DLP and FaCT

Peter F. Patel-Schneider[1] and Ian Horrocks[2]

[1] Bell Labs Research, Murray Hill, NJ, U.S.A.
pfps@research.bell-labs.com
[2] University of Manchester, Manchester, UK
horrocks@cs.man.ac.uk

DLP [Patel-Schneider(1998)] and FaCT [Horrocks(1998)] are two recent description logic systems that contain sound and complete reasoners for expressive description logics. Due to the equivalences between expressive description logics and propositional modal logics, both DLP and FaCT can be used as satisfiability checkers for propositional modal logics.

FaCT is a full-featured system that contains a highly-optimized satisfiability checker for a superset of $\mathbf{K4_{(m)}}$. FaCT has an interface to allow the direct satisfiability checking of propositional modal formulae. FaCT is available for research purposes from *http://www.cs.man.ac.uk/˜ horrocks*.

DLP is an experimental system, designed to investigate various optimization techniques for description logic systems, including many of the optimizations pioneered in FaCT. DLP is available for research purposes from *http://www.bell-labs.com/user/pfps /dlp*. DLP contains a highly-optimized satisfiability checker for a superset of Propositional Dynamic Logic ($\mathbf{PDL}$), and includes a simple interface for the direct checking of the satisfiability of formulae in $\mathbf{PDL}$ as well as the modal logics $\mathbf{K_{(m)}}$, $\mathbf{KT_{(m)}}$, $\mathbf{K4_{(m)}}$, and $\mathbf{S4_{(m)}}$.

Both DLP and FaCT have performed very well on several comparisons of modal provers [Horrocks and Patel-Schneider(1998a),Horrocks and Patel-Schneider(1998b)]. The remainder of this submission will concentrate on DLP, as it is somewhat faster than FaCT. Significant differences from FaCT will be noted.

Architecture and Algorithm

At the heart of the DLP system is its highly-optimized tableaux satisfiability engine. DLP first performs a lexical normalization phase, which uniquely stores sub-formulae; eliminates repeated conjuncts and disjuncts; replaces local tautologies and contradictions with true and false, respectively; and performs several other normalization steps. It then attempts to construct a model of the normalized formulae; if it can construct the model then the formulae is satisfiable, if not, the formula is unsatisfiable.

DLP deals with non-determinism in the model construction algorithm by performing a semantic branching search, as in the Davis-Putnam-Logemann-Loveland procedure (DPLL), instead of the syntactic branching search used by most earlier tableaux based implementations [Giunchiglia and Sebastiani(1996)]. DLP deterministically expands disjunctions that present only one expansion possibility and detects a clash when a disjunction has no expansion possibilities.

DLP performs a form of dependency directed backtracking called *backjumping*, backtracking to the most-recent choice point that participates in a clash instead of to the most-recent choice point. To support backjumping, DLP keeps associated with each formula the set of choice points that gave rise to that formula.

DLP (but not FaCT, which has a different caching mechanism) caches the satisfiability status of all modal nodes that it encounters, and uses this status when a node with the same formula is seen again. DLP uses a combination of heuristics to determine the next disjunct on which to branch: it tries to maximize backjumping by first selecting disjunctions that do not depend on recent choice points, and it tries to maximize deterministic expansion by using the MOMS heuristic [Freeman(1996)] to select a disjunct from amongst these disjunctions. DLP defers modal processing until all propositional processing is complete at a node, again using a backjumping maximization heuristic to determine the order in which modal successors are explored.

To handle transitive modalities, and modality constructs in $\mathbf{PDL}$, DLP checks for loops in the model it is constructing. If a loop is detected, it must be classified as either as a loop that leads to satisfiability or a loop that is unsatisfiable. This loop checking allows DLP to handle the S4 problem classes.

Implementation

DLP is implemented in Standard ML of New Jersey, and uses many of the features of the standard libraries of this language. DLP is a mostly-functional program in that the core of the engine has no side-effects. In fact, the only side effects in the satisfiability engine involve the unique storage of sub-formulae and node caching. (FaCT has a more traditional implementation in LISP.)

The unique storage of sub-formula and node caching are handled in DLP by a formula cache. When a formula is encountered, it is looked up in the cache. If the formula is in the cache, it is reused; if not, a new formula is created and added to the cache. Each formula has a satisfiability status; when a new node is created, the formulae for the node are conjoined and this formula is looked up in the formula cache; when a node's status is determined, the satisfiability status of its formula is updated.

Special Features

Full $\mathbf{PDL}$ loop checking can be replaced in DLP by a simpler (and much less costly) loop checking mechanism for transitive modalities. An optimization that is valid for transitive modalities but not for transitive closure can also be enabled. These changes turn DLP into a satisfiability checker for a multi-modal logic where some or all of the modalities may be transitive. The standard embedding can also be used to allow DLP to reason with reflexive modalities. DLP is therefore able to handle many modal logics, including $\mathbf{K_{(m)}}$, $\mathbf{KT_{(m)}}$, $\mathbf{K4_{(m)}}$, and $\mathbf{S4_{(m)}}$. DLP was recently extended to allow global axioms.

DLP has many options, including options to turn off all the above non-heuristic optimizations and options to vary the heuristic optimizations. The version of DLP used in the tests employs the simpler transitive modality loop checking, has all optimizations enabled, and uses the backjumping maximization and MOMS heuristics as described above.

Problem	Num	Sat	Time Outs	Time	(sat)
p-bound-cnf-K3-C8-V4-D2	16	16	0	0.016	0.003
p-bound-cnf-K3-C16-V4-D2	16	16	0	0.032	0.006
p-bound-cnf-K3-C32-V4-D2	16	16	0	0.064	0.014
p-bound-modK-K3-C8-V4-D2	16	16	0	0.050	0.007
p-bound-modK-K3-C16-V4-D2	16	16	0	0.096	0.013
p-bound-modK-K3-C32-V4-D2	16	16	0	0.190	0.027
p-bound-modS4-K3-C8-V4-D2	16	8	7	57.349	57.116
p-bound-modS4-K3-C16-V4-D2	16	1	15	93.910	93.861
p-bound-modS4-K3-C32-V4-D2	16	0	16	100.000	100.000
p-unbound-qbf-cnf-K4-C8-V2-D3	16	15	0	0.162	0.094
p-unbound-qbf-cnf-K4-C16-V2-D3	16	3	0	0.183	0.094
p-unbound-qbf-cnf-K4-C32-V2-D3	16	0	0	0.229	0.093
p-unbound-qbf-modK-K4-C8-V2-D3	16	0	0	0.232	0.053
p-unbound-qbf-modK-K4-C16-V2-D3	16	0	0	0.319	0.058
p-unbound-qbf-modK-K4-C32-V2-D3	16	0	0	0.478	0.064
p-unbound-qbf-modS4-K4-C8-V2-D3	16	0	0	2.496	0.018
p-unbound-qbf-modS4-K4-C16-V2-D3	16	0	0	3.659	0.026
p-unbound-qbf-modS4-K4-C32-V2-D3	16	0	0	6.463	0.049
persat-cnf-K4-C8-V4-D2	16	16	0	0.016	0.009
persat-cnf-K4-C16-V4-D2	16	16	0	0.038	0.025
persat-cnf-K4-C32-V4-D2	16	16	0	0.207	0.185
persat-modK-K4-C8-V4-D2	16	11	5	56.245	56.234
persat-modK-K4-C16-V4-D2	16	1	15	93.769	93.766
persat-modK-K4-C32-V4-D2	16	3	9	74.895	74.869
persat-modS4-K4-C8-V4-D2	16	7	0	10.833	10.666
persat-modS4-K4-C16-V4-D2	16	4	0	7.039	6.714
persat-modS4-K4-C32-V4-D2	16	2	0	3.041	2.395

Table 1. Reference Problems Results

DLP is also a complete description logic system. It has an interface that can be used to define a collection of concepts and roles. DLP automatically computes the subsumption hierarchy of these concepts and provides facilities for querying this hierarchy.

Performance Analysis

DLP was only tested on the problems that used logics $\mathbf{K_{(m)}}$ and $\mathbf{S4_{(m)}}$, possibly including global axioms. Testing was done on a machine with roughly the power of a Sparc Ultra 1. A time limit of 100 seconds was imposed for each problem instance. A special parser was written for DLP to input the problems. (Due to the fact that the problems were not in a format that FaCT could easily handle, FaCT was not run on the problems.)

There are two times reported for each problem class in Table 1. The first times are for an entire run, including inputing the file and normalizing the resulting formula. The second times are for just the satisfiability checker itself.

V	D	C/V									
		2	3	4	5	6	7	8	16	32	64
4	1	0.012	0.014	0.019	0.022	0.026	0.030	0.036	0.087	0.123	0.236
8	1	0.020	0.031	0.039	0.056	0.065	0.078	0.090	0.947	*29.581	0.739
16	1	0.047	0.075	0.105	0.142	0.180	0.214	0.264	0.992	*87.599	* 3.711
4	2	0.016	0.023	0.031	0.040	0.046	0.055	0.064	0.136	0.359	*7.701
8	2	0.034	0.049	0.067	0.084	0.106	0.129	0.154	0.402	1.235	2.269
16	2	0.016	0.023	0.031	0.040	0.046	0.055	0.064	0.136	0.359	*7.701

Table 2. Generated Problems Results—bound-cnf-K3 (average time)

Many of the reference problems were easy for DLP. The problems that were hard for DLP were the bound-modS4 problems and the persat-modK-K4 problems. They were *much* harder than the other problems. For the S4 problems this is probably because DLP uses an equality test to cut off modal loops. A subset test would probably be more effective We do not know why DLP is slow on the persat-modK-K4 problems.

DLP was run on some larger bound-cnf-K3 problems. The results are shown in Table 2. Times marked with a '*' indicate that some problem instances in the particular test exceeded the time bound.

DLP has also been run on a number of other test suites. It did very well on the Tableaux'98 test suite. It also performs well on random formulae generated by other generators. A plot of its performance on random bound-cnf-K3 formulae with a modal depth of 2 and and 9 variables is given Figure 1. The plot shows the 50th, 60th, 70th, 80th, 90th, and 100th percentiles of run time in seconds for various values of C/V (the ratio of clauses to variables). These results are competitive with the fastest propositional modal provers.

Future Work

We are in the process of designing and implementing a successor to DLP. This successor will have a different algorithmic base, and incorporate a newer backtracking optimization called dynamic backtracking [Ginsberg(1993)]. This will allow the optimized handling of nominals, or description logic individuals.

References

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

[Ginsberg(1993)] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[Giunchiglia and Sebastiani(1996)] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures—the case study of modal K. In M. McRobbie and J. Slaney, editors, *Proceedings of the Thirteenth International Conference*

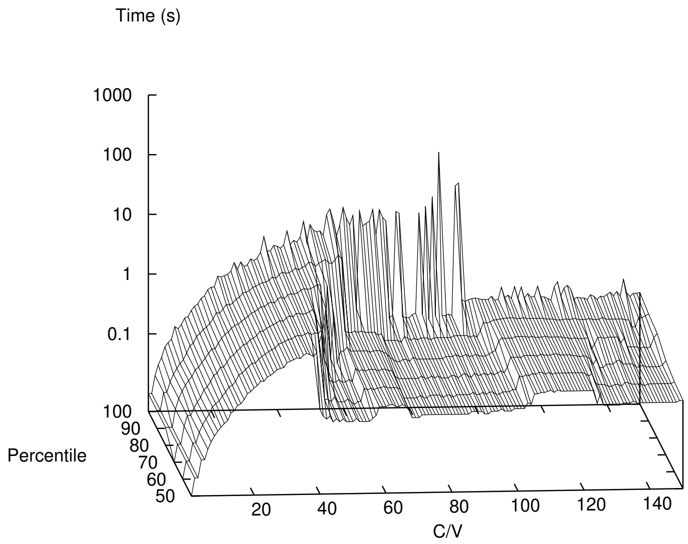

Fig. 1. Percentile times for formulae with 9–1350 clauses (C) and 9 variables (V)

on Automated Deduction (CADE-13), number 1104 in Lecture Notes in Artificial Intelligence, pages 583–597. Springer-Verlag, 1996.

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

[Horrocks and Patel-Schneider(1998a)] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In H. de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 27–30. Springer-Verlag, 1998a.

[Horrocks and Patel-Schneider(1998b)] I. Horrocks and P. F. Patel-Schneider. Optimising propositional modal satisfiability for description logic subsumption. In *International Conference AISC'98*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1998b.

[Patel-Schneider(1998)] Peter F. Patel-Schneider. DLP system description. In E. Franconi, G. De Giacomo, R. M. MacGregor, W. Nutt, C. A. Welty, and F. Sebastiani, editors, *Collected Papers from the International Description Logics Workshop (DL'98)*, pages 87–89, 1998.