Ordering Heuristics for Description Logic Reasoning

Dmitry Tsarkov and Ian Horrocks
School of Computer Science
University of Manchester
Manchester, UK
tsarkov|horrocks@cs.man.ac.uk

Abstract

We present a new architecture for Description Logic implementations, a range of new optimisation techniques and an empirical analysis of their effectiveness.

1 Introduction

Description Logics (DLs) are a family of logic based knowledge representation formalisms. Although they have a range of applications (e.g., configuration [McGuinness & Wright, 1998], and reasoning with database schemas and queries [Calvanese *et al.*, 1998b; 1998a]), they are perhaps best known as the basis for widely used ontology languages such as OIL, DAML+OIL and OWL [Horrocks *et al.*, 2003]. As well as DLs providing the formal underpinnings for these languages (i.e., a declarative semantics), DL systems are also used to provide computational services for ontology tools and applications [Knublauch *et al.*, 2004; Rectore, 2003].

Most modern DL systems are based on tableaux algorithms. Such algorithms were first introduced by Schmidt-Schauß and Smolka [Schmidt-Schauß & Smolka, 1991], and subsequently extended to deal with ever more expressive logics [Baader *et al.*, 2003]. Many systems now implement the $\mathcal{SHIQ}$ DL, a tableaux algorithm for which was first presented in [Horrocks *et al.*, 1999]; this logic is very expressive, and corresponds closely to the OWL ontology language. In spite of the high worst case complexity of the satisfiability/subsumption problem for this logic (ExpTime-complete), highly optimised implementations have been shown to work well in many realistic (ontology) applications [Horrocks, 1998].

Optimisation is crucial to the viability of tableaux based systems: in experiments using both artificial test data and application ontologies, (relatively) unoptimised systems performed very badly, often being (at least) several orders of magnitude slower than optimised systems; in many cases, hours of processing time (in some cases even hundreds of hours) proved insufficient for unoptimised systems to solve problems that took only a few milliseconds for an optimised system [Massacci, 1999; Horrocks & Patel-Schneider, 1998]. Modern systems typically employ a wide range of optimisations, including (at least) those described in [Baader *et al.*, 1994; Horrocks & Patel-Schneider, 1999].

Tableaux algorithms try to construct a graph (usually a tree) representation of a model of a concept, the structure of which is determined by syntactic decomposition of the concept. Most implementations employ a space saving optimisation known as the *trace technique* that uses a top-down construction requiring (for PSpace logics) only polynomial space in order to delineate a tree structure that may be exponential in size (with respect to the size of the input concept). For the ExpTime logics implemented in modern systems, however, guaranteeing polynomial space usage is no longer an option. Moreover, for logics that support inverse roles (such as $\mathcal{SHIQ}$), a strictly top down approach is no longer possible as constraints may be propagated both "up" and "down" the edges in the tree.

We describe an alternative architecture for tableaux implementations that uses a (set of) queue(s) instead of (an adaption of) the standard top-down approach. This architecture, which we have implemented in our new FaCT++ system, has a number of advantages when compared to the top-down approach. Firstly, it is applicable to a much wider range of logics, including the expressive logics implemented in modern systems, because it makes no assumptions about the structure of the graph (in particular, whether tree shaped or not), or the order in which the graph will be constructed. Secondly, it allows for the use of more powerful heuristics that try to improve typical case performance by varying the global order in which different syntactic structures are decomposed; in a top-down construction, such heuristics can only operate on a local region of the graph—typically a single vertex.

2 Preliminaries

We present here a brief introduction to DL (in particular $\mathcal{SHIQ}$) syntax, semantics and reasoning; for further details the reader is referred to [Baader *et al.*, 2003].

2.1 Description Logics

Syntax Let $\mathbf{R}$ be a set of *role names* with both transitive and normal role names $\mathbf{R}_+ \cup \mathbf{R}_\mathsf{P} = \mathbf{R}$, where $\mathbf{R}_+ \cap \mathbf{R}_\mathsf{P} = \emptyset$. The set of $\mathcal{SHIQ}$*-roles* (or *roles* for short) is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. Let N_C be a set of *concept names*. The set of $\mathcal{SHIQ}$*-concepts* (or *concepts* for short) is the smallest set such that every concept name $C \in N_C$ is a concept, and if C and D are concepts, R is a role, S is a *simple* role[1] and $n \in \mathbb{N}$, then $(C \sqcap D)$, $(C \sqcup D)$, $(\neg C)$, $(\forall R.C)$, $(\exists R.C)$, $(\leqslant nR.C)$ and $(\geqslant nR.C)$ are also concepts; the last four are called, respectively, value, exists, atmost and atleast restrictions.

For R and S (possibly inverse) roles, $R \sqsubseteq S$ is called a *role inclusion axiom*, and a finite set of role inclusion axioms is called a *role hierarchy*. For C and D (possibly complex)

[1] A simple role is one that is neither transitive nor has any transitive subroles. Restricting number restrictions to simple roles is required for decidability [Horrocks *et al.*, 1999].

concepts, $C \sqsubseteq D$ is called a *general concept inclusion* (GCI), and a finite set of GCIs is called a *TBox*.

Semantics An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, the *domain* of $\mathcal{I}$, and a function $\cdot^{\mathcal{I}}$ which maps every role to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that, for $P \in \mathbf{R}$ and $R \in \mathbf{R}_+$, $\langle x, y\rangle \in P^{\mathcal{I}}$ iff $\langle y, x\rangle \in P^{-\mathcal{I}}$, and if $\langle x, y\rangle \in R^{\mathcal{I}}$ and $\langle y, z\rangle \in R^{\mathcal{I}}$ then $\langle x, z\rangle \in R^{\mathcal{I}}$. The interpretation function $\cdot^{\mathcal{I}}$ of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ maps, additionally, every concept to a subset of $\Delta^{\mathcal{I}}$ such that

$$\begin{array}{ll}
(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}, & (C \sqcup D)^{\mathcal{I}} \;\; = C^{\mathcal{I}} \cup D^{\mathcal{I}}, \\
\quad \neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, & \\
(\exists R.C)^{\mathcal{I}} \;\; = \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, C) \neq \emptyset\}, & \\
(\forall R.C)^{\mathcal{I}} \;\; = \{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x, \neg C) = \emptyset\}, & \\
(\leqslant nR.C)^{\mathcal{I}} \;\; = \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \leqslant n\}, \text{ and} & \\
(\geqslant nR.C)^{\mathcal{I}} \;\; = \{x \in \Delta^{\mathcal{I}} \mid \sharp R^{\mathcal{I}}(x, C) \geqslant n\}, &
\end{array}$$

where $\sharp M$ is the cardinality of a set M and $R^{\mathcal{I}}(x, C)$ is defined as $\{y \mid \langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$.

An interpretation $\mathcal{I}$ *satisfies* a role hierarchy $\mathcal{R}$ iff $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$ for each $R \sqsubseteq S \in \mathcal{R}$, and it satisfies a TBox $\mathcal{T}$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each $C \sqsubseteq D \in \mathcal{T}$; such an interpretation is called a *model of* $\mathcal{R}$ and $\mathcal{T}$.

A concept C is *satisfiable* w.r.t. a role hierarchy $\mathcal{R}$ and a TBox $\mathcal{T}$ iff there is a model $\mathcal{I}$ of $\mathcal{R}$ and $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a *model* of C w.r.t. $\mathcal{R}$ and $\mathcal{T}$. As usual for expressive DLs, subsumption can be reduced to satisfiability, and reasoning w.r.t. a TBox and role hierarchy can be reduced to reasoning w.r.t. a role hierarchy only [Horrocks *et al.*, 1999].

2.2 Tableaux Algorithms

The basic idea behind a tableau algorithm is to take an input concept C and role hierarchy $\mathcal{R}$, and to try to prove the satisfiability of C w.r.t. $\mathcal{R}$ by constructing a model $\mathcal{I}$ of C w.r.t. $\mathcal{R}$. This is done by syntactically decomposing C so as to derive constraints on the structure of such a model. For example, any model of C must, by definition, contain some individual x such that x is an element of $C^{\mathcal{I}}$, and if C is of the form $\exists R.D$, then the model must also contain an individual y such that $\langle x, y\rangle \in R^{\mathcal{I}}$ and y is an element of $D^{\mathcal{I}}$; if D is non-atomic, then continuing with the decomposition of D would lead to additional constraints. The construction fails if the constraints include a *clash* (an obvious contradiction), e.g., if some individual z must be an element of both C and $\neg C$ for some concept C. Algorithms are normally designed so that they are guaranteed to terminate, and guaranteed to construct a model if one exists; such an algorithm is clearly a decision procedure for concept satisfiability.

In practice, algorithms often work on a tree shaped graph that has a close correspondence to a model; this may be because, e.g., models could be non-finite (although obviously finitely representable), or non-trees (although usually tree-like). Typically this will be a labelled graph (usually a tree or collection of trees) where nodes represent individuals in the model, and are labelled with a set of concepts of which they are instances, and edges represent role relationships between pairs of individuals, and are labelled with a set of role names.

The decomposition and construction is usually carried out by applying so called tableaux expansion rules to the concepts in node labels, with one rule being defined for each of the syntactic constructs in the language (with the exception of negation, which is pushed inwards using re-writings such as de Morgan's laws, until it applies only to atomic concepts). For example, the expansion rule for conjunction causes C and D to be added to any node label already containing $C \sqcap D$ (in order to guarantee termination, side conditions prevent rules from being applied if they do not change either the graph or its labelling).

There are two forms of non-determinism in the expansion procedure. In the first place, many rules may be simultaneously applicable, and some order of rule applications must be chosen. From a correctness perspective, this choice is usually irrelevant[2] (because, if there is a model, then it will be found by any expansion ordering), but as we will see later, the order of expansion can have a big effect on efficiency. In the second place, some rules expand the graph non-deterministically; e.g., the expansion rule for disjunction causes *either* C or D to be added to any node label already containing $C \sqcup D$. From a correctness perspective, this choice *is* relevant (because one choice may lead to the successful construction of a model, while another one does not), and is usually dealt with by backtracking search. Although such search must (in the worst case) consider *all* possible expansions, the order in which they are considered can still have a big effect on efficiency.

Two kinds of rule will be of particular interest in the following discussion: *non-deterministic* rules, such as the $\sqcup$-rule mentioned above, and *generating* rules, such as the $\exists$-rule, that add new nodes to the graph. Applying these rules is likely to be more "costly", as they either increase the size of the graph or increase the size of the search space, and they are typically applied with lower priority than other rules.

3 FaCT++ System Architecture

As discussed above, many implementations use a top-down expansion based on the trace technique. The idea of the top-down expansion is to apply the $\exists$-rule with the lowest priority (i.e., only apply this rule when no other rule is applicable); the added refinement of the trace technique is to discard fully expanded sub-trees, so that only a single "trace" (i.e., a branch of the tree) is kept in memory at any one time.

This technique has the advantage of being very simple and easy to implement—a procedure that exhaustively expands a node label can be applied to the current node and then, recursively, to each of its successors. It does, however, have some serious drawbacks. In the first place, for logics with inverse roles, the top-down method simply breaks down as it relies on the fact that rules only ever add concepts to the label of the node to which they are applied or to the label of one of its successor nodes. The result is that, once the rules have been exhaustively applied to a given node label, no further expansion of that label will be possible. In the presence of inverse roles, expansion rules may also add concepts to the labels of predecessor nodes, which could then require further expansion. Moreover, discarding fully expanded sub-trees may no longer be possible, as the expansion of a concept added to the label of a predecessor may cause concepts to be added to the label of a sibling node that had previously been fully expanded.

In the second place, the top down method forces non-deterministic rules to be applied with a higher priority than generating rules. As the size of the search space caused by non-deterministic rule expansions is, in practice, by

[2] Although the correctness of some algorithms requires a priority ordering for different rules.

far the most serious problem for tableaux based systems [Horrocks, 1997], it may be advantageous to apply non-deterministic rules with the lowest priority [Giunchiglia & Sebastiani, 1996]. In fact, top-down implementations typically apply non-deterministic rules with a priority that is lower than that of all of the other rules *except* the generating rules [Horrocks & Patel-Schneider, 1999].

ToDo List Architecture The FaCT++ system was designed with the intention of implementing DLs that include inverse roles, and of investigating new optimisation techniques, including new ordering heuristics. Currently, FaCT++ implements $\mathcal{SHIF}$, a slightly less expressive variant of $\mathcal{SHIQ}$ where the values in atleast and atmost restrictions can only be zero or one.[3]

Instead of the top-down approach, FaCT++ uses a *ToDo list* to control the application of the expansion rules. The basic idea behind this approach is that rules may become applicable whenever a concept is added to a node label. When this happens, a note of the node/concept pair is added to the ToDo list. The ToDo list sorts all entries according to some order, and gives access to the "first" element in the list.

A given tableaux algorithm takes an entry from the ToDo list and processes it according to the expansion rule(s) relevant to the entry (if any). During the expansion process, new concepts may be added to node labels, and hence entries may be added to the ToDo list. The process continues until either a clash occurs or the ToDo list become empty.

In FaCT++ we implement the ToDo list architecture as a set of queues (FIFO buffers). It is possible to set a priority for each rule type (e.g., $\sqcap$ and $\exists$), and a separate queue is implemented for each unique priority. Whenever the expansion algorithm asks for a new entry, it is taken from the non-empty queue with the highest priority, and the algorithm terminates when all the queues are empty. This means that if the $\exists$-rule has a low priority (say 0), and all other rules have the same priority (say 1), then the expansion will be (modulo inverse roles) top-down and breadth first; if stacks (LIFO buffers) were used instead of queues with the same priorities, then the expansion would simulate the standard top-down method.

4 Heuristics

When implementing reasoning algorithms, heuristics can be used to try to find a "good" order in which to apply inference rules (we will call these *rule-ordering* heuristics) and, for non-deterministic rules, the order in which to explore the different expansion choices offered by rule applications (we will call these *expansion-ordering* heuristics). The aim is to choose an order that leads rapidly to the discovery of a model (in case the input is satisfiable) or to a proof that no model exists (in case the input is unsatisfiable). The usual technique is to compute a weighting for each available option, and to choose the option with the highest (or lowest) weight. Much of the "art" in devising useful heuristics is in finding a suitable compromise between the cost of computing the weightings and their accuracy in predicting good orderings.

Such heuristics can be very effective in improving the performance of propositional satisfiability (SAT) reasoners [Freeman, 1995], but finding useful heuristics for description and modal logics has proved to be more difficult. Choosing a good heuristic, or at least not choosing a bad one, is very important: an inappropriate heuristic may not simply fail to improve performance, it may seriously degrade it. Even more problematical is, given a range of possible heuristics, choosing the best one to use for a given (type of) problem.

So far, the heuristics tried with DL reasoners have mainly been adaptions of those already developed for SAT reasoners, such as the well known MOMS heuristic [Freeman, 1995] and Jeroslow and Wang's weighted occurrences heuristic [Jeroslow & Wang, 1990]. These proved to be largely ineffective, and even to degrade performance due to an adverse interaction with backjumping [Baader *et al.*, 2003]. An alternative heuristic, first presented in [Horrocks, 1997], tries to maximise the effect of dependency directed backtracking (backjumping) by preferentially choosing expansions that introduce concept with "old" dependencies. Even this heuristic, however, has relatively little effect on performance with realistic problems, e.g., problems encountered when reasoning with application ontologies.

We conjecture that the standard top-down architecture has contributed to the difficulty in finding useful heuristics as it rules out many possible choices of rule-ordering; in particular, the top-down technique may require generating rules to be applied with a low priority, and so lead to non-deterministic rules being applied before deterministic generating rules. In contrast, the ToDo list architecture gives a much wider range of possible rule orderings, and so has allowed us to investigate a range of new rule-ordering heuristics, in particular heuristics that give non-deterministic rules the lowest priority.

Another factor that has contributed to the weakness of SAT derived heuristics is that they treat concepts as though they were atoms. This is obviously appropriate in the case of propositional satisfiability, but not in the case of concept satisfiability where sub-concepts may have a complex structure. We have also investigated expansion-ordering heuristics that take into account this structure, in particular a concept's size, maximum quantifier depth, and frequency of usage in the knowledge base.

Implementation in FaCT++ The FaCT++ reasoner uses the standard backtracking search technique to explore the different possible expansions offered by non-deterministic rules (such as the $\sqcup$-rule). Before applying a non-deterministic rule, the current state is saved, and when backtracking, the state is restored before re-applying the same rule (with a different expansion choice). When inverse roles are supported, it is possible for a sequence of deterministic rule applications to propagate changes throughout the graph, and it may, therefore, be necessary to save and restore the whole graph structure (in addition to other data structures such as the ToDo list). FaCT++ trys to minimise the potentially high cost of these operations by lazily saving the graph, (i.e., saving parts of the graph only as necessitated by the expansion), but the cost of saving the state still makes it expensive to apply a non-deterministic rule, even if the state is never restored during backtracking.

As discussed in Section 3, FaCT++ uses a ToDo list architecture with separate queues for each priority level. Different rule-ordering heuristics can, therefore, be tried simply by varying the priorities assigned to different rule types. Low priorities are typically given to generating and non-deterministic rules, but the ToDo list architecture allows different priority ordering of these rule types; in contrast, the top-down architecture forces a lower priority to be given to generating rules.

FaCT++ also includes a range of different expansion-ordering heuristics that can be used to choose the order in which to explore the different expansion choices offered by the non-deterministic $\sqcup$-rule. This ordering can be on the ba-

[3] $\mathcal{SHIF}$ corresponds to the OWL-Lite ontology language [Horrocks *et al.*, 2003].

sis of the size, maximum quantifier depth, or frequency of usage of each of the concepts in the disjunction, and the order can be either ascending (smallest size, minimum depth and lowest frequency first) or descending. In order to avoid the cost of repeatedly computing such values, FaCT++ gathers all the relevant statistics for each concept as the knowledge base is loaded, and caches them for later use.

5 Empirical Analysis

In order to evaluate the usefulness of the heuristics implemented in FaCT++, we have carried out an empirical analysis using both real-life ontologies and artificial tests from the DL'98 test suite [Horrocks & Patel-Schneider, 1998].

Ontologies can vary widely in terms of size and complexity (e.g., structure of concepts, and types of axiom used). We used three ontologies with different characteristics in order to see how the heuristics would perform in each case:

WineFood A sample ontology that makes up part of the OWL test suit[4] [Carroll & De Roo, 2004]; it is small, but has a complex structure and includes 150 GCIs.

DOLCE A foundational (top-level) ontology, developed in the WonderWeb project [Gangemi *et al.*, 2002]; it is of medium size and medium complexity.

GALEN The anatomical part of the well-known medical terminology ontology [Rogers *et al.*, 2001]; it is large (4,000 concepts) and has a relatively simple structure, but includes over 400 GCIs.

FaCT++ separates the classification process into satisfiability testing (SAT) and subsumption testing (SUB) phases; the results from the SAT phase are cached and used to speed up subsequent tests via a standard "model-merging" optimisation [Horrocks & Patel-Schneider, 1999]. FaCT++ allows different heuristics to be used in the two phases of the process; this is because the tests have different characteristics: in the SAT phase, nearly all of the tests are satisfiable (ontologies typically do not give names to unsatisfiable concepts), while in the SUB phase, up to one in four of the tests are unsatisfiable. We measured the time (in CPU seconds) taken by FaCT++ to complete each phase.

In addition to the ontologies, we used artificially generated test data from the DL'98 test suite. Artificial tests are in some sense corner cases for a DL reasoner designed primarily for ontology reasoning, and these tests are mainly intended to investigate the effect of hard problems with very artificial structures on the behaviour of our heuristics. For this purpose we selected from the test suite several of the tests that proved to be hard for FaCT++.

Each of these tests consists of a set of 21 satisfiability testing problems of similar structure, but (supposedly exponentially) increasing difficulty; the idea of the test is to determine the number of the largest problem that can be solved within a fixed amount of processing time (100 seconds of CPU time in our case). The names of the tests are of the form "test_p" or "test_n", where "test" refers to the kind of problem (e.g., the "ph" tests are derived from encodings of pigeon hole sorting problems), and "p/n" refers to whether the problems in the test set are satisfiable (n) or unsatisfiable (p). For these tests we have reported the number of the largest problem solved in less than 100 seconds (21 means that all the problems were solved), along with the time (in CPU seconds) taken for the hardest problem that was successfully solved.

For all the tests, FaCT++ v.0.99.2 was used on Pentium 4 2.2 GHz machine with 512Mb of memory, running Linux. Times were averaged over 3 test runs.

5.1 Rule-ordering Heuristics

In these tests we tried a range of different rule-ordering strategies. Each "strategy" is shown as a sequence of letters specifying the priorities (highest first) of the different rule types, where "O" refers to the $\sqcup$-rule, "E" to the $\exists$-rule, and "a" to any other rule type. E.g., "aO" describes the strategy where the $\sqcup$-rule has the lowest priority, and all other rules have an equal higher priority.

Ontology tests The results of using different rule-ordering strategies with the various ontologies are shown in Table 1. All ontologies were tested with the best disjunction-ordering heuristic, as determined in separate tests (see below).

KB	DOLCE		WineFood		GALEN	
	SAT	SUB	SAT	SUB	SAT	SUB
a	0.74	0.74	0.22	2.44	99.44	1678.11
aO	0.64	0.68	**0.14**	**1.64**	29.80	569, 64
aEO	**0.58**	**0.57**	0.15	1.67	**9.88**	**173.79**
aE	0.60	0.58	0.27	2.87	13.35	205.32
aOE	0.61	0.59	0.27	2.93	13.22	201.40

Table 1: Ontology tests with different rule-orderings

The first thing to note is that rule-orderings have relatively little effect on the DOLCE and WineFood ontologies; in contrast, the performance of the best and worst strategies differs by a factor of almost 10 in the GALEN tests. Even in the GALEN case, however, the difference between the "-O" strategies (i.e., those that assign the lowest priority to the $\sqcup$-rule) and "-E" strategies (i.e., those that assign the lowest priority to the $\exists$-rule) is relatively small. In most cases the best result is given by the "aEO" strategy, i.e., by assigning the lowest priority to the $\sqcup$-rule and the next lowest priority to the $\exists$-rule, and even when "aEO" is not the best strategy, the difference between it and the best strategy is very small. Moreover, the difference between the "aEO" and "aOE" strategies is small in most cases, and never more than a factor of 2.

DL98 tests The results of using different rule-ordering strategies with the DL98 tests are shown in Table 2. The first thing to note from these results is that rule-ordering heuristics can have a much more significant effect than in the ontology tests: in some cases the performance of the best and worst strategies differs by a factor of more than 100. In most tests, the "-E" strategies give the best results, with the difference between "-O" and "-E" strategies being much more marked than in the case of the ontology tests. In the case of the d4_n test, however, performance is dramatically improved (by a factor of 20) when an "-O" strategy is used.

test	br_n		br_p		d4_n		ph_n		ph_p	
	last	time	last	time	last	time	last	time	last	time
a	8	16.7	9	20.5	20	94.8	11	99.0	7	15.5
aO	11	38.2	11	38.1	21	0.8	10	10.8	7	32.1
aEO	11	38.8	11	39.0	21	**0.8**	10	10.9	7	32.9
aE	11	**17.1**	12	**18.3**	21	15.7	11	**97.4**	7	**15.2**
aOE	11	19.3	12	21.1	21	16.1	11	99.5	7	15.9

Table 2: DL-98 tests with different rule-ordering strategies

5.2 Expansion-ordering Heuristics

In these tests we tried a range of different expansion-ordering heuristics. Each heuristic is denoted by two letters, the first of

[4] This ontology therefore has a much weaker claim to being "real-life".

which indicates whether the ordering is based on concept size ("S"), maximum depth ("D") or frequency of usage ("F"), and the second of which indicates ascending ("a") or descending ("d") order. In each group of tests we used the best rule-ordering heuristic as determined by the tests in Section 5.1.

Ontology tests For the ontology tests, we tried different orderings for the SAT and SUB phases of classification. The results are presented in Tables 3, 4 and 5; the first figure in each column is the time taken by the SAT phase using the given ordering, and the remaining figures are the subsequent times taken using different SUB phase orderings.

For DOLCE (Table 3), the difference between the best and worst orderings was a factor of about 4, and many possible orderings were near optimal. For WineFood (Table 4), the difference between the best and worst orderings was a factor of about 2, and using Sd for SAT tests and Dd for SUB tests gave the best result, although several other orderings gave similar results. For GALEN (Table 5), the difference between the best and worst orderings was so large that we were only the orderings given allowed tests to be completed in a reasonable time. The best result was given by using Da for both phases.

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	1.29	1.28	1.24	0.61	0.6	0.6
Sa	2.53	2.52	2.52	2.46	2.45	2.41
Da	2.53	2.53	2.53	2.44	2.44	2.41
Fa	0.91	0.91	0.89	0.97	0.98	0.88
Sd	0.61	0.60	0.60	**0.59**	**0.59**	**0.59**
Dd	0.60	0.60	0.60	**0.60**	**0.59**	**0.60**
Fd	1.33	1.34	1.33	1.30	1.34	1.33

Table 3: DOLCE test with different expansion-orderings

SAT	Sa	Da	Fa	Sd	Dd	Fd
SUB	0.26	0.29	0.19	0.13	0.13	0.20
Sa	3.15	3.57	3.27	3.21	3.21	3.68
Da	3.54	3.57	3.44	3.20	3.40	3.47
Fa	3.67	3.57	2.32	2.12	2.41	2.35
Sd	1.77	1.80	1.71	1.80	1.80	1.83
Dd	1.69	1.77	1.87	**1.66**	1.78	1.78
Fd	2.30	2.26	2.75	3.14	3.54	2.76

Table 4: WineFood test with different expansion-orderings

SAT	Sa	Da
SUB	18.76	9.88
Sa	276.90	276.16
Da	185.79	**172.89**
Fd	1049.74	943.06

Table 5: GALEN test with different expansion-orderings

DL98 tests Table 6 presents the results for the DL98 tests. Each column shows the times taken using different expansion orderings to solve the hardest problem that was solvable within the stipulated time limit using any ordering.

In almost every test, the difference between the best and worst strategies is large: a factor of more than 300 in the d4_n test. Moreover, strategies that are good in one test can be very bad in another (the Sd and Dd strategies are the best ones in the branch tests (br_n and br_p), but (by far) the worst in the d4_n test), and this is not strongly dependent on the satisfiability result (in the br tests, all strategies perform similarly in both satisfiable and unsatisfiable cases). The Fd strategy is, however, either optimal or near optimal in all cases.

order	br_n	br_p	d4_n	ph_n	ph_p
	test 11	test 12	test 21	test 10	test 7
Sa	22.6	24.8	0.9	8.1	29.5
Da	22.6	24.8	0.9	>300	24.5
Fa	>300	>300	32.0	22.9	20.2
Sd	**17.0**	**18.3**	>300	38.7	24.7
Dd	**17.1**	**18.3**	>300	19.7	19.3
Fd	22.2	25.1	**0.8**	**6.2**	**15.3**

Table 6: DL98 tests with different Or strategies

5.3 Analysis

The different rule-ordering heuristics we tried had relatively little effect on the performance of the reasoner when classifying the DOLCE and WineFood ontologies. With the GALEN ontology, any strategy that gave a lower priority to the $\exists$- and $\sqcup$-rules worked reasonably well, and the aEO strategy was optimal or near-optimal in all cases. The crucial factor with GALEN is giving low priority to the $\exists$-rule. This is due to the fact that GALEN is large, contains many GCIs and also contains existential cycles in concept inclusion axioms (e.g., $C \sqsubseteq \exists R.D$ and $D \sqsubseteq \exists R^{-}.C$); as a result, the graph can grow very large, and this increases both the size of the search space (because GCI related non-determinism may apply on a per-node basis) and the cost of saving and restoring the state during backtracking search. Giving a low priority to the $\exists$-rule minimises the size of the graph and hence can reduce both the size of the search space and the cost of saving and restoring. This effect is less noticeable with the other ontologies because their smaller size and/or lower number of GCIs greatly reduces the maximum size of graphs and/or search space. In view of these results, FaCT++'s default rule-ordering strategy has been set to aEO.[5]

The picture is quite different in the case of the DL'98 tests. Here, different strategies can make a large difference, and no one strategy is universally near optimal. This is to be expected, given that some of the tests include very little non-determinism, but are designed to force the construction of very large models (and hence graphs), while others are highly non-deterministic, but have only very small models. Given that these extreme cases are not representative of typical real-life ontologies, the test results may not be directly relevant to a system designed to deal with such ontologies. It is interesting, however, to see how *badly* the heuristics can behave in such cases: in fact the standard aEO strategy is near optimal in two of the tests, and is never worse than the optimal strategy by a factor of more than 2.

The expansion-ordering heuristics had a much bigger effect on ontology reasoning performance (than the rule-ordering heuristics). In the case of DOLCE and WineFood, almost any strategy that uses Sd or Dd in the SUB phase is near optimal. For GALEN, however, using Da in both phases gives by far the best results. This is again due to the characteristic structure of this ontology, and the fact that preferentially choosing concepts with low modal depth tends to reduce the size of the graph. Unfortunately, no one strategy is universally good (Da/Da is best for GALEN but worst for DOLCE and WineFood); currently, Sd/Dd is the default setting, as the majority of real life ontologies resemble DOLCE and WineFood more than GALEN), but this can of course be changed

[5] Top-down architectures necessarily give lowest priority to the $\exists$-rule, and generally give low priority to $\sqcup$-rule, which is why they work relatively well with ontologies.

by the user if it is known that the ontology to be reasoned with will have a GALEN-like structure.

For the DL'98 tests, the picture is again quite confused: the Sd strategy (the default in the SAT phase) is optimal in some tests, but bad in others—disastrously so in the case of the d4_n test. As in the ontology case, the only "solution" offered at present is to allow users to tune these settings according to the problem type or empirical results.

6 Discussion and Future Work

We have described the ToDo list architecture used in the FaCT++ system along with a range of heuristics that can be used for rule and expansion ordering. We have also presented an empirical analysis of these heuristics and shown how these have led us to select the default setting currently used by FaCT++.

These default settings reflect the current predominance of relatively small and simply structured ontologies. This may not, however, be a realistic picture of the kinds of ontology that we can expect in the future: many existing ontologies (including, e.g., WineFood) pre-date the development of OWL, and have been translated from less expressive formalisms. With more widespread use of OWL, and the increasing availability of sophisticated ontology development tools, it may be reasonable to expect the emergence of larger and more complex ontologies. As we have seen in Section 5.1, heuristics can be very effective in helping us to deal efficiently with such ontologies, but choosing a suitable heuristic becomes of critical importance.

In our existing implementation, changing heuristics requires the user to set the appropriate parameters when using the reasoner. This is clearly undesirable at best, and unrealistic for non-expert users. We are, therefore, working on techniques that will allow us to guess the most appropriate heuristics for a given ontology. The idea is to make an initial guess based on an analysis of the syntactic structure of the ontology (it should be quite easy to distinguish GALEN-like ontologies from DOLCE and WineFood-like ontologies simply by examining the statistics that have already been gathered for use in expansion-ordering heuristics), with subsequent adjustments being made based on the behaviour of the algorithm (e.g., the size of graphs being constructed).

Another limitation of the existing implementation is that a single strategy is used for all the tests performed in the classification process. In practice, the characteristics of different tests (e.g., w.r.t. concept size and/or satisfiability) may vary considerable, and it may make sense to dynamically switch heuristics depending on the kind of test being performed. This again depends on having an effective (and cheap) method for analysing the likely characteristics of a given test, and syntactic and behavioural analyses will also be investigated in this context.

References

[Baader *et al.*, 1994] 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. *Applied Artificial Intelligence*, 4:109–132, 1994.

[Baader *et al.*, 2003] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, eds. *The Description Logic Handbook*. CUP, 2003.

[Calvanese *et al.*, 1998a] D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. of PODS'98*, 1998.

[Calvanese *et al.*, 1998b] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Description logic framework for information integration. In *Proc. of KR'98*, pages 2–13, 1998.

[Carroll & De Roo, 2004] J. Carroll and J. De Roo. OWL web ontology language test cases. W3C Recommendation, 2004.

[Freeman, 1995] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, University of Pennsylvania, 1995.

[Gangemi *et al.*, 2002] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with DOLCE. In *Proc. of EKAW 2002*, 2002.

[Giunchiglia & Sebastiani, 1996] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for $\mathcal{ALC}$. In *Proc. of KR'96*, pages 304–314, 1996.

[Horrocks *et al.*, 1999] I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In *Proc. of LPAR'99*, pages 161–180, 1999.

[Horrocks *et al.*, 2003] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

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

[Horrocks, 1998] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[Horrocks & Patel-Schneider, 1998] I. Horrocks and P. F. Patel-Schneider. DL systems comparison. In *Proc. of DL'98*, pages 55–57, 1998.

[Horrocks & Patel-Schneider, 1999] I. Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3):267–293, 1999.

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

[Knublauch *et al.*, 2004] H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The protégé OWL plugin: An open development environment for semantic web applications. In *Proc. of ISWC 2004*, 2004.

[Massacci, 1999] F. Massacci. TANCS non classical system comparison. In *Proc. of TABLEAUX'99*, 1999.

[McGuinness & Wright, 1998] D. L. McGuinness and J. R. Wright. An industrial strength description logic-based configuration platform. *IEEE Intelligent Systems*, pages 69–77, 1998.

[Rectore, 2003] A. Rectore. Description logics in medical informatics. In *The Description Logic Handbook*, pages 306–346. CUP, 2003.

[Rogers *et al.*, 2001] J. E. Rogers, A. Roberts, W. D. Solomon, E. van der Haring, C. J. Wroe, P. E. Zanstra, and A. L. Rector. GALEN ten years on: Tasks and supporting tools. In *Proc. of MEDINFO2001*, pages 256–260, 2001.

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