satisfiability problem in the obvious manner: concept C subsumes concept D if and only if the concept $\mathsf{D} \sqcap \neg\mathsf{C}$ is not satisfiable. The systems then solve this problem by attempting to construct a model for the concept, just as a tableaux satisfiability checker for a propositional modal logic attempts to construct a model for a formula. The model is represented by a tree in which nodes represent individuals and edges represent roles. Each node is labeled with a set of concepts—we will use $\mathcal{L}(x)$ to denote the label of a node x. The meaning of the label is that the individual represented by x must be in the extension of every concept in $\mathcal{L}(x)$. Edges are labeled with

[4]The addition of role inclusion axioms also allows concept inclusion axioms to be dealt with adding them to concept expressions, a technique known as *internalisation* [1, 29].

	Syntax	Semantics
Concepts	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	$\top$	$\Delta^{\mathcal{I}}$
	$\bot$	$\emptyset$
	$\neg C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
	$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$
	$\geqslant n\mathsf{P}$	$\{d \in \Delta^{\mathcal{I}} : \mid \mathsf{P}^{\mathcal{I}}(d) \mid\ \geq n\}$
	$\leqslant n\mathsf{P}$	$\{d \in \Delta^{\mathcal{I}} : \mid \mathsf{P}^{\mathcal{I}}(d) \mid\ \leq n\}$
Roles	P	$\mathsf{P}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	$R \sqcup S$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$
	$/C$	$\Delta^{\mathcal{I}} \times C^{\mathcal{I}}$
	$R \circ S$	$R^{\mathcal{I}} \circ S^{\mathcal{I}}$
	R^+	$\bigcup_{n \geq 1} {R^{\mathcal{I}}}^n$
Axioms	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Table 2: Syntax and semantics of the DLP description logic

role names—if an edge $\langle x, y\rangle$ is labeled R, then y is said to be an R-successor of x. If x is connected to y via an arbitrary sequence of edges then x is said to be an ancestor of y. The tree is said to contain a *clash* if for some node x and some concept C, either $\{C, \neg C\} \subseteq \mathcal{L}(x)$ or $\bot \in \mathcal{L}(x)$. From a modal logic perspective, one can view nodes as representing possible worlds, a node label as a set of formulae that must evaluate to true at the world, (labeled) edges as (multi-)modal relationships, and clashes as obvious contradictions.

To test the satisfiability of a concept (formula) D, the basic algorithm initialises a tree to contain a single node x, with $\mathcal{L}(x) = \{D\}$, representing an individual that must be in the extension of D. The tree is then expanded by applying rules that either extend $\mathcal{L}(x)$ for some node x or add new leaf nodes. Disjunctive concepts ($C \sqcup D$) give rise to non-deterministic expansion, existential role concepts ($\exists R.C$) cause the creation of a new R-successor nodes, and universal role concepts ($\forall R.C$) extend the labels of R-successor nodes. The tree is fully expanded when none of the expansion rules can be applied. If a fully expanded and clash-free tree can be found then the algorithm returns *satisfiable*; otherwise it returns *unsatisfiable*.

In the FaCT algorithm, transitive roles and the role hierarchy are dealt with by a more complex rule for expanding universal role concepts and by incorporating a check for cycles (which could otherwise cause non-termination) in the rule for expanding existential role concepts. In description logic algorithms, such cycle checks are often called *blocking*. In FaCT, a node x is said to be blocked if there is some ancestor node y such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$; if this is the case then the two nodes can be collapsed into a cycle ($x = y$).[5]

The expansion rules for the FaCT algorithm are summarised in Figure 1. In order to simplify the rules it is assumed that x is a node in the tree; that a rule is not applicable if applying the rule would not change the tree; and that for two roles R and S, $R \sqsubseteq S$ if either $R = S$, $R \sqsubseteq S$ is an axiom, or there is a role P such that $P \sqsubseteq S$ is an axiom and $R \sqsubseteq P$.

[5] In this description logic all cycles are good—they can be interpreted as valid cyclical models.

$\sqcap$-rule:	if $C_1 \sqcap C_2 \in \mathcal{L}(x)$ then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if $C_1 \sqcup C_2 \in \mathcal{L}(x)$ then nondeterministically add either C or D to $\mathcal{L}(x)$
$\exists$-rule:	if $\exists R.C \in \mathcal{L}(x)$ and x is not blocked and there is no R-successor y of x with $C \in \mathcal{L}(y)$ then create a new R-successor z of x with $\mathcal{L}(z) = \{C\}$
$\forall$-rule:	if $\forall R.C \in \mathcal{L}(x)$ then for each S-successor y of x such that $S \sqsubseteq R$, $\mathcal{L}(y) := \mathcal{L}(y) \cup \{C\} \cup \{\forall P.D \mid P \in \mathbf{R}_+,\ S \sqsubseteq P \text{ and } P \sqsubseteq R\}$

Figure 1: Expansion rules for FaCT algorithm

Moreover, y is considered to be an R-successor of x if the edge $\langle x, y\rangle$ is labeled with a role S such that $S \sqsubseteq R$. Concept inclusion axioms are also ignored because, as noted above, they can be dealt with by internalisation.

In practice, expansion is performed one node at a time, with the expansion of successors being postponed until the current node is fully expanded. Successor nodes can then be expanded one at a time and discarded once their local satisfiability has been determined. Nondeterminism in the $\sqcup$-rule is implemented by a depth first search, halting either when a model is found or when all possible choices have been explored and found to lead to a clash. Full details of the algorithm along with a proof of its soundness and completeness can be found in [30].

The algorithm for the logic implemented by DLP in its more expressive configuration is more complex due to the presence of the transitive closure operator and of number restrictions. However, as the optimisation and testing described in the rest of this paper refer to DLP in its less expressive configuration,[6] the algorithm that DLP implements for its more expressive logic will not be described in detail. The main diferences between this algorithm and the FaCT algorithm described above are:

1. expanding a concept of the form $\exists R^+.C$ is treated as a non-deterministic choice between $\exists R.C$ and $\exists R.(\exists R^+.C)$;
2. cycles caused by concepts of the form $\exists R^+.C$ must be checked to see if they actually satisfy the concept (a good cycle) or simply postpone satisfying it until a cycle is encountered (a bad cycle);
3. number restrictions can give rise to extra non-deterministic choice when $\leqslant nP$ concepts restrict the number of P-successors that can be created;
4. the definition of a clash is extended to include the case where $\{\geqslant mP, \leqslant nP\} \subseteq \mathcal{L}(x)$ and $m > n$.

[6]The main reason for this is a lack of suitable test data. It is hoped that the availability of DLP will lead to the development of such data, in particular knowledge bases that use its more expressive description logic.

Further details can be found in [37, 1, 13].

The implementations of both FaCT and DLP include a *term classifier* or *Tbox* that can build and maintain a concept hierarchy—a partial ordering of named concepts based on the subsumption relation. Named concepts are defined by axioms of the form $\mathsf{CN} \doteq C$ (equivalent to $\mathsf{CN} \sqsubseteq C$ and $C \sqsubseteq \mathsf{CN}$) and $\mathsf{CN} \sqsubseteq C$, where CN is a concept name.

The task of computing the partial ordering of named concepts for a given knowledge base is itself amenable to a range of optimisations. In particular, concept definitions can, in general, be dealt with much more efficiently than other axioms using a technique called *unfolding* [2], the basic idea being simply to substitute names with their corresponding definition wherever they occur. These techniques are not, however, relevant to the optimisation of the underlying subsumption (satisfiability) tester, and are not studied in this paper.

3 Optimisation Techniques

The basic algorithm given above is too slow to form the basis of a useful description logic system. We have therefore investigated and employed a range of known, adapted and novel optimisations that improve the performance of the satisfiability testing algorithm. These optimisations include: lexical normalisation, semantic branching search, simplification, dependency directed backtracking, heuristic guidance of the search, and caching. Each of these techniques will be described in detail in the following sections.

3.1 Lexical Normalisation

Theoretical descriptions of tableaux algorithms generally assume that the concept expression to be tested is in negation normal form, with negations applying only to atomic concepts [27, 4, 11]. This simplifies the (description of the) algorithm but it means that a clash will only be detected when an atomic concept and its negation occur in the same node label.

For example, when testing the satisfiability of the concept expression

$$\exists R.(C \sqcap D) \sqcap \forall R.\neg C,$$

where C is an atomic concept, a clash would be detected when the algorithm creates an R-successor y because $\{C, \neg C\} \subseteq \mathcal{L}(y)$. However, if C is a concept expression, then the clash would not be detected immediately because $\neg C$ would have been transformed into negation normal form. If C is large this could lead to costly wasted work.

The late detection of clashes can be addressed by transforming concept expressions (and recursively their sub-expressions) into a lexically normalised form, and by identifying lexically equivalent expressions. All concept expressions can then be treated equally with a clash being detected whenever a concept expression and its negation occur in the same node label.[7] In this lexically normalised form, concept expressions consist only of atomic concepts, conjunction concepts, universal role concepts, and their negations. Moreover,conjunctions are treated as sets so that their equivalence is recognised regardless of ordering, repetition or nesting of conjuncts; a conjunction in this form will be written $\sqcap\{C_1, \ldots, C_n\}$, where $\{C_1, \ldots, C_n\}$ is the set of conjuncts. The full set of normalisation rules employed by FaCT and DLP are given in Table 3.

[7] KRIS addresses the same problem, in a less complete manner, by lazily expanding named concepts, and retaining their names in node labels [2].

Concept expression	Normal form
$\bot$	$\neg\top$
$C \sqcup D$	$\neg(\neg C \sqcap \neg D)$
$\exists R.C$	$\neg(\forall R.\neg C)$
$\neg\neg C$	C
$C \sqcap D$	$\sqcap\{C, D\}$
$\sqcap\{\sqcap\{C_1, \ldots, C_n\}, \ldots\}$	$\sqcap\{C_1, \ldots, C_n, \ldots\}$
$\sqcap\{C\}$	C

Table 3: Normalisation rules for FaCT and DLP

Concept expression	Simplification
$\forall R.\top$	$\top$
$\sqcap\{\top, C, \ldots\}$	$\sqcap\{C, \ldots\}$
$\sqcap\{\neg\top, \ldots\}$	$\neg\top$
$\sqcap\{C, \neg C, \ldots\}$	$\neg\top$

Table 4: Lexical simplification rules for FaCT and DLP

The normalisation process can also include lexical simplifications that eliminate redundancy and help to identify obvious satisfiability and unsatisfiability; those performed by FaCT and DLP are shown in Table 4. Other simplifications, such as $\sqcap\{\forall R.C, \forall R.C, \ldots\} \longrightarrow \sqcap\{\forall R. \sqcap \{C, D\}, \ldots\}$, would also be possible.

The detection and handling of contradictory conjuncts can make a dramatic difference in solution time. In extreme cases the need for a tableau expansion can be completely eliminated by simplifying the expression to $\top$ or $\neg\top$. Efficiency can be further enhanced by tagging each lexically distinct expression with a unique code so that equivalent expressions can be identified simply by comparing tags,[8] or by uniquely storing expressions.

Tableau expansion of concepts in this form is no more complex than if they are in negation normal form: $\neg(\forall R.C)$ can be dealt with in the same way as $\exists R.\neg C$, while $\neg\sqcap\{C_1, \ldots, C_n\}$ can be dealt with in the same way as $(\neg C_1 \sqcup \ldots \sqcup \neg C_n)$. For example, the expression $\exists R.(C \sqcap D) \sqcap \forall R.\neg C$ would be transformed into $\sqcap\{\neg(\forall R.\neg \sqcap \{C, D\}), \forall R.\neg C\}$, and the $\neg(\forall R.\neg \sqcap \{C, D\})$ term would lead directly to the creation of an R-successor whose label contained both C and $\neg C$. As the two occurrences of C will be lexically normalised and tagged as the same concept, a clash will immediately be detected, regardless of the structure of C.

3.2 Semantic Branching Search

Standard tableaux algorithms are inherently inefficient because they use a search technique based on syntactic branching. When expanding the label of a node x, syntactic branching works by choosing an unexpanded disjunction $(C_1 \sqcup \ldots \sqcup C_n)$ in $\mathcal{L}(x)$ and searching the different models obtained by adding each of the disjuncts $C_1, \ldots, C_n$ to $\mathcal{L}(x)$ [22]. As the alternative branches of the search tree are not disjoint, there is nothing to prevent the

[8] A similar technique is used in KSAT, but without the benefit of tagging [23].

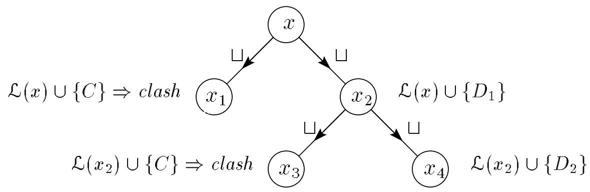

Figure 2: Syntactic branching

recurrence of an unsatisfiable disjunct in different branches. The resulting wasted expansion could be costly if discovering the unsatisfiability requires the solution of a complex sub-problem. For example, tableau expansion of a node x, where $\{(C \sqcup D_1), (C \sqcup D_2)\} \subseteq \mathcal{L}(x)$ and C is an unsatisfiable concept expression, could lead to the search pattern shown in Fig. 2, in which the unsatisfiability of C must be demonstrated twice.

This problem can be dealt with by using a semantic branching technique adapted from the Davis-Putnam-Logemann-Loveland procedure (DPL) commonly used to solve propositional satisfiability (SAT) problems [12, 21]. Instead of choosing an unexpanded disjunction in $\mathcal{L}(x)$, a single disjunct D is chosen from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two possible sub-trees obtained by adding either D or $\neg D$ to $\mathcal{L}(x)$ are then searched. Because the two sub-trees are strictly disjoint, there is no possibility of wasted search as in syntactic branching. If D is a large concept, the addition of $\neg D$ could result in a significantly larger search space. However, as we will see in Section 4, this does not seem to be a significant problem in practice, and semantic branching consistently wins out.

An additional advantage of using a DPL based search technique is that a great deal is known about the implementation and optimisation of this algorithm. In particular, both *simplification* and *heuristic guided search* can be used to try to minimise the size of the search tree.

3.3 Simplification

Simplification is a technique used to reduce the amount of non-determinism (branching) in the expansion of node labels. Before any non-deterministic expansion of a node label $\mathcal{L}(x)$ is performed, disjunctions (actually negated conjunctions) in $\mathcal{L}(x)$ are examined, and if possible simplified. The simplification used by both FaCT and DLP is to deterministically expand disjunctions in $\mathcal{L}(x)$ that present only one expansion possibility and to detect a clash when a disjunction in $\mathcal{L}(x)$ has no expansion possibilities.

This simplification has been called boolean constraint propagation (BCP) [20]. In effect, the inference rule

$$\frac{\neg C_1, \ldots, \neg C_n, C_1 \sqcup \ldots \sqcup C_n \sqcup D}{D}$$

is being used to simplify the expression represented by $\mathcal{L}(x)$.

For example, given a node x such that

$$\{(C \sqcup (D_1 \sqcap D_2)), (\neg D_1 \sqcup \neg D_2), \neg C\} \subseteq \mathcal{L}(x),$$

BCP deterministically expands the disjunction $(C \sqcup (D_1 \sqcap D_2))$ because $\neg C \in \mathcal{L}(x)$. The deterministic expansion of $(D_1 \sqcap D_2)$ adds both D_1 and D_2 to $\mathcal{L}(x)$, allowing BCP to identify $(\neg D_1 \sqcup \neg D_2)$ as a clash without any branching having occurred.

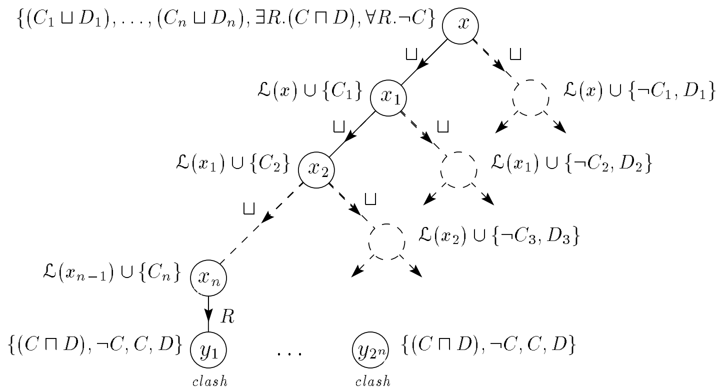

Figure 3: Thrashing in backtracking search

3.4 Dependency Directed Backtracking

Inherent unsatisfiability concealed in sub-problems can lead to large amounts of unproductive backtracking search known as thrashing. The problem is exacerbated when blocking is used to guarantee termination, because blocking may require that sub-problems only be explored after all other forms of expansion have been performed. For example, expanding a node x, where

$$\mathcal{L}(x) = \{(C_1 \sqcup D_1), \dots, (C_n \sqcup D_n), \exists R.(C \sqcap D), \forall R.\neg C\},$$

would lead to the fruitless exploration of 2^n possible R-successors of x before the inherent unsatisfiability is discovered.[9] The search tree created by the tableau expansion algorithm is illustrated in Fig. 3.

This problem is addressed by adapting a form of dependency directed backtracking called *backjumping*, which has been used in solving constraint satisfiability problems [6] (a similar technique was also used in the HARP theorem prover [36]). Backjumping works by labeling concept expressions with a dependency set indicating the branching points on which they depend. A concept expression $C \in \mathcal{L}(x)$ depends on a branching point if C was added to $\mathcal{L}(x)$ by the branching point or if $C \in \mathcal{L}(x)$ was generated by an expansion rule (including simplification) that depends an another concept expression $D \in \mathcal{L}(y)$, and $D \in \mathcal{L}(y)$ depends on the branching point. A concept expression $C \in \mathcal{L}(x)$ depends on a concept expression $D \in \mathcal{L}(y)$ when C was added to $\mathcal{L}(x)$ by a deterministic expansion that used $D \in \mathcal{L}(y)$, e.g., if $A \in \mathcal{L}(x)$ was derived from the expansion of $(A \sqcap B) \in \mathcal{L}(x)$, then $A \in \mathcal{L}(x)$ depends on $(A \sqcap B) \in \mathcal{L}(x)$.

When a clash is discovered, the dependency sets of the clashing concepts 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.

In more detail, when a clash is detected the dependency sets of the clashing concepts are unioned and backtracking is initiated. During backtracking, each branching point encountered is checked against the dependency set to see if it is a member. If it is not in the

[9] Note that if $\mathcal{L}(x)$ simply included $\exists R.C$ instead of $\exists R.(C \sqcap D)$, then the inherent unsatisfiability would have been detected immediately due to the lexical normalisation of $\exists R.C$ as $\neg\forall R.\neg C$.

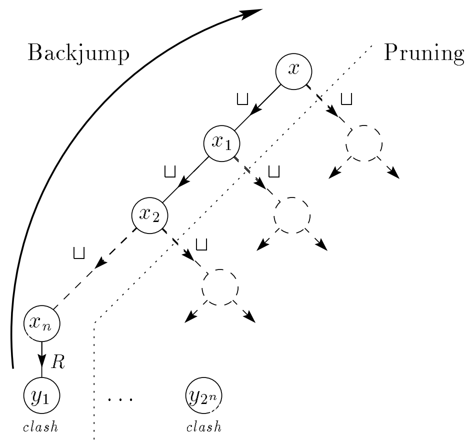

Figure 4: Pruning the search using backjumping

dependency set, then the other branch is ignored and backtracking continues. If the branching point is in the dependency set, and the other branch has not been explored, then backtracking stops and the algorithm proceeds with the exploration of the second branch. If both branches have already been explored, then the dependency sets from the two branches are unioned and backtracking continues.

For example, when expanding the node x from the previous example, the search algorithm will perform a sequence of n branches, eventually leading to the node x_n with $\{\exists R.(C \sqcap D), \forall R.\neg C\} \subset \mathcal{L}(x_n)$. When $\exists R.(C \sqcap D) \in \mathcal{L}(x_n)$ is expanded the algorithm will generate an R-successor y_1 with $\mathcal{L}(y_1) = \{(C \sqcap D), \neg C\}$. The concept expression $(C \sqcap D)$ will then be expanded and a clash will be detected because $\{C, \neg C\} \subset \mathcal{L}(y_1)$. As neither C nor $\neg C$ in $\mathcal{L}(y_1)$ will have the branching points leading from x to x_n in their dependency sets, the algorithm can either return *unsatisfiable* immediately (if both the dependency sets were empty), or backtrack to the most recent branching point on which one of C or $\neg C$ did depend, without exploring the alternative branches at any of the intervening branching points. Figure 4 illustrates how the search tree below x is pruned by backjumping, with the number of R-successors explored being reduced by $2^n - 1$.

3.5 Heuristic Guided Search

Heuristic techniques can be used to guide the search in a way that tries to minimise the size of the search tree. A method that is widely used in DPL SAT algorithms is to branch on the disjunct that has the Maximum number of Occurrences in disjunctions of Minimum Size—the well known MOMS heuristic [20]. By choosing a disjunct that occurs frequently in small disjunctions, the MOMS heuristic tries to maximise the effect of BCP. For example, if the label of a node x contains the unexpanded disjunctions $C \sqcup D_1$ through $C \sqcup D_n$, then branching on C leads to their deterministic expansion in a single step: when C is added to $\mathcal{L}(x)$, all of the disjunctions are fully expanded and when $\neg C$ is added to $\mathcal{L}(x)$, BCP will expand all of the disjunctions, causing $D_1, \dots, D_n$ to be added to $\mathcal{L}(x)$. Branching first on any of $D_1, \dots, D_n$, on the other hand, would only cause a single disjunction to be expanded.

There are several variants of the MOMS heuristic, including the heuristic from Jeroslow and Wang [34]. The Jeroslow and Wang (JW) heuristic considers all occurrences of a disjunct, weighting them according to the size of the disjunction in which they occur. The heuristic then selects the disjunct with the highest overall weighting, again with the objective of maximising BCP and reducing the size of the search tree.

Unfortunately MOMS-style heuristics interact adversely with the backjumping optimisation because they do not prefer older disjuncts, i.e., disjuncts that result from earlier branching points and that will thus lead to more effective pruning if a clash is discovered [29]. Moreover, MOMS-style heuristics are of little value themselves in description logic systems because they rely for their effectiveness on finding the same disjuncts recurring in multiple unexpanded disjunctions: this is likely in hard propositional problems, where the disjuncts are propositional variables, and where the number of different variables is usually small compared to the number of disjunctive clauses (otherwise problems would, in general, be trivially satisfiable); it is unlikely in concept satisfiability problems, where the disjuncts are concept expressions, and where the number of different concept expressions is usually large compared to the number of disjunctive clauses. As a result, these heuristics will often discover that all disjuncts have similar or equal priorities, and the guidance they provide is not particularly useful.

An alternative strategy is to employ an *oldest-first* heuristic that tries to maximise the effectiveness of backjumping by using dependency sets to guide the expansion. Whenever a choice is presented, the heuristic tries to choose a disjunction whose dependency set does not include any recent branching points. This technique can be used both when selecting disjuncts on which to branch and when selecting the order in which R-successors are expanded. The oldest-first heuristic can be combined with a MOMS-style heuristic (such as the JW heuristic) by using the MOMS-style heuristic to select a disjunct from one of the oldest disjunctions or from all of the oldest disjunctions.

3.6 Caching

During a satisfiability check there may be many successor nodes created. These nodes tend to look very similar, particularly as the R-successors for a node x each have the same concept expressions for the universal role concepts in $\mathcal{L}(x)$. Considerable time can thus be spent re-performing the computations on nodes that end up having the same label. As the satisfiability algorithm only cares whether a node is satisfiable or not, this time is wasted.

If successors are only created when other possibilities at a node are exhausted, then the entire set of concept expressions that come into a node label can be generated at one time.[10] The satisfiability status of the node is then completely determined by this set of concept expressions. If there exists another node with the same set of initial formulae then the two nodes will have the same satisfiability status. Work need be done only on one of the two nodes, potentially saving a considerable amount of processing, as not only is the work at one of the nodes saved, but also the work at any of the successors of this node.

The downside of caching is that the dependency information required for backjumping cannot be effectively calculated for the nodes that are not expanded. This happens because the dependency set of any clash detected depends on the dependency sets of the incoming concept expressions, which may differ between the two nodes. Backjumping can still be

[10]This may by required by blocking, and is generally a good idea anyway as it reduces the number of nodes that are created. Both FaCT and DLP operate in this manner.

performed, however, by combining the dependency sets of all incoming concept expressions and using that as the dependency set for the unsatisfiable node.

Another problem with caching is that it requires that nodes be retained until the end of a satisfiability test (or longer, if the results are to be used in later satisfiability tests). This extra storage consumption can be reduced by only storing the sets of concepts and their satisfiability conditions, instead of storing a complete node, but caching can still require considerable extra storage.

DLP uses the device of storing just sets of concepts and their satisfiability condition, which can be satisfiable, unsatisfiable, or unknown. In fact, as DLP uniquely stores concept expressions, it performs caching by constructing the conjunction of the initial set of concepts in a node label and treating it as a concept expression. If this concept expression is not in the concept store, then it is added and its satisfiability condition set to unknown; if it is already in the concept store, then its existing satisfiability condition is simply accessed. If the resulting satisfiability condition is is either satisfiable or unsatisfiable, then this is used instead of expanding the node; otherwise expansion continues, and when the node's satisfiability is determined the concept store is updated accordingly.

4 Comparison with Other Systems

To analyse the effectiveness of the above optimisations, we compared the performance of FaCT and DLP against the performance of another description logic system (KRIS [5]) and a propositional modal logic prover (KSAT [23]). We used KRIS here as an example of an unoptimised description logic system. Other unoptimised description logic systems, such as Crack [10], give similar or worse results. We used KSAT as an example of a heavily-optimised reasoner for propositional modal logics. However, neither KRIS nor KSAT can be used on all our tests. Neither handle transitive roles, and KSAT cannot handle a knowledge base.

We used two propositional modal test suites: the test suite from the Tableaux'98 propositional modal logic comparison [26][11] and a collection of random formulae like those generated by Hustadt and Schmidt [33]. These test suites are not ideal, but we were unable to find many description logic knowledge bases that were suitable for testing the performance of FaCT and DLP.

The Tableaux'98 test suite consists of nine classes of formulae (e.g. *branch*), in both provable (p) and non-provable (n) forms,[12] for each of $\mathbf{K}$, $\mathbf{KT}$, and $\mathbf{S4}$. For each class of formula, 21 examples of supposedly exponentially increasing difficulty have been generated from a basic pattern that incorporates features intended to make the formulae hard to solve. The test suite tries to emphasise modal reasoning, and for most classes of formulae the increase in difficulty is achieved, at least in part, by increasing the modal depth; the maximum modal depth of $\mathbf{K}$-*branch* formulae, for example, increases from 2 for the easiest problem to 22 for the hardest problem. However, in some cases the increase in difficulty is due purely to harder (or at least larger) propositional reasoning; $\mathbf{K}$-*ph* formulae, for example, all have a maximum modal depth of 3. Full details of the generation technique and the characteristics of the various classes of formulae can be found in the comparison description [26].

The test methodology here is to ascertain the number of the largest formula of each type that the system is able to solve within 100 seconds of CPU time (>20 indicates that the hardest

[11] We entered both FaCT and DLP in this comparison [32].

[12] Note that a formula is proved by demonstrating the unsatisfiability of its negation.

	FaCT		DLP		KSAT		Kris	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
branch	6	4	19	13	8	8	3	3
d4	>20	8	>20	>20	8	5	8	6
dum	>20	>20	>20	>20	11	>20	15	>20
grz	>20	>20	>20	>20	17	>20	13	>20
lin	>20	>20	>20	>20	>20	3	6	9
path	7	6	>20	>20	4	8	3	11
ph	6	7	7	9	5	5	4	5
poly	>20	>20	>20	>20	13	12	11	>20
t4p	>20	>20	>20	>20	10	18	7	5
KT	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
45	>20	>20	>20	>20	5	5	4	3
branch	6	4	19	12	8	7	3	3
dum	11	>20	>20	>20	7	12	3	14
grz	>20	>20	>20	>20	9	>20	0	5
md	4	5	3	>20	2	4	3	4
path	5	3	16	14	2	5	1	13
ph	6	7	7	>20	4	5	3	3
poly	>20	7	>20	12	1	2	2	2
t4p	4	2	>20	>20	1	1	1	7

Table 5: Results for **K** and **KT**

	FaCT		DLP	
S4	*p*	*n*	*p*	*n*
45	>20	>20	>20	>20
branch	4	4	18	12
grz	2	>20	>20	>20
ipc	5	4	10	>20
md	8	4	3	>20
path	2	1	15	15
ph	5	4	7	>20
s5	>20	2	>20	>20
t4p	5	3	>20	>20

Table 6: Results for **S4**

problem was solved in less than 100 seconds). The results of the **K** and **KT** tests with FaCT, DLP, KSAT [13] and KRIS [14] are summarised in Table 5 while those for **S4** with FaCT and DLP are summarised in Table 6. Neither KSAT nor KRIS can reason with transitive roles, so they cannot be used to perform **S4** satisfiability tests. All times reported are for runs on machines with approximately the speed of a SPARC Ultra 1.

[13] The tests here used the original Lisp implementation of KSAT; a much faster C implementation is now available.
[14] It should be noted that KRIS was not designed to deal efficiently with large satisfiability problems.

In these tests FaCT and DLP outperformed the other systems, with DLP being a clear winner. DLP also outperformed the other systems that took part in the Tableaux'98 comparison [7].

Our second propositional modal logic test suite uses a common method for testing SAT decision procedures [17] that has been adapted for use with propositional modal **K** by Giunchiglia and Sebastiani [23], and further refined by Hustadt and Schmidt [33]. The method uses a random generator to produce formulae, with the characteristics of the formulae being controlled by a number of parameters. Each formula produced is a conjunction of L K-clauses, where a K-clause is a disjunction of K elements, each element being negated with a probability of 0.5. An element is either a modal atom of the form $\forall R.C$, where C is itself a K-clause, or at the maximum modal depth D, a propositional variable chosen from the N propositional variables that appear in the formula. Trivial satisfiability of K-clauses is avoided by choosing a combination of propositional variables from the ${}^{N}\mathbf{C}_{K}$ possibilities.

Hustadt and Schmidt used two sets of formulae, denoted **PS12** and **PS13**, choosing $N = 4$ and $N = 6$ respectively, with $K = 3$ and $D = 1$ in both cases. Because the depth was set to 1, this test suite overemphasises propositional reasoning, and has little interesting modal reasoning. Initial work indicated that this was where the hard problems occur in modal satisfiability; we are reexamining this finding.

The test sets are created by varying L from N to $30N$, giving formulae with a probability of satisfiability varying from $\approx$1 to $\approx$0, and generating 100 formulae for each integer value of L/N. For SAT problems it has been demonstrated that when the other parameters are fixed, the value of L/N determines the "hardness" of formulae.

The median times required to test the satisfiability of the **PS12** and **PS13** formulae using FaCT, DLP, KSAT and KRIS are shown in Figures 5 and 6. In order to keep the the total time required to perform the tests within reasonable bounds, a maximum of 1,000s was allowed for testing a single formula, and testing was terminated at a data point as soon as evidence was gathered that the median solution time for that data point would exceed 1,000 seconds.

It can be seen that in these tests the performance differences between FaCT, DLP and KSAT are much less marked than was the case in the Tableaux'98 tests. This is because, with such a small number of literals, the purely propositional problems at depth 1 can almost always be solved deterministically, and performance is therefore dependent on the efficiency of propositional reasoning at depth 0. Several optimisations in FaCT and DLP, notably caching, are of little use with these formulae as there are no hard modal sub-problems.

The speed difference between FaCT and DLP on these formulae is a bit puzzling, as caching, the main difference between FaCT and DLP, is not effective here (see below). The difference is probably due to low level improvements in DLP, such as optimised data structures.

Although the Tableaux'98 and random test suites show how our optimisations perform on propositional modal logics, neither is very good for our purposes. In particular, the collection of random formulae has a modal depth of 1 and most of the computational difficulties have to do with the initial non-modal component. When using the algorithm for subsumption testing with a realistic knowledge base we expect to encounter hard problems where the hardness comes from the number of successors that have to be considered and their interaction with the non-modal component. The Tableaux'98 formulae have this form, but there are too few hard collections there to validate our optimisations, and the regular structure of the formulae tends to exaggerate the utility of the caching optimisation, particularly for satisfiable (non-

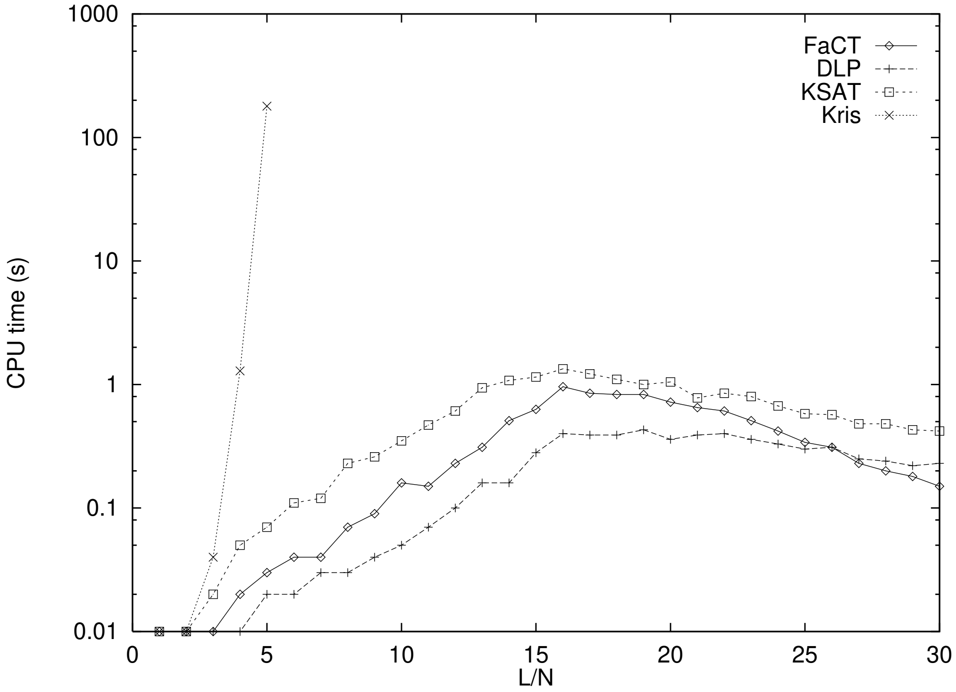

Figure 5: Median solution times for **PS12** formulae

provable) formulae.

One test that we have been able to do with an expressive knowledge base is to take the GALEN knowledge base [42] and construct a version of it that is acceptable to FaCT, DLP, KRIS and CRACK [15]. The GALEN knowledge base is a high level ontology that has been designed to form the foundation of a large concept model representing medical terminology. It has been created using the specially developed GRAIL description logic [41] that supports a primitive role hierarchy, transitive roles and concept inclusion axioms. GRAIL has a limited terminological language—only conjunction and existential role concepts are supported—and an unusual syntax that restricts the way concept expressions can be formed.

The test knowledge base was constructed by first translating the GRAIL syntax of the GALEN knowledge base into the standard syntax used by most implemented description logics [3]. Concept inclusion axioms were then eliminated using a pre-processing technique called *absorption* [29], which can convert some forms of inclusion axiom into augmented concept definitions while still retaining their meaning—an important effect of GRAIL's restricted syntax is that *all* concept inclusion axioms can be eliminated in this way.[16] Finally, all role axioms were discarded. This last step makes the knowledge base acceptable to a larger number of implemented description logics (including KRIS and CRACK), and has relatively little impact on the "hardness" of subsumption testing, which derives primarily from the large number of highly disjunctive concepts generated by absorption [29].

[15] Test results for CRACK are due to Enrico Franconi [18].

[16] We have been unable to find a realistic knowledge base containing non-absorbable inclusion axioms. This is not surprising as prior to FaCT no implemented description logic could deal with such a knowledge base.

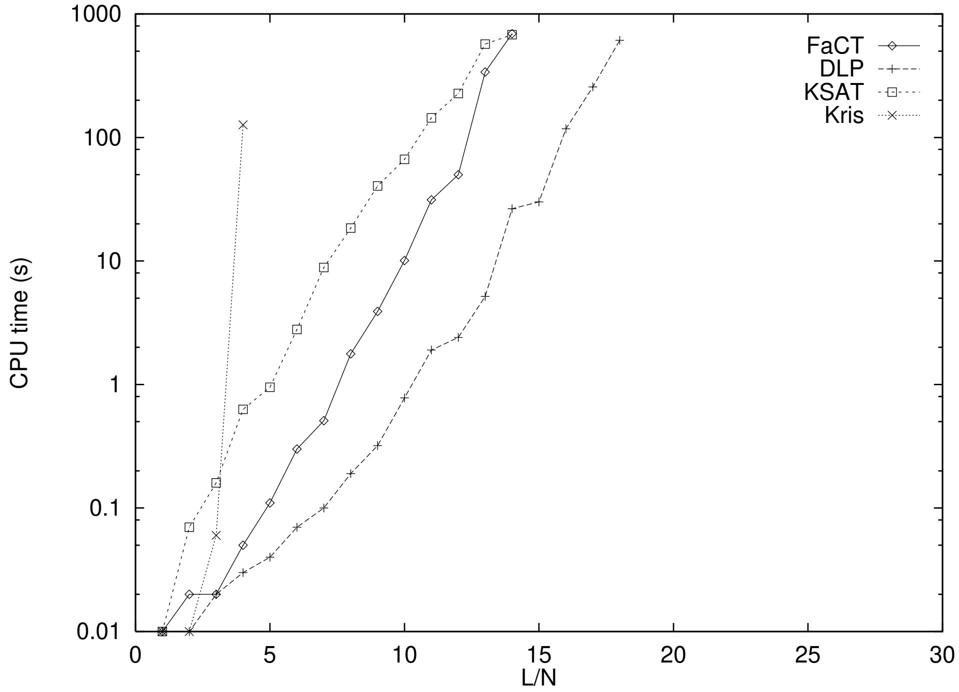

Figure 6: Median solution times for **PS13** formulae

	FaCT	DLP	KRIS	CRACK
Load	6.03	—	135.90	—
Pre-process	0.85	—	—	—
Classify	204.03	—	$\gg$400,000	$\gg$10,000
Total CPU time (s)	210.91	69.56	$\gg$400,000	$\gg$10,000

Table 7: Classification times for GALEN knowledge base

The resulting knowledge base contains 2,719 named concepts and 413 roles. The results of the tests using this knowledge base are summarised in Table 7. Although the structure of the concept hierarchy turns out to be quite simple,[17] it is still necessary to perform tens of thousands of subsumption tests in order to compute the partial ordering, and some of these tests prove to be extremely hard for less optimised subsumption reasoners: neither KRIS nor CRACK was able to classify the knowledge base as they got stuck on single subsumption tests whose solution required more CPU time than was allowed for the whole test. In contrast, FaCT classified the knowledge base in 211 seconds while DLP did so in 70 seconds.

We have also tested FaCT and DLP on slightly modified versions of several other existing knowledge bases. In this case the test was broadened to include NeoClassic, a reimplementation of one of the older description logic systems for which these knowledge bases were

[17] The concept hierarchy closely resembles a tree, with less than 15% of concepts having more than one parent. The average concept has 2 direct sub-concepts, and the maximum depth of the hierarchy is 14.

Knowledge base	Concepts	FaCT	DLP	KRIS	CRACK	NeoClassic
ckb-roles	79	0.19	0.27	0.68	1.19	0.42
datamont-roles	120	0.42	0.36	0.89	1.18	0.65
espr-roles	142	0.33	0.13	0.58	0.00	0.63
fss-roles	132	0.66	0.64	1.16	0.37	0.78
wines	267	4.71	2.05	2.99	2.37	2.77
wisber-roles	140	0.48	0.78	1.03	1.63	1.03

Table 8: Classification times for other knowledge bases (CPU seconds)

developed. More information on these tests can be found in the Systems Comparison section of the *Proceedings of the 1998 International Workshop on Description Logics* [31].

The results of these tests, given in Table 8 show that FaCT and DLP perform very well compared to other systems, even those, like NeoClassic, designed to work very quickly with simple constructs. However, the problem with these knowledge bases is that they are too small or too simple to show off the optimisations in FaCT and DLP. They can only serve to show that there is no significant overhead in using the approach employed in FaCT and DLP.

5 Comparing Optimisations

The comparison with other systems indicates that the suite of optimisations in FaCT and DLP is effective, taken as a group, on several kinds of formulae and knowledge bases. However, it does not show which of the optimisations are most effective. To answer this question, recent versions of DLP have had compile-time configuration options included that can be used to turn on and off or vary the above optimisations. There are too many possible configurations of the optimisations to test them all, but we have run DLP in various configurations on the above test suites.

We chose to test various heuristic combinations with all the other optimisations enabled, and then to test the same heuristic options, at least as far as possible, with each of the optimisations turned off one by one. This could have resulted in some slightly misleading results if two optimisations had similar benefits, as they would both seem to be ineffective, but this does not appear to have been the case in our tests.

The heuristic combinations that were tested are:

Oldest-random: Select a disjunction at random from the set of oldest disjunctions and use the JW heuristic to select a disjunct in it. For syntactic branching this reduces to selecting an oldest disjunction at random.

Oldest-JW: Use the JW heuristic to select a disjunct from within the set of oldest disjunctions.

JW: Use the JW heuristic to select a disjunct from within the entire set of disjunctions.

Random: Select a disjunct at random. (Actually just select the first disjunct that has no value from the first disjunction.)

The optimisations that were removed are:

Optimisation Removed	Heuristics Used			
	Oldest-random	Oldest-JW	JW	Random
NONE	967	915	936	874
Caching	882	826	851	671
Backjumping	880	847	877	795
Semantic Branching	849	—	—	851
BCP	932	873	879	839
Normalisation	911	913	931	781

Table 9: Total Tableaux'98 problems solved

No caching: Turn off caching.

No backjumping: Turn off backjumping.

No semantic branching: Use syntactic instead of semantic branching. This only allows two heuristic variants, random selection of disjunctions and random selection among the oldest disjunctions.

No BCP: Turn off boolean constraint propagation.

No normalisation: Turn off normalisation. (This does not turn off early detection of clashes between formulae that are syntactically identical.)

Since the JW heuristic cannot be used with syntactic branching, we ended up with 22 configurations. We ran each of these configurations over the three test suites described in the previous section. Presenting the amount of detail given in the previous section for each of these configurations would result in too much information, so we have condensed the results.

For the Tableaux'98 test suite we present the total number of problems solved (within 100s of CPU time) by various configurations, and for provable **S4** formulae, the number of problems solved in each problem set. The Tableaux'98 test suite contains 1,134 problems in total; the totals solved by the various configurations are given in Table 9.

Note that the problems in each set are expected to be exponentially more difficult for a naive prover, so even a small increase in the number of problems solved is significant. For example, running on a machine that is roughly twice as fast results in only 8 more problems being solved for the fastest configuration; running with a time-limit of 1000 seconds on this faster machine results in only 55 more problems being solved. Bearing this in mind, we can see from the table that each of the optimisations makes a considerable difference. We can also see that the effectiveness of the optimisations varies depending on which heuristic is used, complicating any determination of which optimisation is "best". The heuristics were less effective, but still quite important. However, the best heuristic overall included a "random" pick from the oldest disjunctions. This probably reflects some localisation heuristic (the pick was actually the first disjunct in the list of disjuncts), as well as the relatively high cost of evaluating the JW weighting for each disjunct.

Overall the optimisation whose removal causes the greatest change to the results is the caching optimisation, followed by backjumping and semantic branching.[18] Caching is very

[18]It is especially hard to compare semantic branching to the other optimisations, as syntactic branching does not admit the same collection of heuristics as does semantic branching.

Optimisation Removed	Heuristic Used	*45*	*branch*	*grz*	*ipc*	*md*	*path*	*ph*	*s5*	*t4p*
NONE	Oldest-random	21	18	21	10	3	15	7	21	21
	Oldest-JW	21	21	21	10	3	8	5	21	21
	JW	21	18	21	10	3	9	5	21	21
	Random	21	18	21	21	4	7	7	21	21
Caching	Oldest-random	21	18	21	8	7	8	7	21	21
	Oldest-JW	21	21	21	7	7	5	5	10	21
	JW	21	18	21	7	7	6	5	21	21
	Random	21	18	0	13	4	3	7	21	21
Backjumping	Oldest-random	21	17	21	9	3	3	7	2	21
	Oldest-JW	21	21	21	9	3	3	4	2	18
	JW	21	17	21	9	3	3	4	4	21
	Random	21	17	21	8	3	3	5	4	21
Semantic Branching	Oldest-random	15	18	21	7	3	3	7	4	7
	Random	15	18	21	7	3	3	7	4	7
BCP	Oldest-random	21	15	21	10	3	11	6	21	21
	Oldest-JW	21	21	21	10	3	7	4	15	21
	JW	21	12	21	10	3	8	4	21	21
	Random	21	16	21	21	4	7	6	21	21
Normalisation	Oldest-random	21	18	7	10	3	9	6	21	21
	Oldest-JW	21	21	21	10	3	10	6	21	21
	JW	21	16	21	10	3	9	6	21	21
	Random	21	10	21	21	4	7	7	4	21

Table 10: Provable $\mathbf{S4}$ Tableaux'98 problems solved

effective on this test suite because of the large amount of structure in the problems, which results in the frequent repetition of sub-problems. Boolean constraint propagation and normalisation are less effective.

Table 10 shows how many of the provable $\mathbf{S4}$ formulae were solved within the time limit. From this table we can see that some of the problems were easy for almost all the configurations, and others were hard for almost all the configurations. The most unusual point in the table is the 0. We can find no reason why that configuration of DLP performs so poorly on this one point, even after rerunning the test several times with different reporting. At a guess, some interaction between the problem and the "random" choices is making that version of DLP examine many successor nodes, and backjumping is not helping to reduce the search space.

For some problems the optimisations result in not only a quantitative change in difficulty, but also a qualitative change, from an exponential growth in solution time to an almost-constant or definitely sub-exponential solution time growth. This is illustrated in Figures 7 and 8, which shows the actual solution times for two classes of formulae with various optimisations disabled. In one of these examples the qualitative improvement is due to caching; in the other it is due to semantic branching and backjumping.

Similar testing was performed on the random formulae from Hustadt and Schmidt. Again,

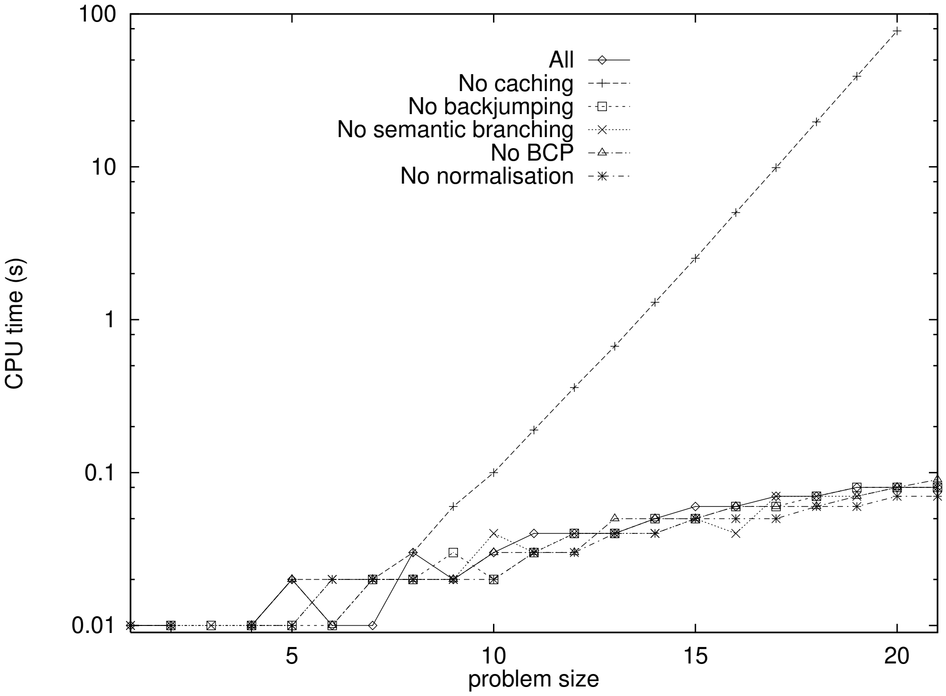

Figure 7: Solution times for Tableaux'98 **K**-*dum-n* formulae

Optimisation Removed	Heuristics Used			
	Oldest-random	Oldest-JW	JW	Random
NONE	0.20	0.85	0.29	0.15
Caching	0.20	0.85	0.29	0.15
Backjumping	1.35	2.06	1.38	1.92
Semantic Branching	13.35	—	—	11.69
BCP	1.26	6.25	1.86	0.60
Normalisation	>87.61	>158.07	>81.34	≫15.10

Table 11: Average median times for **PS12** formulae

there is too much data to present it all. Tables 11 and 12 show the average time for the median-time and 90th-percentile-time formulae across all values of L/N from 1 to 30 for the **PS12** formulae.The data for **PS13** are roughly similar, at least as far as we can tell—one reason for using the **PS12** formulae is that the times can be computed for almost all configurations, whereas many of the **PS13** formulae cannot be solved in 1,000 seconds. The data for removing normalisation is incomplete and only an estimate, as these tests could not be completed due to exhaustion of virtual memory

The results here are somewhat biased against the optimisations, as the implementation methods in DLP were designed to improve maintainability at the expense of the best speed for the optimisations. For example, determining the best JW disjunct requires a separate pass

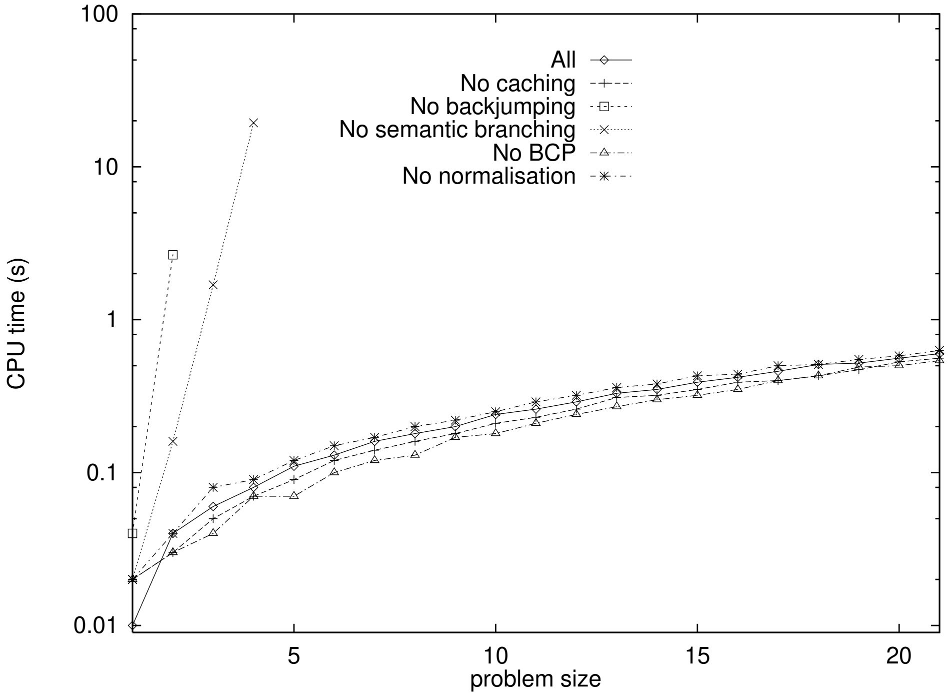

Figure 8: Solution times for Tableaux'98 **S4**-*s5-p* formulae

Optimisation Removed	Heuristics Used			
	Oldest-random	Oldest-JW	JW	Random
NONE	0.40	1.41	0.53	0.32
Caching	0.40	1.42	0.53	0.32
Backjumping	4.61	5.70	4.55	10.21
Semantic Branching	102.54	—	—	90.91
BCP	2.21	9.84	3.17	1.23
Normalisation	>178.46	>236.51	>195.31	≫51.40

Table 12: Average 90th-percentile times for **PS12** formulae

over all the disjuncts and performing boolean constraint propagation requires a separate pass over all the disjunctions. Thus differences of a factor of 2 or 3 against the optimisations are not really significant.

The average median and 90th-percentile data are only a rough indicator as they do not show the variation of solution times as L/N varies. The median and 90th-percentile times for **PS12** for some configurations of DLP are given in Figures 9 and 10. Here only the best variant is given for each removal of an optimisation.

The most effective optimisation by far for this test suite is normalisation. This result is surprising, as direct redundancies and contradictions are supposed to have been removed

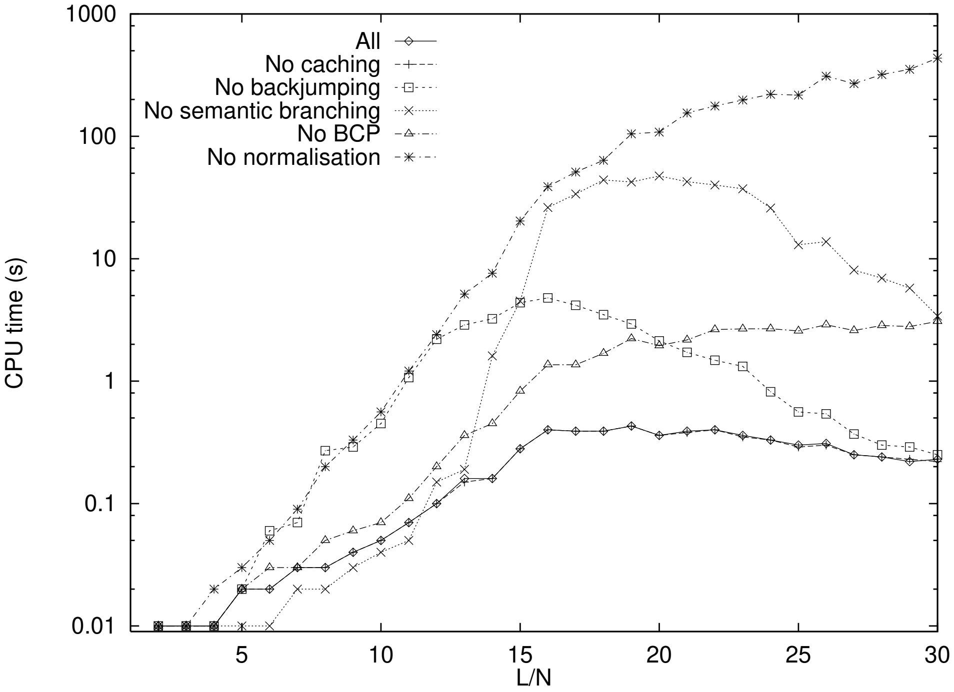

Figure 9: Median solution times for **PS12** formulae

from the data.[19] As this test emphasises propositional reasoning, we expected that semantic branching and backjumping would be the most important optimisations. However, the normalisation process appears to find redundancies and contradictions at a higher level, and the removal of these redundancies makes a dramatic change in the solution time. The next most effective optimisation for this test suite is semantic branching. The probable reason for the effectiveness of these two optimisations is that for large values of L/N most or all propositional solutions must be searched, and thus it is important to ensure that only unique solutions are generated. Syntactic branching obviously fails to do this, but so does non-normalised DLP because it allows the non-modal atoms to look different and thus increase the search space.

Backjumping and boolean constraint propagation, are also effective, but to a lesser extent. Of the two, backjumping is more effective for intermediate values of L/N, where the "harder" problems arise, and boolean constraint propagation is more effective for the larger values of L/N, where the formulae are severely over-constrained, so there is considerable scope for simplification whenever a branching choice is made. The effectiveness of boolean constraint propagation also helps to explain the effectiveness of semantic branching for the over-constrained formulae, as syntactic branching does not allow as much boolean constraint propagation.

Caching is not effective at all in this test suite. This is because, with such a small number of literals in the successor nodes, the purely propositional problems at depth 1 can always be solved deterministically, and performance is therefore dependent on the efficiency of proposi-

[19]Hustadt and Schmidt revised the earlier generation mechanism to remove these problems.

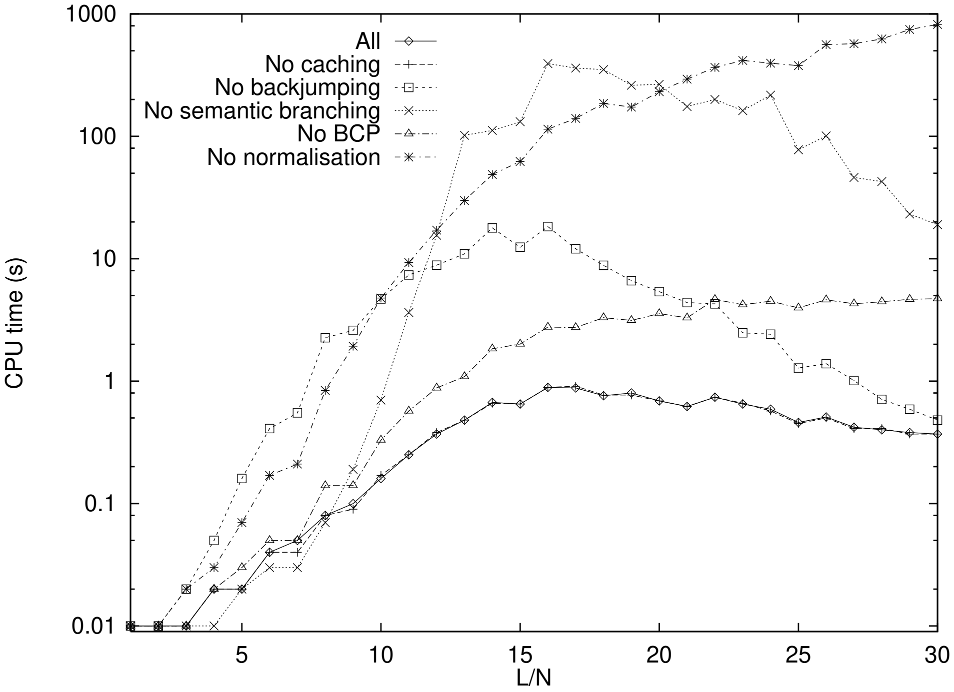

Figure 10: 90th percentile solution times for **PS12** formulae

tional reasoning at the root node. Caching is thus ineffective because there are no hard modal sub-problems to cache.

The versions without heuristics were the fastest in this test suite, showing that the heuristics are not particularly effective. However, computing the information needed for the heuristics is expensive in DLP because of its functional nature. A faster implementation for the heuristics would reduce the difference, but, in this problem set, the structure of the formulae make it unlikely that the configurations with the heuristics would be faster that those without. Because the disjuncts are randomly generated modal sub-formulae there are many different possible disjuncts, so any given disjunct is unlikely to occur in many disjunctions, resulting in little guidance from the JW heuristic. The oldest-first heuristic is ineffective because for formulae in conjunctive normal form every disjunction in the label of the root node has the same "age".

We have also tested the various configurations of DLP on the modified GALEN knowledge base. The times for the various configurations of DLP loading this knowledge base are given in Table 13.

In this test backjumping is by far the most important optimisation. With backjumping turned off DLP was unable to process the knowledge base within 10,000 seconds. The next-most important optimisations are caching and semantic branching. Boolean constraint propagation is even less effective, and normalisation is almost totally ineffective.

The heuristics are not very effective here, with random choice being the fastest. However, this mostly reflects the overhead required to decide which disjunct or disjunction to use and does not mean that the heuristics were actually bad—just that there was too much overhead

Optimisation Removed	Heuristics Used			
	Oldest-random	Oldest-JW	JW	Random
NONE	70	172	153	37
Caching	399	1182	1005	326
Backjumping	>10,000	>10,000	>10,000	>10,000
Semantic Branching	2087	—	—	319
BCP	90	431	616	40
Normalisation	87	207	162	39

Table 13: Classification times for GALEN knowledge base (CPU seconds)

to show the improvement.

We also took some of the hardest subsumption problems from this knowledge base and turned them into satisfiability tests. These tests have proved to be difficult for state-of-the-art propositional modal theorem provers such as KSAT and Hustadt and Schmidt's SPASS based system [33]. In fact, some of these satisfiability tests take over 1,000 seconds for these two theorem provers. We had planned to use these hard subsumption problems to do further evaluation of the optimisations but our initial runs have served to show that there is an unacceptable level of similarity between the different elements of the collection. This is not too surprising, as they all come from the same knowledge base, but it means that little information can be gathered from the problems beyond that gathered from the total time for processing the entire knowledge base.

6 Summary

The collection of optimisations we have described are effective in improving the speed of modal propositional logic reasoners, as shown by the results we have given above. The optimisations can also dramatically improve the speed of subsumption reasoning on the GALEN knowledge base. To our knowledge some of these improvements have not been investigated in the modal propositional reasoning literature. The combination appears to be unique and, moreover, results in a powerful reasoner for the propositional modal logics $\mathbf{K}$, $\mathbf{KT}$, and $\mathbf{S4}$.

The optimisations are not uniformly effective. In particular, semantic branching is extremely effective on constructed hard problems and on random satisfiability problems, but not on the GALEN knowledge base. We plan to perform more experiments to see if semantic branching is indeed ineffective on other realistic knowledge bases. The two other optimisations that are the most effective on the non-random tests are backjumping and caching. These two optimisations make the difference between acceptable and ridiculous performance in many cases. Their absence in previous description logic systems has made them unacceptably slow.

We, along with a colleague, are embarking on a project to create a description logic system for a description logic that includes converse propositional dynamic logic. This project will require more optimisation, as inference in converse propositional dynamic logic is more difficult to efficiently implement than inference in the logics we are currently handling, and will give us further opportunities to investigate the optimisation of satisfiability reasoners. We are also performing more testing of the optimisations we are putting into our provers and we plan to create a test suite that emphasises the modal nature of description logics.

FaCT is available at

`http://www.cs.man.ac.uk/~horrocks`;

the DLP prover is currently under development, but a version is available at

`http://www.bell-labs.com/user/pfps`.

Acknowledgements

We would like to thank the anonymous referees for their valuable comments and suggestions.

References

[1] F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. In Proceedings of IJCAI-91 [40], pages 446–451.

[2] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.-J. Profitlich. An empirical analysis of optimization techniques for terminological representation systems or: Making KRIS get a move on. In B. Nebel, C. Rich, and W. Swartout, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR'92)*, pages 270–281. Morgan-Kaufmann Publishers, San Francisco, CA, 1992. Also available as DFKI RR-93-03.

[3] F. Baader, H.-J. Heinsohn, B. Hollunder, J. Muller, B. Nebel, W. Nutt, and H.-J. Profitlich. Terminological knowledge representation: A proposal for a terminological logic. Technical Memo TM-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1991.

[4] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithms. In *Processing declarative knowledge: International workshop PDK'91*, number 567 in Lecture Notes in Artificial Intelligence, pages 67–86, Berlin, 1991. Springer-Verlag.

[5] F. Baader and B. Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

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

[7] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics — introduction and summary. In de Swart [14], pages 25–26.

[8] R. J. Brachman, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In J. F. Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*, chapter 14, pages 401–456. Morgan-Kaufmann, 1991.

[9] R. J. Brachman, P. G. Selfridge, L. G. Terveen, B. Altman, A. Borgida, F. Halper, T. Kirk, A. Lazar, D. L. McGuinness, and L. A. Renick. Integrated support for data archaeology. *International Journal of Applied and Cooperative Information Systems*, 2(2):159–185, 1993.

[10] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: a preliminary report. In Ellis et al. [15], pages 28–39.

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

[12] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[13] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-pdl. *Information and Computation*, to appear.

[14] H. de Swart, editor. *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence. Springer-Verlag, May 1998.

[15] Gerard Ellis, Robert A. Levinson, Andrew Fall, and Veronica Dahl, editors. *Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium*, 1995.

[16] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18:194–211, 1979.

[17] J. Franco and M. Paull. Probabilistic analysis of the Davis-Putnam procedure for solving the satisfiability problem. *Discrete Applied Mathematics*, 5:77–87, 1983.

[18] E. Franconi. Systems comparison: Crack. In Franconi et al. [19], pages 58–59.

[19] E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors. *Collected Papers from the International Description Logics Workshop (DL'98)*. CEUR, May 1998.

[20] J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, USA, 1995.

[21] J. W. Freeman. Hard random 3-SAT problems and the Davis-Putnam procedure. *Artificial Intelligence*, 81:183–198, 1996.

[22] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal K. In Michael McRobbie and John Slaney, editors, *Proceedings of the Thirteenth International Conference on Automated Deduction (CADE-13)*, number 1104 in Lecture Notes in Artificial Intelligence, pages 583–597. Springer, 1996.

[23] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In L. C. Aiello, J. Doyle, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*, pages 304–314. Morgan Kaufmann Publishers, San Francisco, CA, November 1996.

[24] J. Y. Halpern and Y. Moses. A guide to completeness and complexity for model logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, April 1992.

[25] J. Heinsohn, D. Kudenko, B. Nebel, and H.-J. Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68:367–397, 1994.

[26] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, and S4. Technical report IAM-96-015, University of Bern, Switzerland, October 1996.

[27] B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. In *Proceedings of the 9th European Conference on Artificial Intelligence (ECAI'90)*, pages 348–353. John Wiley & Sons Ltd., 1990.

[28] I. Horrocks. A comparison of two terminological knowledge representation systems. Master's thesis, University of Manchester, 1995.

[29] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[30] I. Horrocks. Using an expressive description logic: FaCT or fiction? In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 636–647. Morgan Kaufmann Publishers, San Francisco, CA, June 1998.

[31] I. Horrocks and P. F. Patel-Schneider. Dl systems comparison. In Franconi et al. [19], pages 55–57.

[32] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In de Swart [14], pages 27–30.

[33] U. Hustadt and R. A. Schmidt. On evaluating decision procedures for modal logic. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, volume 1, pages 202–207, 1997.

[34] R. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[35] B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2):235–249, 1990.

[36] F. Oppacher and E. Suen. HARP: A tableau-based theorem prover. *Journal of Automated Reasoning*, 4:69–100, 1988.

[37] P. F. Patel-Schneider. DLP system description. In Franconi et al. [19], pages 87–89.

[38] V. R. Pratt. A practical decision method for propositional dynamic logic. In *Proceedings of the Tenth ACM Symposium on Theory of Computing (STOC-78)*, pages 326–337, 1978.

[39] Vaughan R. Pratt. Models of program logics. In *Proceedings of the 20th Annual Symposium on the Foundations of Computer Science*, pages 115–122. IEEE Computer Society Press, 1979.

[40] *Proceedings of the 12th International Joint Conference on Artificial Intelligence, (IJCAI-91)*, 1991.

[41] A. Rector, S. Bechhofer, C. A. Goble, I. Horrocks, W. A. Nowlan, and W. D. Solomon. The GRAIL concept modelling language for medical terminology. *Artificial Intelligence in Medicine*, 9:139–171, 1997.

[42] A. L. Rector, W A Nowlan, and A Glowinski. Goals for concept representation in the GALEN project. In *Proceedings of the 17th Annual Symposium on Computer Applications in Medical Care (SCAMC'93)*, pages 414–418, Washington DC, USA, 1993.

[43] U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *20. Deutsche Jahrestagung für Künstliche Intelligenz*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer Verlag, 1996.

[44] K. Schild. A correspondence theory for terminological logics: Preliminary report. In Proceedings of IJCAI-91 [40], pages 466–471.

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

[46] P.-H. Speel, F. van Raalte, P. E. van der Vet, and N. J. I. Mars. Runtime and memory usage performance of description logics. In Ellis et al. [15], pages 13–27.

[47] A. Tarski. *Logic, Semantics, Mathemetics: Papers from 1923 to 1938*. Oxford University Press, 1956.

[48] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and System Sciences*, 32:183–221, 1986.