What's New in DLP

Peter F. Patel-Schneider
Bell Labs Research
Murray Hill, New Jersey 07974
U. S. A.
pfps@research.bell-labs.com

Abstract

DLP has undergone a number of modifications in the last year. Most of the modifications are further optimisations, but DLP now has an option to compute and return the full model it generates for satisfiable formulae. The new optimisations include some low-level optimisations, more cached information to help in backjumping, early investigation of modal successors, and a form of dynamic backtracking. None of these optimisations produce the dramatic gains of earlier optimisations, but they do augment the space of optimisations for incorporation into future description logic systems.

1 Introduction

DLP (for Description Logic Prover) is a description logic system that contains a sound and complete reasoner for an expressive description logic. DLP is an experimental system, in that it does not contain all the support facilities that one might like in a full description logic system. However, DLP does have an interface that allows the creation and examination of taxonomies of concepts.

Because of the correspondence between description logics and propositional modal logics, DLP can serve as a reasoner for several propositional modal logics. DLP provides interfaces that allow the satisfiability checking of formulae in propositional dynamic logic (PDL), $\mathbf{K_{(m)}}$, $\mathbf{K4_{(m)}}$, $\mathbf{KT_{(m)}}$, and $\mathbf{S4_{(m)}}$.

DLP employs a heavily-optimised decision procedure for description logic satisfiability to determine subsumption relationships. This decision procedure has excellent performance on a wide variety of test suites, including several propositional modal logic test suites. Recent changes to DLP have incorporated new optimisation techniques and new capabilities into the newest version of DLP, version 4.1.

	DL Syntax	PDL Syntax	Semantics
Concepts (Formulae)	A	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
	$\top$	T	$\Delta^{\mathcal{I}}$
	$\bot$	F	$\emptyset$
	$\neg C$	$\sim C$	$\Delta^{\mathcal{I}} - C^{\mathcal{I}}$
	$C \sqcap D$	$C \wedge D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
	$C \sqcup D$	$C \vee D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
	$\exists R.C$	$\langle R \rangle\, C$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \cap C^{\mathcal{I}} \neq \emptyset\}$
	$\forall R.C$	$[R]\, C$	$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \subseteq C^{\mathcal{I}}\}$
	$\geqslant \mathsf{nP}$		$\{d \in \Delta^{\mathcal{I}} : \mid R^{\mathcal{I}}(d) \mid \ \geq n\}$
	$\leqslant \mathsf{nP}$		$\{d \in \Delta^{\mathcal{I}} : \mid R^{\mathcal{I}}(d) \mid \ \leq n\}$
	$\mathsf{P{:}n}$		$\{d \in \Delta^{\mathcal{I}} : R^{\mathcal{I}}(d) \ni n\}$
Roles (Modalities) (Actions)	P	P	$\mathsf{P}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	$R \sqcup S$	$R \cup S$	$R^{\mathcal{I}} \cup S^{\mathcal{I}}$
	$R \circ S$	$R\,;S$	$R^{\mathcal{I}} \circ S^{\mathcal{I}}$
	R/C	$R\,;C?$	$R^{\mathcal{I}} \cap (\Delta^{\mathcal{I}} \times C^{\mathcal{I}})$
	R^{+}	$R\,;R^{*}$	$\bigcup_{n>1} R^{\mathcal{I}^n}$

Figure 1: Simplified Syntax for DLP

DLP is available via the WWW at http://www.bell-labs.com/user/pfps. DLP is implemented in SML/NJ.

2 Language

DLP implements the description logic in Figure 1. In the syntax chart A is an atomic concept; C and D are arbitrary concepts; P is an atomic role; R and S are arbitrary roles; and n is an integer. There is an obvious correspondence between most of the constructs in this description logic and propositional dynamic logic, which is given in the chart.

As DLP accepts a superset of Propositional Dynamic Logic (PDL), it has to incorporate a complete full PDL loop-checking mechanism. This mechanism is quite expensive, and there are less-expensive mechanisms for logics that contain only transitive roles. To allow for the use of these less-expensive loop-checking mechanisms, DLP can be run in a mode where it accepts only regular and transitive roles. In this mode there are no role constructs allowed, and roles are as in Figure 2. Here P is an atomic regular role and T is an atomic transitive role.

	DL Syntax	Semantics
Roles	P	$\mathsf{P}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	T	$\mathsf{T}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}, \mathsf{T}^{\mathcal{I}} \supseteq \mathsf{T}^{\mathcal{I}} \circ \mathsf{T}^{\mathcal{I}}$

Figure 2: Role Syntax for non-PDL version of DLP

3 Implementation

DLP uses the now-standard method for subsumption testing in description logics, namely translating subsumption tests into satisfiability tests and checking for satisfiability using an optimised tableaux method. DLP was designed from the beginning to be an experimental system. As a result, more attention has been paid to making the internal algorithms correct and efficient in the worst-case than to reducing constant factors. This is also reflected in the internal data structures of DLP, which have been chosen for their flexibility rather than having the absolute best modification and access speeds.

DLP is implemented in SML/NJ instead of a language like C so that it can be more-easily changed. There is some price to be paid for this, as SML/NJ does not allow some of the low-level optimisations possible in languages like C. Further, DLP is implemented in a mostly-functional fashion. The only non-functional portions of the satisfiability checker in DLP have to do with unique storage of formulae, and caching of several kinds of information. All this caching is monotone, i.e., it does not have be undone during a proof, or even between proofs. Nonetheless, DLP is quite fast on several problem sets, including the Tableaux'98 propositional modal logic comparison benchmark [7] and several collections of hard random formulae in **K** [8, 6, 10].

The basic algorithm in DLP is a simple tableau algorithm that searches for a model that demonstrates the satisfiability of a description logic description or, equivalently, a propositional modal logic formula. The algorithm process modal constructs by building successor nodes with attached formulae that represent related possible worlds. The algorithm incorporates the usual control mechanism to guarantee termination, including the full PDL check for looping and determination of good loops and bad loops.

When DLP is run in the mode that disallows the transitive closure of roles and the other role constructors, but does allow transitive roles to be defined, it uses instead an optimised proof rule and loop checker taken from FaCT [5] that is not correct for full PDL. In either mode of DLP the basic tableaux algorithm is quite standard.

The important part of DLP is its suite of optimisation techniques [9], many already present in FaCT [5]. DLP converts incoming formulae into a normal form and uniquely stores sub-formulae. DLP performs semantic branching search.

DLP looks for formulae whose value is determined by the current set of assignments, and immediately gives these formulae the appropriate value. DLP jumps over irrelevant choice points, a technique often called backjumping [1]. DLP caches the results of the satisfiability checks at all successor nodes and reuses these results when a new node with the same set of incoming formulae is generated. DLP performs some simple heuristic ordering of its search, mostly to try to improve the performance of backjumping.

4 New Techniques

Version 4.1 of DLP includes a number of new techniques. Most of these are new optimisation techniques but one new non-optimisation technique is that DLP is now able to return the entire model that it found for satisfiable formulae.

The low-level computations in DLP used to be quite expensive for very large formulae. If the formula was also difficult to solve, this cost would be masked by the search time, but if the formula was easy to solve, the low-level computation cost would dominate the solution time. Version 4.1 of DLP dramatically reduces the time taken for these computations.

In particular, there is a new parameter in DLP that tells DLP to only compute branching heuristics for a limited number of disjuncts or disjunctions. The basic idea here is that if there are many disjuncts or disjunctions active, little will be gained by examining all of them. The standard setting of this parameter limits DLP to considering only the first 50 active disjuncts or disjunctions. Further, the heuristic calculations used to recompute information over and over again. These calculations have been reworked to cache some of this information.

These two changes have the effect of reducing the time for very large but easy-to-solve formulae by about a factor of two. Of course, DLP is still much slower on such large-but-easy formulae than provers that use imperative techniques, but such provers are much harder to build and debug than DLP.

In previous versions, DLP just cached whether a formula set was satisfiable or not, and not how any unsatisfiability came about. When a cached unsatisfiable set was again encountered, there was no way of determining which of the formulae in the formula set were responsible for the unsatisfiability. This meant that dependency information needed for backjumping was lost, and had to be conservatively approximated by using the entire formula set.

For unsatisfiable formula sets DLP now retains in the cache the subset of the formula set that is responsible for its unsatisfiability. When such a set is again encountered this subset is used to determine accurate dependency information so that a better backjumping point can be determined. Of course, this does increase the size of cache entries.

Note that this is actually the correct (i.e., not conservatively approximated) backjumping point based on the search performed when the cached formula set was first investigated. This is probably somewhat worse, on average, than the backjumping point that would have been determined if the formula set was again investigated. The discrepancy comes about because the original investigation used heuristics that depended on the choice-point order of the formulae when the formulae set was first investigated. Although the formulae are still the same, they could have been deduced in a different order than when the formula set was first investigated.

DLP can now also retain not only the status of formula sets, but also the model found if the formula set is satisfiable. This model can be used to restart the search when reinvestigating modal successors, reducing the time overhead for early investigation of modal successors—at the cost of considerably increasing the space required for the cache. Unfortunately, this optimisation currently depends on DLP retaining the entire model, which increases the overhead for this method.

DLP used to completely generate assignments for the current node before investigating any modal successors. The current version of DLP has an option to investigate modal successors whenever a choice point is encountered, a technique taken from KSatC [4]. This option is beneficial in some cases, but increases solution times in others.

DLP now incorporates a variant of dynamic backtracking [3]. When backtracking, any choice point that does not depend on the formula being removed is retained and reused. In particular, the invalidated branch(es) of that choice point are not reinvestigated. Actually, all the retained choice points are replayed before any other deductions or choice are made.

However, choice points are only kept with a node. If the backjumping goes back to a previous node, the replay information about the current node is discarded. It would have been better to keep the information between nodes, but that would have required a much larger effort.

5 Results

These optimisations have been analysed on two different, easy benchmarks. The first benchmark is the Tableaux'98 benchmark [2]. The second benchmark consists of randomly-generated extended 3CNF formulae [4] with maximum modal depth 1, modal probability 0.5, and 6 propositional variables. These benchmarks are by no means comprehensive, but do serve to give some idea of the benefits of the different optimisations.

As these benchmarks do not use the transitive closure operator, this testing

Optimisations	Total Problems Solved
No new optimisations	915
Cache with dependencies	911
Remember successor models	913
Early successors	907
Dynamic backtracking	915

Figure 3: Total Tableaux'98 Problems Solved

was done under the optimised loop-checking version of DLP. Further, the low-level computation optimisations are not analysed here as they cannot be totally turned off.

The total number of problems solved for the Tableaux'98 benchmark are given in Figure 3. None of the new optimisations are effective here, indicating that the overhead for the various optimisations is outweighing their benefits. All of the changes are minor, except, perhaps, for the early successor option.

The situation is somewhat different with the random formulae. As shown in Figure 4, adding dependency information to the cache is ineffective, as is remembering the successor models. At least computing the full model is not expensive, as also shown in Figure 4.

Dynamic backtracking and early investigation of successors are effective, as shown in Figure 5. Dynamic backtracking is uniformly better, whereas early investigation of successors is worse on small, under-constrained formulae[1] and better even than dynamic backtracking on larger, over-constrained formulae. This behaviour of early investigation of successors has been borne out in tests with KSatC [4], but this is the first controlled experiment with this result.

Combining dynamic backtracking with early investigation of successors produces mixed results. For the over-constrained formula, the combination performs slightly better than early investigation of successors alone. However, for the under-constrained formulae, the combination is worse than dynamic backtracking alone. This may be because the implementation of dynamic backtracking does not maintain the information between modal nodes, and early investigation of successors performs more node crossing, at least for under-constrained formulae.

[1]Under-constrained formulae are those formulae where almost all the choices of values for the propositional variables (concepts) that all make the formula satisfiable. Over-constrained formulae are the opposite; here there almost all of the choices demonstrate that the formula is unsatisfiable.

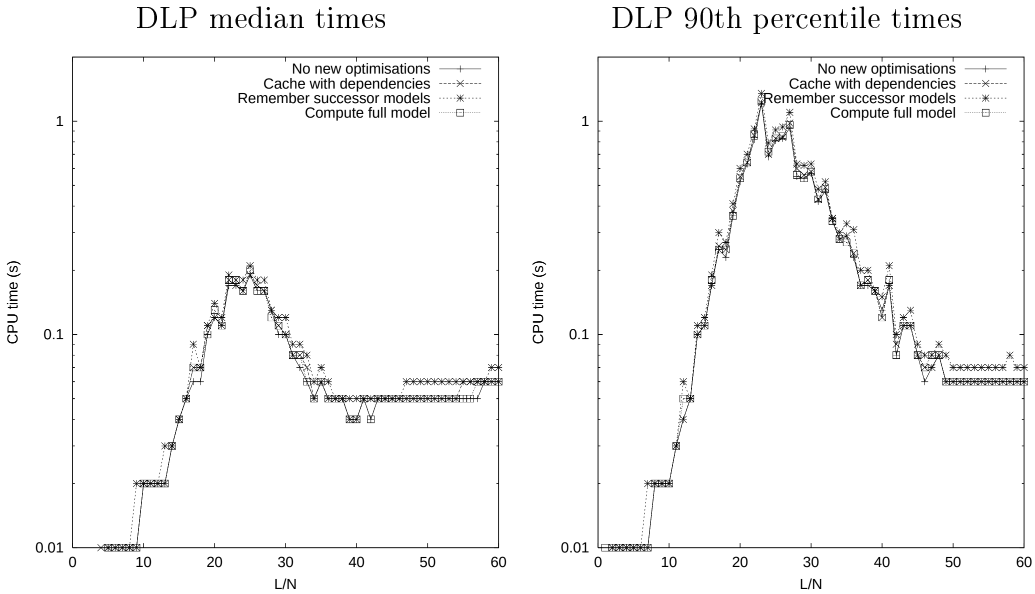

Figure 4: Ineffective Options on Random Formulae

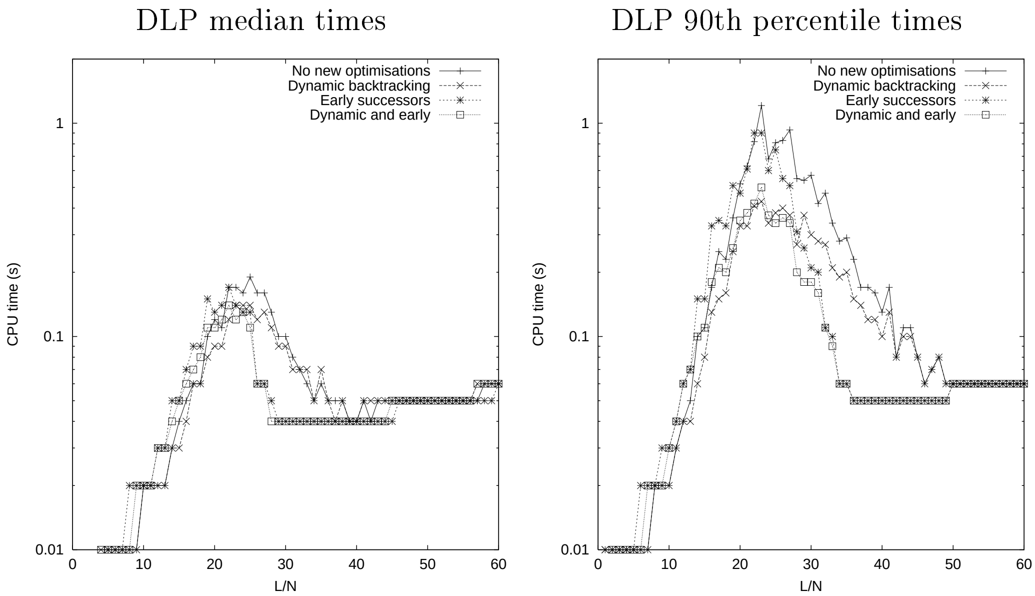

Figure 5: Effective Options on Random Formulae

6 Summary

None of these optimisations produce the dramatic gains of earlier optimisations. The greatest overall speedups in the tests reported above are around a factor of two. This may be because the tests above do not require the new optimisations, or because the optimisation with the most potential, dynamic backtracking, has not been fully implemented.

There are plans to include the above optimisations, along with the other optimisations in DLP and other optimised description logic systems, in a new description system that will implement the description logic $\mathcal{SHIQ}$, with both a TBox and an ABox. This will be a challenge to ingenuity of the developers of the new system as some of the optimisations, in particular caching, are much more difficult to correctly design in the presence of inverse roles, where context is important. However, the development of a highly optimised description logic system directly designed for description logics where context is important is the next hurdle that must be overcome before we can build description logic systems that are viable for useful highly-expressive description logics.

References

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

[2] P. Balsiger and A. Heuerding. Comparison of theorem provers for modal logics — introduction and summary. In Harrie de Swart, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'98*, number 1397 in Lecture Notes in Artificial Intelligence, pages 25–26, Berlin, May 1998. Springer-Verlag.

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

[4] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. More evaluation of decision procedures for modal logics. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98)*, pages 626–635. Morgan Kaufmann Publishers, San Francisco, California, June 1998.

[5] Ian Horrocks. Using an expressive description logic: FaCT or fiction? In Anthony G. Cohn, Lenhart Schubert, and Stuart 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, California, June 1998.

[6] Ian Horrocks, Peter Patel-Schneider, and Roberto Sebastiani. An analysis of empirical testing for modal decision procedures. *Logic Journal of the IGPL*, 8(3):293–324, May 2000.

[7] Ian Horrocks and Peter F. Patel-Schneider. FaCT and DLP. In Harrie 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, Berlin, May 1998. Springer-Verlag.

[8] Ian Horrocks and Peter F. Patel-Schneider. Performance of DLP on random modal formulae. In *Proceedings of the 1999 Description Logic Workshop*, pages 120–124, July 1999.

[9] Peter F. Patel-Schneider. DLP. In *Proceedings of the 1999 Description Logic Workshop*, pages 140–141, July 1999.

[10] Peter F. Patel-Schneider and Ian Horrocks. DLP and FaCT. In Neil V. Murray, editor, *Automated Reasoning with Analytic Tableaux and Related Methods: International Conference Tableaux'99*, number 1617 in Lecture Notes in Artificial Intelligence, pages 19–23, Berlin, June 1999. Springer-Verlag.