Using an Expressive Description Logic: FaCT or Fiction?

Ian R. Horrocks
Medical Informatics Group
Department of Computer Science
University of Manchester
Manchester M13 9PL, UK
horrocks@cs.man.ac.uk www.cs.man.ac.uk/~horrocks

Abstract

Description Logics form a family of formalisms closely related to semantic networks but with the distinguishing characteristic that the semantics of the concept description language is formally defined, so that the subsumption relationship between two concept descriptions can be computed by a suitable algorithm. Description Logics have proved useful in a range of applications but their wider acceptance has been hindered by their limited expressiveness and the intractability of their subsumption algorithms. This paper addresses both these issues by describing a sound and complete tableaux subsumption testing algorithm for a relatively expressive Description Logic which, in spite of the logic's worst case complexity, has been shown to perform well in realistic applications.

1 INTRODUCTION

Description Logics (DLs) form a family of formalisms which have grown out of knowledge representation techniques using frames and semantic networks; their distinguishing characteristic is a formally defined semantics which enables the subsumption (kind-of) relationship to be computed by a suitable algorithm (Woods and Schmolze 1992). DL based knowledge representation systems have proved useful in a range of applications (Berman et al. 1994; Guha and Lenat 1994; Goble et al. 1996; Levy and Rousset 1996; Küssner 1997), but their wider acceptance has been hindered by their limited expressiveness (Doyle and Patil 1991; MacGregor 1991) and the intractability of their subsumption algorithms (Heinsohn et al. 1994; Speel et al. 1995).[1] This paper addresses both these issues by describing a tableaux subsumption testing algorithm for a relatively expressive DL which, in spite of the logic's worst case complexity, has been shown to perform well in realistic applications.

A particularly promising application domain for DLs is in the growth area of ontological engineering—the design, construction and maintenance of large conceptual schemas or ontologies (Mays et al. 1996; Horrocks et al. 1996; Rector and Horrocks 1997). An example of such an application is the European GALEN project, which aims to promote the sharing and re-use of medical data by building a large medical terminology ontology which can be used by application designers as a flexible and extensible classification schema (Rector et al. 1993). However design studies at the outset of the project identified expressive requirements which were satisfied by few if any implemented DL systems (Nowlan 1993), in particular the ability to reason about transitive part-whole, causal and compositional relations (called *roles* in DLs) (Rector et al. 1997). The importance of reasoning with transitive roles has long been recognised (Hors 1994; Padgham and Lambrix 1994; Artale et al. 1996), and has been identified as a requirement in other application domains, particularly those concerned with complex physically composed objects, e.g., engineering (Sattler 1995).

The work presented here was motivated by the desire to provide a sound, complete and empirically tractable algorithm for a DL with the expressive power required by these kinds of application. The logic developed for this purpose was $\mathcal{ALCH}_{R^+}$, an extension of the well known $\mathcal{ALC}$ DL (Schmidt-Schauß and Smolka 1991). $\mathcal{ALC}$ supports concept descriptions using the standard logical connectives, plus existential and universal role restrictions, but only using simple primitive roles. $\mathcal{ALCH}_{R^+}$ augments $\mathcal{ALC}$ with transitively closed primitive roles and role inclusion axioms, the combination of which enables a hierarchy of transitive and non-transitive roles to be defined—a fundamental requirement of the GALEN project—and allows reasoning with respect to general terminologies (see Section 5), a feature which is also required by some applications (Levy

[1] A desire/requirement for sound and complete reasoning is assumed, but see (Borgida 1992) for a discussion of this issue.

and Rousset 1996). A tableaux satisfiability testing algorithm for $\mathcal{ALCH}_{R^+}$ will be presented, along with a proof of its soundness and completeness, and an extension to the algorithm which supports reasoning with attributes (functional/feature roles) will also be described. A highly optimised implementation of this algorithm forms the basis for a terminological classifier, FaCT, which has been developed to demonstrate the feasibility of using the algorithm for subsumption reasoning in realistic applications.

2 TRANSITIVE ROLES

Extensions to $\mathcal{ALC}$ which support some form of transitive roles include $\mathcal{CIQ}$ (Giacomo and Lenzerini 1996), $\mathcal{TSL}$ (Schild 1991), $\mathcal{ALC}_+$ (Baader 1990), $\mathcal{ALC}_{R^+}$ and $\mathcal{ALC}_\oplus$ (Sattler 1996). Of these, $\mathcal{CIQ}$, $\mathcal{TSL}$ and $\mathcal{ALC}_+$ all support role expressions with transitive or transitive reflexive operators, and from correspondence to propositional dynamic logics their satisfiability problems are known to be EXPTIME-complete (Schild 1991). The $\mathcal{ALC}_{R^+}$ and $\mathcal{ALC}_\oplus$ DLs support transitive roles without providing a general transitive closure operator, and were investigated in the hope that a more restricted form of transitive role might lead to a satisfiability problem in a lower complexity class (Sattler 1996).

$\mathcal{ALC}_{R^+}$ augments $\mathcal{ALC}$ with transitively closed primitive roles: an $\mathcal{ALC}_{R^+}$ terminology may include axioms of the form $R \in \mathbf{R}_+$, where R is a role name and $\mathbf{R}_+$ is the set of transitive roles names in the terminology. In (Sattler 1996) an algorithm for deciding the satisfiability of $\mathcal{ALC}_{R^+}$ concept expressions is presented along with a proof of its soundness and completeness. It is also demonstrated that the complexity of the $\mathcal{ALC}_{R^+}$ satisfiability problem is PSPACE-complete, the same as for $\mathcal{ALC}$ (Donini et al. 1995). $\mathcal{ALC}_\oplus$ extends $\mathcal{ALC}_{R^+}$ by associating each non-transitive role R with its transitive orbit. The transitive orbit of a role R, denoted $R^\oplus$, is a transitive role which subsumes R, and could be defined by the axioms $R^\oplus \in \mathbf{R}_+$ and $R \sqsubseteq R^\oplus$. The interpretation of $R^\oplus$ is therefore a superset of the interpretation of the transitive closure of R, i.e., $(R^\oplus)^\mathcal{I} \supseteq (R^+)^\mathcal{I}$. Unfortunately, the complexity of the $\mathcal{ALC}_\oplus$ satisfiability problem is also shown to be EXPTIME-complete.

$\mathcal{ALCH}_{R^+}$ generalises $\mathcal{ALC}_\oplus$ by supporting transitively closed primitive roles and arbitrary role inclusion axioms of the form $R \sqsubseteq S$. As it is more general than $\mathcal{ALC}_\oplus$, but still less expressive than DLs such as $\mathcal{ALC}_+$ which support the transitive closure role forming operator, the $\mathcal{ALCH}_{R^+}$ satisfiability problem is clearly also EXPTIME-complete. However, the tableaux satisfiability testing algorithm for $\mathcal{ALCH}_{R^+}$ is much simpler than for $\mathcal{ALC}_+$:

1. It is simpler to detect cycles in the tableaux construction which could lead to non-termination. Cycle detection (*blocking*) involves comparing sets of concepts, and this is complicated in $\mathcal{ALC}_+$ by the need to deal with concepts containing role expressions. It can be shown, for example, that identifying equivalent role expressions is itself a PSPACE-complete problem (Baader 1990).
2. It is simpler to deal with cycles once they have been detected, because in the $\mathcal{ALCH}_{R^+}$ algorithm all cycles lead to a valid cyclical model. In the $\mathcal{ALC}_+$ algorithm, on the other hand, it is necessary to differentiate between cycles which lead to a valid cyclical model (*good cycles*) and those which do not (*bad cycles*).

$\mathcal{ALCH}_{R^+}$ is sufficiently expressive to be useful in a range of applications, but the simplicity of its satisfiability testing algorithm means that it is easy to implement and amenable to a wide range of optimisation techniques.

3 THE $\mathcal{ALCH}_{R^+}$ DESCRIPTION LOGIC

In this section a tableaux algorithm for testing the satisfiability of $\mathcal{ALCH}_{R^+}$ concept expressions will be described and a proof of its soundness and completeness presented. The algorithm and proof are extensions of those described for $\mathcal{ALC}_{R^+}$ (Sattler 1996).

3.1 SYNTAX AND SEMANTICS

$\mathcal{ALCH}_{R^+}$ is the DL obtained by augmenting $\mathcal{ALC}$ with transitively closed primitive roles and primitive role introduction axioms. An $\mathcal{ALCH}_{R^+}$ terminology consists of a finite set of axioms of the form $C \sqsubseteq D \mid C \doteq D \mid R \sqsubseteq S \mid R \in \mathbf{R}_+$, where C and D are concept expressions, R and S are role names and $\mathbf{R}_+$ is the set of transitive role names. $\mathcal{ALCH}_{R^+}$ concept expressions are of the form $\mathsf{CN} \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C$, where CN is a concept name, C and D are concept expressions and R is a role name.

A standard Tarski style model theoretic semantics is used to interpret concept expressions and to justify subsumption inferences (Baader et al. 1991). 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 name CN to a subset of $\Delta^\mathcal{I}$ and each role to to a set valued function (or equivalently a binary relation): $R^\mathcal{I} : \Delta^\mathcal{I} \longrightarrow 2^{\Delta^\mathcal{I}}$ ($R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$). The semantics of an $\mathcal{ALCH}_{R^+}$ concept expression is derived from the semantics of its components as shown in Figure 1. The semantics of $\mathcal{ALCH}_{R^+}$ axioms is given in Figure 2.

To simplify the description of the algorithm, it will be as-

Syntax	Semantics
CN	$\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$\neg C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\exists R.C$	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
$\forall R.C$	$\{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$

Figure 1: Semantics of $\mathcal{ALCH}_{R^+}$ Concept Expressions

Syntax	Semantics
$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
$C \doteq D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
$R \in \mathbf{R}_+$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$

Figure 2: Semantics of $\mathcal{ALCH}_{R^+}$ Axioms

sumed that $\mathbf{R}_+$ and the $\sqsubseteq$ relation have been defined by an $\mathcal{ALCH}_{R^+}$ terminology $\mathcal{T}$, so that $\mathbf{R}_+ = \{R \mid R \in \mathbf{R}_+$ is an axiom in $\mathcal{T}\}$, and for two roles R and S, $R \sqsubseteq S$ iff $R \sqsubseteq S$ is an axiom in $\mathcal{T}$ or there is a role R' such that $R \sqsubseteq R'$ is an axiom in $\mathcal{T}$ and $R' \sqsubseteq S$. Without loss of generality, it will also be assumed that the concept expression is in negation normal form, so that negations are applied only to concept names, and that the terminology does not contain any concept axioms (i.e., axioms of the form $C \sqsubseteq D$ or $C \doteq D$), so that all concept names are atomic primitives.[2] Arbitrary concept expressions can be transformed into negation normal form using a combination of DeMorgan's laws and the identities $\neg\exists R.C = \forall R.\neg C$ and $\neg\forall R.C = \exists R.\neg C$. How the algorithm can be used to test satisfiability w.r.t. a general terminology will be described in Section 5.

Like other tableaux algorithms, the $\mathcal{ALCH}_{R^+}$ algorithm tries to prove the satisfiability of a concept expression D by demonstrating a model of D—an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ such that $D^{\mathcal{I}} \neq \emptyset$. The model is represented by a tree whose nodes correspond to individuals in the model, each node being labelled with a set of $\mathcal{ALCH}_{R^+}$-concepts. When testing the satisfiability of an $\mathcal{ALCH}_{R^+}$-concept D, these sets are restricted to subsets of $sub(D)$, where $sub(D)$ is the closure of the subexpressions of D defined as follows:

1. if D is an atomic primitive concept or its negation, then $sub(D) = \{D\}$
2. if D is of the form $\exists R.C$ or $\forall R.C$, then $sub(D) = \{D\} \cup sub(C)$
3. if D is of the form $C_1 \sqcap C_2$ or $C_1 \sqcup C_2$, then $sub(D) = \{D\} \cup sub(C_1) \cup sub(C_2)$

[2] An atomic primitive is a concept name CN for which there is no definition in $\mathcal{T}$: all that is known about CN is that $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$.

The soundness and completeness of the algorithm will be proved by showing that a concept has a model iff it has a *tableau*, and that the algorithm constructs a tableau for a concept iff one exists.

Definition 1 If D is an $\mathcal{ALCH}_{R^+}$-concept and $\mathbf{R}_D$ is the set of role names occurring in D, a tableau T for D is defined to be a triple $(\mathbf{S}, \mathcal{L}, \mathcal{E})$ such that: $\mathbf{S}$ is a set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{sub(D)}$ maps each individual to a set of concept expressions which is a subset of $sub(D)$, $\mathcal{E} : \mathbf{R}_D \to 2^{\mathbf{S}\times\mathbf{S}}$ maps each role name occurring in D to a set of pairs of individuals, and there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$. For all $s \in \mathbf{S}$ it holds that:

1. $\bot \notin \mathcal{L}(s)$, and if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$
2. if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$
3. if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$
4. if $\forall R.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$
5. if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathbf{S}$ s.t. $\langle s, t\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$
6. if $\forall R.C \in \mathcal{L}(s)$, $\langle s, t\rangle \in \mathcal{E}(S)$, $S \in \mathbf{R}_+$ and $S \sqsubseteq R$, then $\forall S.C \in \mathcal{L}(t)$
7. if $R \sqsubseteq S$ then $\mathcal{E}(R) \subseteq \mathcal{E}(S)$

Lemma 1 *An $\mathcal{ALCH}_{R^+}$-concept D is satisfiable iff there exists a tableau for D.*

Proof: For the *if* direction, if $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ is a tableau for D, a model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of D can be defined as:

$$
\begin{array}{rcl}
\Delta^{\mathcal{I}} & = & \mathbf{S} \\
\mathsf{CN}^{\mathcal{I}} & = & \{s \mid \mathsf{CN} \in \mathcal{L}(s)\} \\
 & & \text{for all concept names } \mathsf{CN} \text{ in } sub(D) \\
R^{\mathcal{I}} & = & \left\{ \begin{array}{ll} \mathcal{E}(R)^+ & \text{if } R \in \mathbf{R}_+ \\ \mathcal{E}(R) \cup \bigcup\limits_{S \sqsubset R} S^{\mathcal{I}} & \text{otherwise} \end{array} \right.
\end{array}
$$

where $\mathcal{E}(R)^+$ denotes the transitive closure of $\mathcal{E}(R)$.

By induction on the structure of concepts it can be shown that $\mathcal{I}$ is well defined and that $D^{\mathcal{I}} \neq \emptyset$. For concepts of the form $\neg C$, $C_1 \sqcap C_2$, $C_1 \sqcup C_2$ and $\exists R.C$, the correctness of their interpretations follows directly from Definition 1 and the semantics of $\mathcal{ALCH}_{R^+}$ concept expressions given in Figure 1 above:

1. For concepts of the form $\neg C$, if $\neg C \in \mathcal{L}(s)$, then $C \notin \mathcal{L}(s)$, so $s \in \Delta^{\mathcal{I}} - C^{\mathcal{I}}$ and $\neg C$ is correctly interpreted.
2. For concepts of the form $C_1 \sqcap C_2$, if $C_1 \sqcap C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ and $C_2 \in \mathcal{L}(s)$, so $s \in C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ and $C_1 \sqcap C_2$ is correctly interpreted.
3. For concepts of the form $C_1 \sqcup C_2$, if $C_1 \sqcup C_2 \in \mathcal{L}(s)$, then $C_1 \in \mathcal{L}(s)$ or $C_2 \in \mathcal{L}(s)$, so $s \in C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$ and $C_1 \sqcup C_2$ is correctly interpreted.
4. For concepts of the form $\exists R.C$, if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in S$ such that $\langle s,t\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$, so $s \in \{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$ and $\exists R.C$ is correctly interpreted.

For concepts of the form $\forall R.C$, the correctness of their interpretations follows from Definition 1, and the semantics of $\mathcal{ALCH}_{R^+}$ concept expressions and axioms:

1. if $\forall R.C \in \mathcal{L}(s)$ and $\langle s,t\rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$
2. if $\forall R.C \in \mathcal{L}(s)$, $\langle s,t\rangle \notin \mathcal{E}(R)$ and $\langle s,t\rangle \in R^{\mathcal{I}}$, then there exists a path $\langle s,u_1\rangle,\ldots,\langle u_n,t\rangle$ s.t. $n \geqslant 1$, $\{\langle s,u_1\rangle,\ldots,\langle u_n,t\rangle\} \subseteq \mathcal{E}(S)$, $S \in \mathbf{R}_+$ and $S \sqsubseteq R$. From property 6 of Definition 1, $\forall S.C \in \mathcal{L}(u_i)$ for all $1 \leqslant i \leqslant n$, and from property 4 of Definition 1, $C \in \mathcal{L}(t)$

so $s \in \{d \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$ and $\forall R.C$ is correctly interpreted.

Finally, from Definition 1, there is some individual $s \in \mathbf{S}$ such that $D \in \mathcal{L}(s)$, so $s \in D^{\mathcal{I}}$ and $D^{\mathcal{I}} \neq \emptyset$.

For the converse, if $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is a model of D, then a tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ for D can be defined as:

$$\begin{aligned} S &= \Delta^{\mathcal{I}} \\ \mathcal{E}(R) &= R^{\mathcal{I}} \\ \mathcal{L}(s) &= \{C \in \mathit{sub}(D) \mid s \in C^{\mathcal{I}}\} \end{aligned}$$

It only remains to demonstrate that T is a tableau for D:

1. T satisfies properties 1–5 in Definition 1 as a direct consequence of the semantics of the $\neg C$, $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, $\forall R.C$ and $\exists R.C$ concept expressions.
2. If $d \in (\forall R.C)^{\mathcal{I}}$, $\langle d,e\rangle \in S^{\mathcal{I}}$, $S \in \mathbf{R}_+$ and $S \sqsubseteq R$, then $e \in (\forall S.C)^{\mathcal{I}}$ unless there is some f such that $\langle e,f\rangle \in S^{\mathcal{I}}$ and $f \notin C^{\mathcal{I}}$. However, if $\langle d,e\rangle \in S^{\mathcal{I}}$, $\langle e,f\rangle \in S^{\mathcal{I}}$, $S \in \mathbf{R}_+$ and $S \sqsubseteq R$, then from the semantics of role axioms given in Figure 2, $S \in \mathbf{R}_+ \rightarrow \langle d,f\rangle \in S^{\mathcal{I}}$, $S \sqsubseteq R \rightarrow \langle d,f\rangle \in R^{\mathcal{I}}$ and $d \notin (\forall R.C)^{\mathcal{I}}$. T therefore satisfies property 6 in Definition 1.
3. T satisfies property 7 in Definition 1 as a direct consequence of the semantics of role inclusion axioms given in Figure 2. ∎

3.2 CONSTRUCTING AN $\mathcal{ALCH}_{R^+}$ TABLEAU

The algorithm builds a tree where each node x of the tree is labelled with a set $\mathcal{L}(x) \subseteq \mathit{sub}(D)$ and may, in addition, be marked *satisfiable*. The tree is expanded either by extending $\mathcal{L}(x)$ for some leaf node x or by adding new leaf nodes. For a node x, $\mathcal{L}(x)$ is said to contain a *clash* if $\perp \subseteq \mathcal{L}(x)$ or, for some concept C, $\{C, \neg C\} \subseteq \mathcal{L}(x)$. $\mathcal{L}(x)$ is called a *pre-tableau* if it is clash-free and contains no unexpanded conjunction or disjunction concepts. Note that $\emptyset$ is a pre-tableau.

Edges of the tree are either labelled $\sqcup$ or labelled R for some role name R occurring in $\mathit{sub}(D)$. Edges labelled $\sqcup$ are added when expanding $C_1 \sqcup C_2$ concepts in $\mathcal{L}(x)$, and are the mechanism whereby the algorithm searches possible alternative expansions. Edges labelled with a role name R are added when expanding $\exists R.C$ terms in $\mathcal{L}(x)$, and correspond to relationships between pairs of individuals.

A node y is called an *R-successor* of a node x if there is an edge $\langle x,y\rangle$ labelled R; y is called a *$\sqcup$-successor* of x if there is a path, consisting of $\sqcup$-labelled edges, from x to y. A node x is an *ancestor* of a node y if there is a path from x to y regardless of the labelling of the edges. Note that both the $\sqcup$-successor and ancestor relations are reflexive as nodes are connected to themselves by the empty path.

The algorithm initialises a tree $\mathbf{T}$ to contain a single node x_0, called the *root* node, with $\mathcal{L}(x_0) = \{D\}$. $\mathbf{T}$ is then expanded by repeatedly applying the rules from Figure 3 until either the root node is marked *satisfiable* or none of the rules is applicable. Note that the second condition in each rule prevents multiple applications of the rule to the same concept expression(s), while blocking is performed by part b of the $\exists$-rule.

If the algorithm constructs a tree in which the root node is marked *satisfiable*, then it returns *satisfiable*; from this tree a tableau for D can trivially be constructed. If the algorithm terminates without marking the root node *satisfiable*, then it returns *unsatisfiable*.

3.3 SOUNDNESS AND COMPLETENESS

From Lemma 1, the soundness and completeness of the algorithm can be demonstrated by proving that, for an $\mathcal{ALCH}_{R^+}$-concept D, it always terminates and that it returns *satisfiable* iff there exists a tableau for D.

Lemma 2 *For each $\mathcal{ALCH}_{R^+}$-concept D, the tableau construction terminates.*

$\sqcap$-rule:	if 1.	x is a leaf of $\mathbf{T}$, $\mathcal{L}(x)$ is clash-free, $C_1 \sqcap C_2 \in \mathcal{L}(x)$
	2.	$\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
	then	$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if 1.	x is a leaf of $\mathbf{T}$, $\mathcal{L}(x)$ is clash-free, $C_1 \sqcup C_2 \in \mathcal{L}(x)$
	2.	$\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	create two $\sqcup$-successors y, z of x with: $\mathcal{L}(y) = \mathcal{L}(x) \cup \{C_1\}$, $\mathcal{L}(z) = \mathcal{L}(x) \cup \{C_2\}$
$\exists$-rule:	if 1.	$\mathcal{L}(x)$ is a pre-tableau, there is a concept of the form $\exists R.C$ in $\mathcal{L}(x)$
	2.	x is a leaf of $\mathbf{T}$
	then	for each $\exists R.C \in \mathcal{L}(x)$ do:
		a. $\ell_{Rx} := \{C\} \cup \{D \mid \forall S.D \in \mathcal{L}(x)$ and $R \sqsubseteq S\}$ $\cup \{\forall S.D \mid \forall P.D \in \mathcal{L}(x),\ S \in \mathbf{R}_+,\ S \sqsubseteq P$ and $R \sqsubseteq S\}$
		b. if for some ancestor w of x, $\ell_{Rx} \subseteq \mathcal{L}(w)$ then create an R-successor y of x with $\mathcal{L}(y) = \emptyset$
		c. otherwise create an R-successor y of x with $\mathcal{L}(y) = \ell_{Rx}$
SAT-rule:	if 1.	x is a node of $\mathbf{T}$, and either:
		a. $\mathcal{L}(x)$ is a pre-tableau containing no concepts of the form $\exists R.C$
		b. $\mathcal{L}(x)$ is a pre-tableau which has successors, and all successors of x are marked *satisfiable*
		c. $\mathcal{L}(x)$ is not a pre-tableau and some $\sqcup$-successor of x is marked *satisfiable*
	2.	x is not marked *satisfiable*
	then	mark x *satisfiable*

Figure 3: Tableaux Expansion Rules for $\mathcal{ALCH}_{R+}$

Proof: Let $m = |sub(D)|$. As nodes are labelled with subsets of $sub(D)$, $|\mathcal{L}(x)| \leq m$ for all nodes x. For any node x the $\sqcap$-rule can therefore be applied at most m times. The size of any sub-trees is also limited by m: the $\sqcup$-rule can also be applied at most m times along a $\sqcup$-labelled path and the $\exists$-rule can be applied at most 2^m times along any path before there must be some ancestor y s.t. $\ell_{Rx} \subseteq \mathcal{L}(y)$ for any R.

Lemma 3 *For an $\mathcal{ALCH}_{R+}$-concept D, there exists a tableau for D iff the tableau construction returns* satisfiable.

Proof: For the *if* direction (the algorithm returns *satisfiable*), let $\mathbf{T}$ be the tree constructed by the tableaux algorithm for D. A tableau $T = (\mathbf{S}, \mathcal{L}, \mathcal{E})$ can be defined with:

$\mathbf{S} = \{x \mid x$ is a node in $\mathbf{T}$, x is marked *satisfiable* and $\mathcal{L}(x)$ is a non-empty pre-tableau.$\}$

$\mathcal{E}(R) = \{\langle x, y\rangle \in \mathbf{S} \times \mathbf{S} \mid$ *either* y is a $\sqcup$-successor of an R-successor of x; *or* x has an R-successor z with $\mathcal{L}(z) = \emptyset$, y is an ancestor of x and $\ell_{Rx} \subseteq \mathcal{L}(y)$; *or* for some role S, $\langle x, y\rangle \in \mathcal{E}(S)$ and $S \sqsubseteq R\}$

and it can be shown that T is a tableau for D:

1. $D \in \mathcal{L}(x)$ for the root x_0 of $\mathbf{T}$ and for all $\sqcup$-successors of x_0. As x_0 is marked *satisfiable* one of these must be a non-empty pre-tableau marked *satisfiable*, so $D \in \mathcal{L}(s)$ for some $s \in \mathbf{S}$.

2. Properties 1–3 of Definition 1 are satisfied because each $x \in \mathbf{S}$ is a pre-tableau.

3. Property 4 in Definition 1 is satisfied because $\{C \mid \forall R.C \in \mathcal{L}(x)\} \subseteq \ell_{Rx}$ and $\ell_{Rx} \subseteq \mathcal{L}(y)$ for all y with $\langle x, y\rangle \in \mathcal{E}(R)$.

4. Property 5 in Definition 1 is satisfied by the $\exists$-rule which, for all $x \in \mathbf{S}$, creates for each $\exists R.C \in \mathcal{L}(x)$ a new R-successor y with either:
   (a) $C \in \mathcal{L}(y)$ *or*
   (b) $\mathcal{L}(y) = \emptyset$, $C \in \ell_{Rx}$ and $\ell_{Rx} \subseteq \mathcal{L}(z)$ for some ancestor z of x.
5. Property 6 in Definition 1 is satisfied because $\{\forall S.C \mid \forall R.C \in \mathcal{L}(x),\ S \in \mathbf{R}_+ \text{ and } S \sqsubseteq R\} \subseteq \ell_{Sx}$ and $\ell_{Sx} \subseteq \mathcal{L}(y)$ for all y with $\langle x, y\rangle \in \mathcal{E}(S)$.
6. Property 7 in Definition 1 is satisfied because $\langle x, y\rangle \in \mathcal{E}(S)$ for all $\langle x, y\rangle \in \mathcal{E}(R)$ and $R \sqsubseteq S$.

For the converse (the algorithm returns *unsatisfiable*), it can be shown by induction on $h(x)$, the height of the sub-tree below x, that if x is not marked *satisfiable* then the concept $X = \sqcap_{C \in \mathcal{L}(x)} C$ is not satisfiable:

1. If $h(x) = 0$ (x is a leaf) and x is not marked *satisfiable*, then $\mathcal{L}(x)$ contains a clash and X is clearly unsatisfiable.
2. If $h(x) > 0$, $\mathcal{L}(x)$ is not a pre-tableau and x is not marked *satisfiable*, then none of its $\sqcup$-successors is marked *satisfiable*; hence $C_1 \sqcup C_2 \in \mathcal{L}(x)$ and neither y with $\mathcal{L}(y) = \mathcal{L}(x) \cup \{C_1\}$ nor z with $\mathcal{L}(z) = \mathcal{L}(x) \cup \{C_2\}$ is marked *satisfiable*. It follows by induction that X is not satisfiable.
3. If $h(x) > 0$, $\mathcal{L}(x)$ is a pre-tableau and x is not marked *satisfiable*, then there is some R-successor of x which is not marked *satisfiable* and it follows by induction, and the semantics of value restriction concept expressions ($\forall R.C$), that X is not satisfiable. ∎

4 EXTENDING $\mathcal{ALCH}_{R^+}$ WITH ATTRIBUTES

$\mathcal{ALCH}_{R^+}$ can be extended with limited support for attributes (functional/feature roles) to give $\mathcal{ALCH}f_{R^+}$. Unlike $\mathcal{ALCF}$ (Hollunder and Nutt 1990), $\mathcal{ALCH}f_{R^+}$ does not include support for attribute value map concept forming operators, but it only requires a minor extension to the $\mathcal{ALCH}_{R^+}$ algorithm.

$\mathcal{ALCH}f_{R^+}$ extends the syntax of $\mathcal{ALCH}_{R^+}$ by allowing axioms of the form $A \in \mathbf{F}$ to appear in terminologies, where A is a role name and $\mathbf{F}$ is the set of functional role names, or attributes. As well as being correct for $\mathcal{ALCH}_{R^+}$ concept expressions, an $\mathcal{ALCH}f_{R^+}$ interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ must satisfy the additional condition that, for all $A \in \mathbf{F}$, $A^{\mathcal{I}}$ is a single valued partial function, $A^{\mathcal{I}} : dom\ A^{\mathcal{I}} \longrightarrow \Delta^{\mathcal{I}}$.

The $\exists$-rule in the $\mathcal{ALCH}_{R^+}$ tree construction algorithm can be extended to deal with attributes in $\mathcal{ALCH}f_{R^+}$. Expressions of the form $\exists R.C$, where R is a role, are dealt with exactly as before, but expressions of the form $\exists A.C$, where A is an attribute, require special treatment. The extended rule treats attributes in a similar way to roles: $\exists A.C$ expressions in the label of a pre-tableau node x will lead to the creation of new $\mathbf{A}$-successor nodes y_i and labelled edges $\langle x, y_i\rangle$. However, it may group together multiple $\exists A.C$ expressions in x's label to create a single $\mathbf{A}$-successor y, labeling the edge $\langle x, y\rangle$ with a set of attribute names $\mathbf{A}$.

Multiple $\exists A.C$ expressions must be grouped together when, in the model represented by the tree, the $A^{\mathcal{I}}(x)$ are constrained to be the same individual, for example when there are multiple $\exists A.C$ expressions containing the same attribute A. The interaction between attributes and the role hierarchy means that for two expressions $\exists A.C_1 \in \mathcal{L}(x)$ and $\exists B.C_2 \in \mathcal{L}(x)$, where A and B are attributes, $A^{\mathcal{I}}(x)$ and $B^{\mathcal{I}}(x)$ are also constrained to be the same individual when $A \sqsubseteq B$ (because $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$) or $B \sqsubseteq A$ (because $B^{\mathcal{I}} \subseteq A^{\mathcal{I}}$). We will say that an attribute B is *directly-constrained* by an attribute A in $\mathcal{L}(x)$ if ($\exists A.C \in \mathcal{L}(x)$ and $A \sqsubseteq B$) or ($\exists B.C \in \mathcal{L}(x)$ and $B \sqsubseteq A$), and we will say that an attribute B is *constrained* by an attribute A in $\mathcal{L}(x)$ if B is directly-constrained by A in $\mathcal{L}(x)$ or if, for some attribute A', A' is directly-constrained by A in $\mathcal{L}(x)$ and B is constrained by A' in $\mathcal{L}(x)$. For an attribute B and a node x, the set of attributes which are constrained by B in $\mathcal{L}(x)$ will be denoted $\mathbf{A}_{Bx}$, where $\mathbf{A}_{Bx} = \{A \in \mathbf{F} \mid A$ is constrained by B in $\mathcal{L}(x)\}$. The extended $\exists$-rule for $\mathcal{ALCH}f_{R^+}$ is shown in Figure 4.

5 GENERAL TERMINOLOGIES

The algorithm described in Section 3 tests the satisfiability of a concept expression D w.r.t. a terminology which does not contain concept axioms, but it can also be used to test satisfiability w.r.t. an arbitrary terminology $\mathcal{T}$. If $\mathcal{T}$ is a *restricted* terminology, one which contains only acyclic concept definitions,[3] this can be achieved simply by *unfolding* D—using the definitions in $\mathcal{T}$ to expand concept names in D until they are all atomic primitives. The $\mathcal{ALCH}f_{R^+}$ algorithm can, however, also be used to test the satisfiability of a concept expression with respect to a *general* terminology, one which may contain both cycles and general concept inclusion axioms (GCIs). A GCI is an axiom of the form $C \sqsubseteq D$ where C is an arbitrary concept expression.

[3] A concept definition is an axiom of the form $\mathsf{CN} \sqsubseteq C$ or $\mathsf{CN} \doteq C$, where CN is a concept name which appears only once on the left hand side of such an axiom. Concept definitions are acyclic if C does not refer to CN, either directly or indirectly. Concepts defined by an axiom of the form $\mathsf{CN} \sqsubseteq C$ are called primitive while those defined by an axiom of the form $\mathsf{CN} \doteq C$ are called non-primitive.

if 1.		$\mathcal{L}(x)$ is a pre-tableau, there is a concept of the form $\exists R.C$ in $\mathcal{L}(x)$
2.		x is a leaf of $\mathbf{T}$
then		for each $\exists R.C \in \mathcal{L}(x)$ where $R \notin \mathbf{F}$ do:
	a.	$\ell_{Rx} := \{C\} \cup \{D \mid \forall S.D \in \mathcal{L}(x) \text{ and } R \sqsubseteq S\}$ $\cup \{\forall S.D \mid \forall P.D \in \mathcal{L}(x),\ S \in \mathbf{R}_+,\ S \sqsubseteq P \text{ and } R \sqsubseteq S\}$
	b.	if for some ancestor w of x, $\ell_{Rx} \subseteq \mathcal{L}(w)$ then create an R-successor y of x with $\mathcal{L}(y) = \emptyset$
	c.	otherwise create an R-successor y of x with $\mathcal{L}(y) = \ell_{Rx}$
and		for each $\exists A.D \in \mathcal{L}(x)$ where $A \in \mathbf{F}$ do:
	a.	if for some $\mathbf{A}$-successor y of x, $A \in \mathbf{A}$ then do nothing.
	b.	otherwise
	i.	$\mathbf{A} := \mathbf{A}_{Ax}$
	ii.	$\ell_{Ax} := \bigcup_{B \in \mathbf{A}} (\{C \mid \exists B.C \in \mathcal{L}(x)\} \cup \{C \mid \forall S.C \in \mathcal{L}(x) \text{ and } B \sqsubseteq S\} \cup \{\forall S.C \mid \forall P.C \in \mathcal{L}(x),\ S \in \mathbf{R}_+,\ S \sqsubseteq P \text{ and } B \sqsubseteq S\})$
	iii.	if for some ancestor w of x, $\ell_{Ax} \subseteq \mathcal{L}(w)$ then create an $\mathbf{A}$-successor y of x with $\mathcal{L}(y) = \emptyset$
	iv.	otherwise create an $\mathbf{A}$-successor y of x with $\mathcal{L}(y) = \ell_{Ax}$

Figure 4: Extended $\exists$-rule for $\mathcal{ALCH}f_{R+}$

An axiom $C \doteq D$ is equivalent to two GCIs, $C \sqsubseteq D$ and $D \sqsubseteq C$, so we can, without loss of generality, restrict our attention to GCIs.

A procedure called *internalisation* (Baader 1990) can be used to test the satisfiability of a concept expression D with respect to a terminology $\mathcal{T}$ containing an arbitrary set of GCIs $\{A_1 \sqsubseteq B_1, \ldots, A_n \sqsubseteq B_n\}$. Internalisation works by testing the satisfiability of $D \sqcap \mathcal{M} \sqcap \forall U.\mathcal{M}$, where $\mathcal{M}$ is a concept expression formed from the GCIs, $\mathcal{M} \doteq (B_1 \sqcup \neg A_1) \sqcap \ldots \sqcap (B_n \sqcup \neg A_n)$, and U is a specially defined transitive role which subsumes all the other roles which occur in $\mathcal{T}$. The properties of U ensure that, in any model constructed by the tableaux algorithm, every individual satisfies $\mathcal{M}$, and thus satisfies each of the GCIs in $\mathcal{T}$.

Assuming descriptive rather than fixed point semantics (Nebel 1990), terminological cycles can easily be dealt with by treating all concept axioms as GCIs (Buchheit et al. 1993). However, this method is highly inefficient because reasoning with GCIs introduces large numbers of disjunctions and is thus very costly. Terminological cycles can be dealt with in a much more efficient manner by using *lazy unfolding*: using the definitions in $\mathcal{T}$ to expand concept names in D, but only as required by the progress of the tableau expansion (Baader et al. 1992).

When building a tree $\mathbf{T}$, lazy unfolding ensures that if the terminology $\mathcal{T}$ contains a primitive definition axiom $\mathsf{CN} \sqsubseteq C$, then for any node x in $\mathbf{T}$, $\mathsf{CN} \in \mathcal{L}(x) \Rightarrow C \in \mathcal{L}(x)$. Therefore, in the model represented by $\mathbf{T}$, $\mathsf{CN}^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and the axiom is satisfied. If C refers either directly or indirectly to CN, termination of the tree construction algorithm is still guaranteed because of blocking—most implemented DLs are unable to deal with terminological cycles because they have no blocking mechanism and could not guarantee termination.

Lazy unfolding also takes care of non-primitive definition axioms $\mathsf{CN} \doteq C \in \mathcal{T}$, provided that C can be unfolded so that it contains only primitive concepts, as any primitive interpretation (an assignment of values to the interpretations of primitive concepts) will lead, via the semantics, to an interpretation for CN such that $\mathsf{CN}^{\mathcal{I}} = C^{\mathcal{I}}$. However, if C cannot be unfolded so that it contains only primitive concepts, then it cannot be guaranteed that a model constructed by the algorithm satisfies $\mathcal{T}$. For example, if $\mathcal{T} = \{\mathsf{CN}_1 \sqsubseteq \top, \mathsf{CN}_2 \doteq \neg\mathsf{CN}_2\}$, then $\mathcal{T}$ is obviously unsatisfiable (it only has a model with an empty domain). Testing the satisfiability of CN_1 would, however, cause the algorithm to build a tree representing a model where $\Delta^{\mathcal{I}} = \{x\}$ and $\mathsf{CN}_1^{\mathcal{I}} = \{x\}$.

This problem can be dealt with by checking each definition axiom $\mathsf{CN} \doteq C \in \mathcal{T}$, and if C cannot be unfolded until it contains only primitive concepts, then transforming the axiom into a primitive definition $\mathsf{CN} \sqsubseteq C$ and a GCI $C \sqsubseteq \mathsf{CN}$. The axiom $\mathsf{CN}_2 \doteq \neg\mathsf{CN}_2$ from the above example would thus be converted into the primitive definition $\mathsf{CN}_2 \sqsubseteq \neg\mathsf{CN}_2$ and the GCI $\neg\mathsf{CN}_2 \sqsubseteq \mathsf{CN}_2$. The GCI would lead to CN_2 being added to every node label, and the unfolding of CN_2 would then add $\neg\mathsf{CN}_2$, causing an immediate clash.

6 THE FaCT SYSTEM

The FaCT system is a terminological classifier (TBox) which has been developed as a testbed for a highly optimised implementation of the $\mathcal{ALCH}f_{R^+}$ satisfiability testing algorithm, and to evaluate its empirical tractability. FaCT reasons about concept, role and attribute descriptions, and maintains a concept hierarchy based on the subsumption relation. The algorithm is used for subsumption testing in the usual way: C subsumes D iff $D \sqcap \neg C$ is not satisfiable. Correspondences between modal and description logics (Schild 1991) mean that FaCT can also be used as a theorem prover for the propositional modal logics $\mathbf{K}_{(\mathbf{m})}$, $\mathbf{KT}_{(\mathbf{m})}$, $\mathbf{K4}_{(\mathbf{m})}$ and $\mathbf{S4}_{(\mathbf{m})}$.

6.1 OPTIMISATION TECHNIQUES

A naive implementation of the algorithm would be of limited value in realistic applications: when trying to classify the GALEN medical terminology ontology, for example, single satisfiability problems were encountered which the unoptimised algorithm had failed to solve after 100 hours of CPU time. To improve the performance of the algorithm, a range of optimisations have been employed (Horrocks 1997). These include:

- Lexical normalisation and encoding of concept expressions—a technique which takes the hierarchical structure of terminologies to its logical conclusion by lexically normalising and encoding all concept expressions and, recursively, their sub-expressions. In this form, concept expressions consist only of (possibly negated) concept names, conjunctions ($C_1 \sqcap \ldots \sqcap C_n$) and value restrictions ($\forall R.C$): expressions of the form $\exists R.C$ are transformed into $\neg(\forall R.\neg C)$ and expressions of the form $(C_1 \sqcup \ldots \sqcup C_n)$ are transformed into $\neg(\neg C_1 \sqcap \ldots \sqcap \neg C_n)$. In addition, the sub-expressions forming conjunctions are sorted and any duplicates eliminated. The normalisation process also identifies and simplifies sub-expressions which are obviously satisfiable (e.g., $\forall R.\top$) or obviously unsatisfiable (e.g., $(C \sqcap \neg C \sqcap \ldots)$), replacing them with $\top$ or $\bot$ respectively: in extreme cases (when the whole expression simplifies to $\top$ or $\bot$) the need for a tableau expansion can be completely eliminated.

  The encoding process gives a unique identifier to each lexically distinct concept expressions which, in conjunction with lazy unfolding and the retention of unfolded identifiers, facilitates early clash detection when an identifier and its negation occur in the same node label (Baader et al. 1992).

- Absorption—a technique which eliminates GCIs from a terminology by absorbing them into primitive concept definition axioms. For example, if a terminology contains the axiom $\mathsf{P} \sqsubseteq C$ and the GCI $\mathsf{P} \sqcap D_1 \sqsubseteq D_2$, the GCI can be eliminated from the terminology by absorbing it into the axiom to give $\mathsf{P} \sqsubseteq C \sqcap (D_2 \sqcup \neg D_1)$.

  Although absorption adds a disjunction to the primitive concept definition axiom, it is much more efficient than reasoning w.r.t. the GCI, which would require the disjunction $D_2 \sqcup \neg(\mathsf{P} \sqcap D_1)$ to be added to the label of every node. In effect, absorption restricts the application of this disjunction to nodes where it is really required.

- Semantic branching—a search technique adapted from the Davis-Putnam-Logemann-Loveland procedure (DPL) commonly use to solve propositional satisfiability (SAT) problems (Davis et al. 1962). Semantic branching works by selecting a concept C from one of the unexpanded disjunctions in the label of a node x and searching $\mathcal{L}(x) \cup \{C\}$ and $\mathcal{L}(x) \cup \{\neg C\}$. Wasted search is avoided because the two branches are strictly disjoint. For example, if $\{C \sqcup D, C \sqcup E\} \subseteq \mathcal{L}(x)$ and $\mathcal{L}(x) \cup \{C\}$ is found to be unsatisfiable, then $\neg C$ is added to $\mathcal{L}(x)$ and a second, possibly costly, evaluation of the unsatisfiability of $\mathcal{L}(x) \cup \{C\}$ is avoided. A similar technique is also used in the KSAT modal $\mathbf{K}_{(\mathbf{m})}$ (equivalently $\mathcal{ALC}$ (Schild 1991)) satisfiability testing algorithm (Giunchiglia and Sebastiani 1996).

- Dependency directed backtracking—a technique adapted from constraint satisfiability problem solving (Baker 1995) which addresses the problem of thrashing (large amounts of unproductive backtracking search) caused by inherent unsatisfiability concealed in sub-problems. Backjumping labels concept expressions with a dependency set indicating the branch points on which they depend. When a clash is discovered, the dependency sets can be used to identify the most recent branch point where exploring the other branch might alleviate the cause of the clash. The algorithm can then jump back over intervening branch points *without* exploring alternative branches. A similar technique was used in the HARP theorem prover (Oppacher and Suen 1988).

- Caching and re-using partial models—a technique which takes advantage of the repetitive structure of the satisfiability problems generated during terminological classification by using cached partial tableaux to demonstrate "obvious" satisfiability. For example, the satisfiability of the concept expression $C \sqcap \neg D$ (and thus the non-subsumption $C \not\sqsubseteq D$) can be demonstrated by showing that tableaux for C and $\neg D$ joined at their root nodes result in a valid tableau for $C \sqcap \neg D$.

6.2 EMPIRICAL EVALUATION

The performance of the FaCT system has been evaluated using a variety of empirical testing procedures (Horrocks 1997). When assessing the results of these tests it is important to note the the current system is an experimental prototype written in Common Lisp, and that very little consideration has been given to low-level efficiency issues. The tests have been performed using Allegro CL 4.3 (compiled) on a Sun SPARCstation 20/61 equipped with a 60MHz superSPARC processor, a 1Mbyte off-chip cache and 128Mbytes of RAM. The FaCT system and test KBs are available from the author's home page.

To demonstrate the feasibility of using FaCT with a large, realistic KB, it has been used to classify an $\mathcal{ALCH}f_{R+}$ KB representing the GALEN medical terminology ontology. The KB used in the tests (which has since been extend as part of the ongoing GALEN project) contains 2,740 concepts, 699 of which are non-primitive, 413 roles, 26 of which are transitive, and 1,214 GCIs. Using the optimised algorithm, FaCT is able to classify the KB in ≈379s of CPU time, performing a total of 122,695 subsumption tests at an average of 0.003s per test. FaCT's performance contrasts with that of the KRIS system (Baader and Hollunder 1991) which had only classified a small proportion (≈10%) of a simplified version of the KB (with cycles and transitive roles eliminated) after 100 hours of CPU time.

FaCT also performs well as a modal logic theorem prover: Table 1 compares FaCT with CRACK (Bresciani et al. 1995), KSAT and KRIS using a suite of benchmark formulae for modal **K** (Heuerding and Schwendimann 1996). The tests use 9 classes of formula (*k_branch*, *k_d4*, etc.) in both provable (p) and non-provable (n) forms.[4] For each type of formula, 21 examples of exponentially increasing difficulty are provided, and the table shows the number of the largest formula which each system was able to solve within 100 seconds of CPU time (21 indicates that the hardest problem was solved in less than 100s).

FaCT significantly outperformed all the other systems, and in many cases also exhibited a completely different qualitative performance. For example, with *k_dum_p* formulae (see Figure 5) the other systems all showed an exponential increase in solution times with increasing formula size, whereas the times taken by FaCT increased very little for larger formulae (and FaCT was already 2,000 times faster for the largest formula solved by another system).

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

Table 1: Modal **K** Theorem Proving

	FaCT		**Crack**		**KSAT**		**Kris**	
Test	p	n	p	n	p	n	p	n
k_branch	6	4	2	1	8	8	3	3
k_d4	21	8	2	3	8	5	8	6
k_dum	21	21	3	21	11	21	15	21
k_grz	21	21	1	21	17	21	13	21
k_lin	21	21	5	2	21	3	6	9
k_path	7	6	2	6	4	8	3	11
k_ph	6	7	2	3	5	5	4	5
k_poly	21	21	21	21	13	12	11	21
k_t4p	21	21	1	1	10	18	7	5

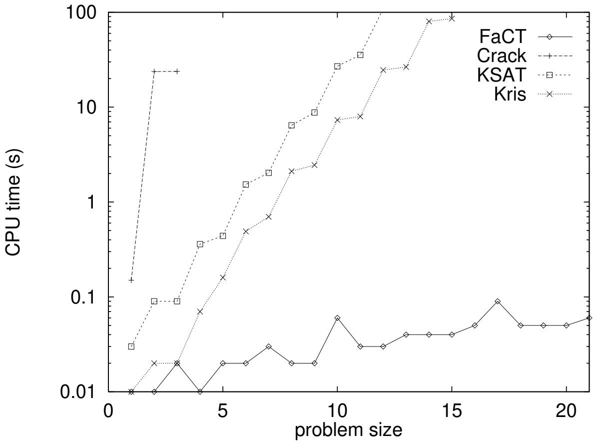

Figure 5: Solution Times for *k_dum_p* Formulae

7 DISCUSSION

This paper describes a sound and complete satisfiability testing algorithm for a relatively expressive DL, one which can reason with respect to a general terminology and a primitive hierarchy of transitive and non-transitive roles. In contrast to most other theoretical presentations, a practical system which uses an (optimised) implementation of the algorithm is also described. The FaCT system has been used to investigate the practicability of using the algorithm for subsumption reasoning, and results so far suggest that in spite of the logic's worst-case intractability the algorithm can provide acceptable performance in realistic applications. FaCT has also been shown to perform well when used as a propositional modal logic theorem prover, and detailed results from these experiments will be the subject of a future paper.

Although the "nice" properties of transitive roles, as opposed to a transitive closure operator (see Section 2), made it simple to implement and optimise the algorithm, many of the techniques investigated could be used with other tableaux satisfiability testing algorithms, and should become standard in future tableaux based DL implementations. Normalisation, encoding and absorption can, for example, be performed as pre-processing steps, and could be used with any DL regardless of its subsumption testing algorithm, although integrating normalisation and encoding with the classifier is preferable in order to avoid the overhead of classifying new concepts generated by the encoding process. The results obtained with FaCT suggest that some of the very expressive DLs for which tableaux algorithms are now available may also be usable in realistic applications, and work is already underway to produce an optimised implementation of such an algorithm.

Acknowledgements

I would like to thank Graham Gough, Alan Rector, Carole Goble and Richard Banach for their help and support. Ulrike Sattler made significant contributions, both technical and inspirational. Thanks are also due to Franz Baader, Sean Bechhofer, Enrico Franconi, Fausto Giunchiglia, Peter Patel-Schneider, Ian Pratt, Dominik Schoop and Roberto Sebastiani for invaluable discussions, suggestions and feedback. Finally, thanks again to Enrico Franconi for his recent hospitality at ITC-IRST, Trento, Italy were some of the work presented was carried out.

The work has been supported by the Engineering and Physical Sciences Research Council, the "Integration and Access to Heterogeneous Databases" Italian Space Agency (ASI) project, and by the "Foundations of Data Warehouse Quality" (DWQ) European ESPRIT Long Term Research Project 22469.

References

L. C. Aiello, J. Doyle, and S. Shapiro, editors. *Principals of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR'96)*. Morgan Kaufmann Publishers, San Francisco, CA, 1996.

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

F. Baader. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Research Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1990.

F. Baader, M. Buchheit, M.A. Jeusfeld, and W. Nutt, editors. *Reasoning about structured objects: knowledge representation meets databases. Proceedings of the 3rd Workshop KRDB'96*, 1996.

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, *Principals 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.

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.

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.

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

J. I. Berman, H. H. Moore IV, and J. R. Wright. CLASSIC and PROSE stories: Enabling technologies for knowledge based systems. *AT&T Technical Journal*, pages 69–78, 1994.

A. Borgida. Description logics are not just for the flightless-birds: A new look at the utility and foundations of description logics. Technical Report DCS-TR-295, New Brunswick Department of Computer Science, Rutgers University, 1992.

P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: a preliminary report. In Ellis et al. (1995), pages 28–39.

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

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

F. M. Donini, M. Lenzerini, D. Nardi, and W. Nutt. The complexity of concept languages. Research Report RR-95-07, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1995.

J. Doyle and R. Patil. Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. *Artificial Intelligence*, 48:261–297, 1991.

Gerard Ellis, Robert A. Levinson, Andrew Fall, and Veronica Dahl, editors. *Knowledge Retrieval, Use and Stor-*

age for Efficiency: Proceedings of the First International KRUSE Symposium, 1995.

G. De Giacomo and M. Lenzerini. TBox and ABox reasoning in expressive description logics. In Aiello et al. (1996), pages 316–327.

F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In Aiello et al. (1996), pages 304–314.

C. A. Goble, C. Haul, and S. Bechhofer. Describing and classifying multimedia using the description logic GRAIL. In *Proceedings of IS&T/SPIE, vol 2670, Storage and Retrieval for Still Image and Video Databases* $\{IV\}$, pages 132–143, San Jose, California, USA, 1996.

R. V. Guha and D. B. Lenat. Enabling agents to work together. *Communications of the ACM*, 37(7):127–142, 1994.

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

A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. Technical report IAM-96-015, University of Bern, Switzerland, 1996.

B. Hollunder and W. Nutt. Subsumption algorithms for concept languages. Research Report RR-90-04, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH (DFKI), 1990.

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

I. Horrocks, A. Rector, and C. Goble. A description logic based schema for the classification of medical data. In Baader et al. (1996), pages 24–28.

P. Hors. Description logics to specify the part–whole relation. In *Proceedings of the ECAI'94 Workshop on Parts and Wholes: Conceptual Part–Whole Relations and Formal Mereology*, pages 103–109, 1994.

U. Küssner. Applying DL in automatic dialoge interpreting. In M.-C. Rousset, R. Brachman, F. Donini, E. Franconi, I. Horrocks, and A. Levy, editors, *Collected Papers from the International Description Logics Workshop (DL'97)*, pages 54–58, 1997.

A. Y. Levy and M.-C. Rousset. Using description logics to model and reason about views. In Baader et al. (1996), pages 48–49.

R. M. MacGregor. The evolving technology of classification-based knowledge representation systems. In J. F. Sowa, editor, *Principals of Semantic Networks: Explorations in the representation of knowledge*, chapter 13, pages 385–400. Morgan-Kaufmann, 1991.

E. Mays, R. Weida, R. Dionne, M. Laker, B. White, C. Liang, and F. J. Oles. Scalable and expressive medical terminologies. In *Proceedings of the 1996 AMAI Fall Symposium*, 1996.

B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Number 422 in Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 1990.

W. A. Nowlan. *Structured methods of information management for medical records*. PhD thesis, University of Manchester, 1993.

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

L. Padgham and P. Lambrix. A framework for part–of hierarchies in terminological logics. In J. Doyle, E. Sandewall, and P. Torasso, editors, *Principals of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR'94)*, pages 485–496. Morgan-Kaufmann Publishers, San Francisco, CA, 1994.

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.

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

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.

U. Sattler. A concept language for engineering applications with part–whole relations. In *Proceedings of the International Conference on Description Logics—DL'95*, pages 119–123, Roma, Italy, 1995.

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.

K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 466–471, 1991.

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

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. (1995), pages 13–27.

W. A. Woods and J. G. Schmolze. The KL-ONE family. *Computers and Mathematics with Applications – Special Issue on Artificial Intelligence*, 23(2–5):133–177, 1992.