A Comparison of Solvers for Propositional Dynamic Logic

Ullrich Hustadt and Renate A. Schmidt

[1] Department of Computer Science, University of Liverpool, Liverpool, UK
U.Hustadt@liverpool.ac.uk
[2] School of Computer Science, The University of Manchester, UK
Renate.Schmidt@manchester.ac.uk

Abstract

Calculi for propositional dynamic logics have been investigated since the introduction of this logic in the late seventies. Only in recent years have practical procedures been suggested and implemented. In this paper, we compare three such systems, namely, the Tableau Workbench by Abate, Goré, and Widmann (2009), the pdlProver system by Goré and Widmann (2009), and the MLSolver system by Friedmann and Lange (2009).

1 Introduction

Propositional dynamic logic (PDL) is an expressive logic for reasoning about programs and actions [7]. Initially intended for program verification, it has found applications in a wide range of areas including verification of rule-based expert systems, synthesis of composite web services, and the formalisation of multi-agent systems.

In recent years there has been renewed interest in PDL and, in particular, in complexity optimal calculi and implementations of theorem provers for PDL [10, 13, 15]. The aim of this paper is to investigate the effectiveness of the current generation of PDL decision procedures. In particular, we are interested in evaluating two features recently introduced into such systems, namely, caching and on-the-fly eventuality checking. To this end we introduce two classes of benchmark formulae for PDL and test the performance of three implemented PDL decision procedures on them.

In Section 2 we give a brief definition of the syntax and semantics of PDL. In Section 3 we discuss the earliest decision procedures for PDL while in Section 4 we do the same for the most recent efforts to develop efficient calculi and implemented systems. In Section 5 we then describe two classes of benchmark formulae that we have used to compare these systems. Section 6 presents the results of benchmarking the Tableau Workbench, pdlProver, and MLSolver on these two classes.

2 Propositional dynamic logic

The language of PDL is defined over a countable set $\mathsf{AP} = \{p, q, \ldots\}$ of *propositional variables* and a countable set $\mathsf{AA} = \{a, b, c, \ldots\}$ of *atomic actions*. The connectives of PDL are the Boolean connectives $\neg$, $\wedge$, $\vee$, the dynamic logic connectives $\vee$, ; *, ?, and the modal operators $[\_]$ and $\langle \_ \rangle$.

The set F of *formulae* and A of *action formulae* are the smallest sets such that (i) $\mathsf{AA} \subseteq \mathsf{A}$, $\mathsf{AP} \subseteq \mathsf{F}$, (ii) if φ and ψ are formulae in F and α and β are action formulae in A then $\varphi?$, α^*, $\alpha \cup \beta$, $\alpha\,;\beta$ are action formulae in A and $\neg\varphi$, $\varphi \wedge \psi$, $[\alpha]\varphi$, and $\langle\alpha\rangle\varphi$ are formulae in F. Additional connections including $\top$, $\bot$, $\vee$, and $\rightarrow$ are defined as usual.

The semantics of PDL is based on Kripke structures. A *frame* is a pair (W, R) where W is a non-empty set of *worlds* and R is a function that maps each atomic action a to a binary relation $R(a)$ over W. A *model* (W, R, I) consists of a frame (W, R) together with an *interpretation function* I that maps each propositional variable p to a set $I(p)$ of worlds. The functions R and I can then be extended to arbitrary action formulae and formulae as follows:

$$
\begin{aligned}
I(\neg\varphi) &= W - I(\varphi) & I(\varphi \wedge \psi) &= I(\varphi) \cap I(\psi)\\
I([\alpha]\varphi) &= \{w \mid \forall v \in W.(w,v) \in R(\alpha) \rightarrow v \in I(\varphi)\} \\
I(\langle\alpha\rangle\varphi) &= \{w \mid \exists v \in W.(w,v) \in R(\alpha) \wedge v \in I(\varphi)\} \\
R(\varphi?) &= \{(w,w) \mid w \in I(\varphi)\} & R(\alpha \cup \beta) &= R(\alpha) \cup R(\beta)\\
R(\alpha\,;\beta) &= \{(w,v) \mid \exists u \in W.(w,u) \in R(\alpha) \wedge (u,v) \in R(\beta)\} \\
R(\alpha^*) &= \{(w,v) \mid \exists n \in \mathbb{N} \exists u_0,\dots,u_n \in W.(u_0 = w \wedge u_n = v \wedge \forall 1 \leq i \leq n-1.(u_i, u_{i+1}) \in R(\alpha)\}
\end{aligned}
$$

Given a model (W, R, I) and a formula φ, we say φ is *true at a world* $w \in W$ iff $w \in I(\varphi)$. A model (W, R, I) *satisfies* a formula φ iff $I(\varphi)$ is non-empty. In this case we also say that φ is *satisfiable in* (W, R, I). A formula φ is *satisfiable* iff there exists a model (W, R, I) satisfying φ.

As is described in more detail in the following two sections, given a formula φ, tableau-based decision procedures for PDL try to build a representation of a model satisfying φ. Such a representation can be viewed as a directed graph whose nodes represent worlds and whose edges represent, and are labelled with, atomic actions linking two worlds. The nodes of the graph are not just labelled with propositional variables, but are also labelled with PDL formulae. The intended meaning is that each of the formulae labelling a node n is true at the world represented by n. If two nodes n and n' are connected via a directed edge from n to n' labelled with an atomic action a, then we say that n' is a-reachable from n. Given the labelling of nodes and edges, we can extend this notion of reachability to arbitrary action formulae.

A particular problem in the construction of a model graph are so-called *eventualities*. Eventualities are formulae of the form $\langle\alpha^*\rangle\varphi$. Suppose a node n in the model graph is labelled with an eventuality $\langle\alpha^*\rangle\varphi$. In order for the graph to represent a model in which $\langle\alpha^*\rangle\varphi$ is true at the world represented by n, we need a node n' in the graph which is α-reachable from n and which is labelled with the formula φ. In the absence of such a node our model will not adhere to the truth conditions for $\langle\alpha^*\rangle\varphi$ as set out by the semantics of PDL. In such a situation, $\langle\alpha^*\rangle\varphi$ is also called an *unfulfilled eventuality.* Detecting unfulfilled eventualities as early as possible in the construction process is a key concern for PDL decision procedures.

3 Early PDL decision procedures

Decision procedures for the satisfiability problem for PDL were first presented by Fischer and Ladner [7] and Pratt [16]. The satisfiability problem for PDL is EXPTIME-complete and already the decision procedure by Pratt [16] was complexity optimal.

Pratt's procedure proceeds in stages. Given a formula, in the first stage a directed graph is constructed with each node being labelled with a set of (labelled) formulae. The construction ensures that there are no two nodes with the same labelling set and that the number of nodes is at most exponential in the size of the given formula. The graph represents a class of potential models of the given formula, but may contain nodes and subgraphs which cannot occur in a model, for example, nodes labelled with inconsistent sets of formulae or subgraphs with unfulfilled eventualities. In subsequent stages these are deleted from the graph. The given formula is satisfiable iff a non-empty graph remains after all necessary deletions have been performed. The construction stage of the procedure can be completed in exponential time in the size of the given formula, each deletion step requires polynomial time in the size of the graph obtained from the construction stage, and there can be at most as many deletion stages as there are nodes in the graph. Overall, this leads to an EXPTIME decision procedure.

Pratt's method has drawbacks which make it impractical for a lot of applications. Most importantly, the initial construction stage can lead to a structure of exponential size even if the satisfiability or unsatisfiability of the given formula only depends on a small subgraph of the whole structure. This means that by the time the procedure enters the second stage, and may detect the satisfiability or unsatisfiability relatively quickly, exponential effort has already been expended on the construction stage.

The tableau calculus for PDL and Converse PDL by De Giacomo and Massacci [6] aims to address this problem. It uses a more traditional approach in which a tableau tree is constructed and explored using depth-first, left-to-right search. Each branch of the tree represents a single candidate model. Propositionally inconsistent sets of formulae are recognised immediately while a check for unfulfilled eventualities is conducted as soon as the construction of a candidate model is completed. This approach leads to a NEXPTIME algorithm. De Giacomo and Massacci claim that storing the whole tableau tree, instead of just a branch, the re-use of tableau nodes across different branches of the tableau, and an "on-the-fly" propagation of information about unsatisfiable sets of formulae leads to an EXPTIME algorithm. In this approach a check for fulfilled and unfulfilled eventualities is still necessary. Important details of this check are however missing in [6].

4 Current PDL calculi and systems

An approach combining features of both Pratt's procedure and De Giacomo and Massacci's tableau calculus is the on-the-fly tableau-based decision procedure by Abate, Goré and Widmann [3]. The procedure constructs a tableau tree where nodes are not only labelled with sets of formulae but also with so-called histories and variables. Histories are used to prevent cyclic applications of the tableau rules. Variables pass information from child nodes to parent nodes, in particular, information about the satisfiability status of a node and information about unfulfilled eventualities. The rules of the calculus specify how the formula sets of child nodes are computed from the formula set of a parent node as well as how the values of variables of a parent node are computed from the values of the corresponding variables in its child nodes. Side conditions on the rules ensure that no infinite branches are constructed thus ensuring termination. Since branches can be at worst exponentially long, a tableau can be of double exponential size. Overall, this results in a 2EXPTIME algorithm. The Tableau Workbench (TWB) [1, 2] includes an implementation of a this algorithm.

In its tableau construction the procedure by Abate, Goré and Widmann is close to that of De Giacomo and Massacci. However, an important difference between the two is the way the check for unfulfilled eventualities is performed. In the tableau calculus of De Giacomo and Massacci, this check can be performed as soon as the construction of one branch of the tableau is completed. The check takes into account information from all the nodes in that branch. If no unfulfilled eventualities are found (and none of the nodes is labelled with an inconsistent set of formulae), then the candidate model associated with the branch is indeed a model for the given PDL formula. However, if the check identifies an unfulfilled eventuality, then the construction moves to an alternative branch of the tableau and another check for unfulfilled eventualities takes place as soon as its construction is completed. Since branches share nodes, this means that nodes will be considered again and again in consecutive checks.

In contrast, the tableau calculus of Abate, Goré, and Widmann uses information passed from child nodes to parent nodes through variables in order to compute whether there are unfulfilled eventualities. The advantage is that the computation is only done once for each node. However, the disadvantage is that the computation can only take place when the information required for the computation is available for all child nodes. This also includes the case where the child nodes are generated by application of a β-rule, e.g., a rule performing a case distinction for a disjunctive formula. Consequently, the value of the variable used for the check for unfulfilled eventualities associated with the root node can potentially only be determined once the whole tableau has been constructed. Thus, while the check for unfulfilled eventualities is not separated into a separate stage of the procedure, the overall behaviour is quite similar to that of Pratt's procedure.

The PDL decision procedure by Goré and Widmann [13] deviates from the classical tableau approach by constructing an and-or graph instead of a tree. Again nodes are labelled by sets of formulae plus additional attributes recording the satisfiability status of a node, information on which eventualities present in the set of formulae associated with the node have been expanded in the node, and which nodes might potentially be used to fulfil each of the eventualities. The construction process ensures that there are no two nodes with the same set of formulae and the same set of eventualities expanded in the node. That is, whenever the application of a tableau rule generates a set of formulae and set of expanded eventualities already present in the graph, the corresponding node is re-used, a technique also called *caching*. As there are at worst an exponential number of distinct sets of formulae and sets of eventualities generated by the tableau rules, the size of the and-or graph is at worst exponential. Just as in the tableau-based decision procedure by Abate, Goré and Widmann [3] the value of the attributes for the satisfiability status of a node and for the information which nodes might potentially be used to fulfil each of the eventualities are computed taking into account information on its successor nodes. The way in which this information is computed appears to differ in that an unsatisfiable status is propagated earlier, but there is no detailed description of the process in [13]. The overall result is an EXPTIME decision procedure. The pdlProver system [12] provides an implementation of that procedure.

LoTREC 2.0 [11, 17] is a generic tableau-based system for building models of formulae in modal and description logics. It includes a module for PDL, however, it cannot be used as a 'black-box' decision procedure like the other systems and is consequently not included in our comparison.

Finally, Friedmann and Lange [10] have proposed a platform for satisfiability checking for various modal fixpoint logics, including PDL. Given a formula their approach generates a parity game as a product of a tableau for the formula and a deterministic automaton recognising 'bad branches' in the tableau. The satisfiability of the formula is then determined by solving the parity game. A generator for these parity games and a solver for them are implemented in the MLSolver system [8] and the PGSolver [9] system, respectively.

5 Benchmark formulae for PDL

Benchmarking implemented systems for non-classical logics is not easy. The number of non-classical logics far outstrips the number of available implemented decision procedures. While each logic is usually reasonably well-motivated by potential applications, the lack of implemented systems usually means that there is no motivation to formalise a large number of problems in one of these logics. Commonly, all one can find is a small number of illustrative formalisations of problems. In the worst case, all one can find is an axiomatisation of the logic which allows one to use instances of the axioms to be used as test cases for an implemented decision procedure. Neither illustrative formalisations nor instances of axioms typically turn out to be particularly challenging and do not allow us to infer much about the properties of the implemented systems.

As an alternative to using real world problems, Balsiger, Heuerding, and Schwendimann [5] suggested the use of synthetic benchmarks consisting of sets of scalable formulae. The selection of suitable benchmarks was supposed to be guided by the following principles: (i) the benchmark sets should contain provable as well as non-provable formulae; (ii) the benchmark sets should vary in structure; (iii) some of the benchmark sets should be hard enough for future decision procedures; (iv) for each formula the satisfiability status should be known; (v) simple 'tricks should not help to solve the formulae; (vi) a 'complete test' should be possible in reasonable time; and (vii) it should be possible to concisely summarise the benchmarking results.

In particular, for each of the modal logics K, KT, and S4 they proposed nine sets of scalable satisfiable formulae and nine sets of scalable unsatisfiable formulae. These benchmark sets were used in a comparison of decision procedures for modal logics conducted in conjunction with the TABLEAUX conference in 1998 [4]. Based on the benchmark results obtained by the various systems at the time, it appears that the benchmark sets have shortcomings regarding the three most important of the seven principles, namely, (iii), (v), and (vii). In particular, it turned out that most of the 18 sets of benchmark formulae were easily solvable. The reason seemed to be that these benchmark formulae were amenable to techniques like Boolean constraint propagation, non-chronological backtracking or the use of proof methods not based on tableau calculi, e.g., translation methods and resolution methods. A few benchmark sets were hard for all the systems involved, for example, pigeon hole formulae disguised by adding occurrences of modal operators. Pigeon hole formulae are known to possess only exponential length refutation in most calculi and obtaining shorter proofs requires conceptually different methods, e.g., the use of cutting plane proof methods. Another problem is that while the results of performance tests for the eighteen classes can be easily summarised, there is no sufficiently fine-grained metric, which one could use to say that one system performs better than another. In general, given the number of benchmark sets the most likely situation is that a system performs slightly better on some and slightly worse on others. For example, in 1998 none of the systems participating in the comparison outperformed all others on all benchmarks sets.

A consequence of these problems is that these benchmark formulae do not provide a motivation for developers of modal theorem provers to further improve their systems. If the system is already reasonably well-developed, then it will solve most of the benchmark formulae easily. Those that remain hard seem to require other methods than the automata, tableau, or resolution methods that most modal theorem provers are based on.

In [14], we have proposed an alternative benchmarking approach, called *scientific benchmarking* or *hypothesis-driven benchmarking*. In this approach benchmark problems are chosen to verify a particular hypothesis concerning the decision procedures under consideration.

In the following, we want to test two hypotheses for the PDL solvers TWB, pdlProver and MLSolver. The first hypothesis concerns the type of formulae for which the re-use of nodes in a tableau construction is advantageous. This should be the case if the number of distinct nodes in a tableau is rather small, but without caching the tableau would still be rather large. The second hypothesis concerns the drawbacks of the two stage approaches or approaches which can only determine the satisfiability of a formula once a tableau has been fully explored.

To test these hypotheses, we re-use two classes of benchmark formulae originally introduced for propositional linear time temporal logic (PLTL) in [14], but reformulated for PDL. The first class, $\mathcal{C}^1_{\mathrm{PDL}}$, consists of formulae of the form

$$\begin{gathered}[a^*]\langle a\rangle\top \wedge [a^*]([a]L^1_1 \vee \ldots \vee [a]L^1_k) \wedge \ldots \wedge [a^*]([a]L^\ell_1 \vee \ldots \vee [a]L^\ell_k)\\ \wedge [a^*](\neg p_1 \vee \langle a^*\rangle p_2) \wedge [a^*](\neg p_2 \vee \langle a^*\rangle p_3) \wedge \ldots \wedge [a^*](\neg p_n \vee \langle a^*\rangle p_1),\end{gathered}$$

while the second class, $\mathcal{C}^2_{\mathrm{PDL}}$, consists of formulae of the form

$$\begin{gathered}[a^*]\langle a\rangle\top \wedge (r_1 \vee L^1_1 \vee \ldots \vee L^1_k) \wedge \ldots \wedge (r_1 \vee L^\ell_1 \vee \ldots \vee L^\ell_k) \wedge (\neg r_1 \vee q_1)\\ \wedge (\neg r_1 \vee \neg q_n) \wedge [a^*](\neg r_n \vee [a]r_1) \wedge [a^*](\neg r_{n-1} \vee [a]r_n) \wedge \ldots \wedge [a^*](\neg r_1 \vee [a]r_2)\\ \wedge [a^*](\neg r_n \vee [a]\neg q_n) \wedge \ldots \wedge [a^*](\neg r_1 \vee [a]\neg q_n) \wedge [a^*](\neg q_1 \vee \langle a^*\rangle s_2)\\ \wedge [a^*](\neg s_2 \vee q_2 \vee [a]q_n \vee \ldots \vee [a]q_3) \wedge \ldots \wedge [a^*](\neg q_{n-1} \vee \langle a^*\rangle s_n) \wedge [a^*](\neg s_n \vee q_n).\end{gathered}$$

For benchmarking purposes, the L^i_1, …, L^i_k are propositional literals generated by choosing k distinct variables randomly from a set $\{p_1, \ldots, p_n\}$ of n propositional variables and by determining the polarity of each literal with probability p. The remainder of each formula only depends on the parameter n. To use these formulae for benchmarking purposes we fix the parameters k, n and p. Then, for each of the values of ℓ between 1 and $8n$ we have generated a test set of 100 formulae, which are tested for satisfiability using the various systems under consideration. Similar to random kSAT formulae, formulae in $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$ are likely to be satisfiable if the number ℓ is small and likely to be unsatisfiable if ℓ is large.

Most of the observations made in [14] about the corresponding PLTL formulae carry over to their PDL counterparts. For example, if a formula in $\mathcal{C}^1_{\mathrm{PDL}}$ is satisfiable, then it is satisfiable in a model with just n worlds. If a formula in $\mathcal{C}^2_{\mathrm{PDL}}$ is satisfiable, then it is satisfiable in a model with just one world and r_1 has to be false at that world.

Given these model-theoretic insights about the formulae, their satisfiability is relatively easy to check, in particular, they are as easy to solve as propositional kSAT formulae over n propositional variables. But the classes are constructed in such a way that PDL decision procedures, which have to rely on proof-theoretic means, find them challenging.

In the case of $\mathcal{C}^1_{\mathrm{PDL}}$, each formula φ_1 in it imposes a uniform set of constraints on all worlds of a model which gives little guidance in the search for a satisfying model. Furthermore, if the propositional formula $(L^1_1 \vee \ldots \vee L^1_k) \wedge \ldots \wedge (L^\ell_1 \vee \ldots \vee L^\ell_k)$ is satisfiable, then potentially every sequence of satisfying truth assignments for this formula could be a model $\mathcal{M}_1$ of φ_1. Only when we check whether all eventualities $\langle a^*\rangle p_i$ are satisfied within $\mathcal{M}_1$ will we know that our search for a model has been successful. We thus expect that naive tableau-based systems and systems, which like Pratt's method only perform an eventuality check after some exhaustive search for candidate models, will perform poorly. On the other hand, decision procedures which use caching should be able to take advantage of the small number of distinct truth assignments that exist for p_1, …, p_n.

The class $\mathcal{C}^2_{\mathrm{PDL}}$ is meant to illustrate how quickly a tableau-based system can find a model for a formula provided it makes the right choices for disjunctive formulae and how efficiently it can recover from making the wrong choices. Decision procedures which use a two stage approach or which can only determine the satisfiability of a formula once a tableau has been fully explored will always consider the part of the tableau on which the propositional variable r_1 is true. However, constructing this part of the tableau is computationally costly and fruitless as no model can be constructed in which r_1 is true. In contrast, a decision procedure which can test candidate models one by one, and happens to first consider models in which r_1 is false, will quickly find a model for satisfiable formulae in this class. For unsatisfiable formulae we do not expect to see a significant difference between the two types of decision procedures as both would need to consider the two cases of r_1 being true and r_1 being false.

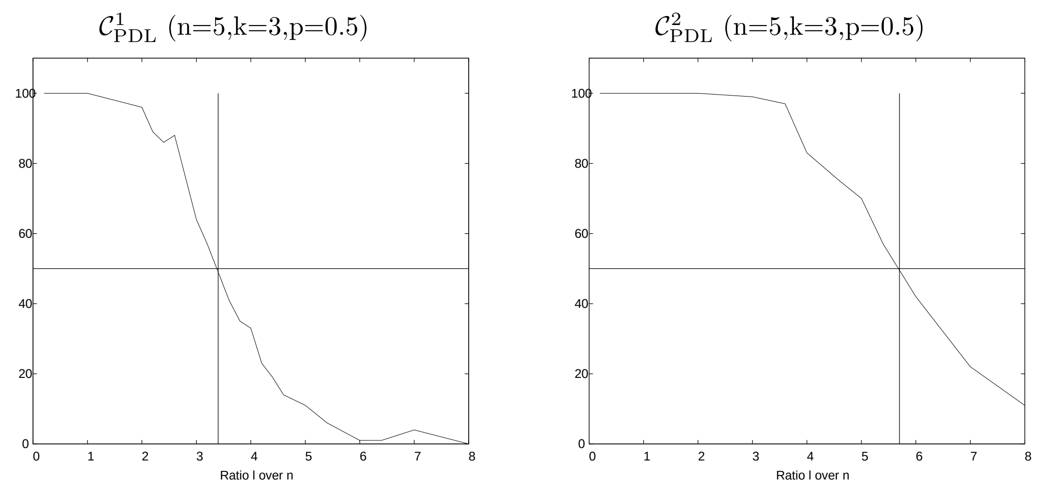

Figure 1: Satisfiability of formulae in $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$

The class can also be used to illustrate problems with transferring variable selection heuristics used in SAT solvers to more complex logics. Commonly used heuristics select the variable with the highest number of occurrences first. In $\mathcal{C}^2_{\mathrm{PDL}}$ this is the variable r_1. If in addition, the first truth assignment used is the one which maximises the number of clauses that are satisfied, then r_1 will be made true first. The fallacy here is to focus solely on a Boolean abstraction of a modal formula. This ignores that in modal logics not all indecomposable subformulae are 'equal'.

6 Benchmarking results

We conducted the benchmarks with the Tableau Workbench, pdlProver and MLSolver on the two classes $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$. The benchmarks were performed on PCs with Intel Core 2 Duo E6400 CPU @ 2.13GHz with 3GB main memory using Fedora 11. For each individual satisfiability test a time-limit of 1000 CPU seconds was used.

In all experiments, for both classes, the parameters k, n and p were fixed to 3, 5, and 0.5, respectively. Remember that the satisfiability problem of propositional 2SAT formula is solvable in polynomial time. So, for $k = 2$, the satisfiability problem of $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$ is also solvable in polynomial time and $k = 3$ is the minimal value for k that ensures that the satisfiability problem of $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$ is NP-complete. The particular choice of p means that the randomly generated literals in our formulae have an equal probability of being positive or negative. Regarding the parameter n, the number of propositional variables we can use in our formulae, note that for $n = 3$ there is only one way of choosing $k = 3$ distinct propositional variables. For $n = 5$ there are ten different ways of choosing three distinct propositional variables, which in turn allows us to build a sufficiently large number of distinct formulae for our experiments.

Figure 1 shows the percentage of satisfiable formulae in $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$ for these parameter values. For $\mathcal{C}^1_{\mathrm{PDL}}$ we see that for ratios ℓ/n smaller than 2 almost all formulae are satisfiable while for ratios ℓ/n greater than 5 almost all formulae are unsatisfiable. For ratios ℓ/n between 2 and 5 we see a phase transition in the satisfiability of formulae. For a ratio ℓ/n equal to 3.4 half the formulae are satisfiable. For $\mathcal{C}^2_{\mathrm{PDL}}$ we see that for ratios ℓ/n smaller than 3.5 almost all formulae are satisfiable and only for ℓ/n greater than 8.0 almost all formulae are unsatisfiable. Here, for the ratio ℓ/n equal to 5.7 half of the formulae are satisfiable.

Figure 2 shows the median CPU time graphs for all three procedures on $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$. In each graph the vertical line indicates the ratio ℓ/n at which test sets contain 50% satisfiable and 50% unsatisfiable formulae.

As can be seen in Figure 2, $\mathcal{C}^1_{\mathrm{PDL}}$ separates pdlProver, the only system which uses caching, from the other two. As suggested, caching allows a prover to take advantage of the uniformity of the constraints imposed on the worlds of a model by formulae in $\mathcal{C}^1_{\mathrm{PDL}}$. Thus, the good performance of pdlProver on this class was predictable. The absence of similar optimisations in the PDL module of the Tableau Workbench and in MLSolver are the most likely explanation for their poor performance. However, even then one might have expected both systems to be able to solve formulae in $\mathcal{C}^1_{\mathrm{PDL}}$ with $\ell/n > 6$, which are almost all unsatisfiable and have a very constrained and limited search space for models.

For $\mathcal{C}^2_{\mathrm{PDL}}$ the ideal system has negligible median runtime for $\ell/n < 5.7$, as up to this point the majority of formulae is satisfiable and a model for a satisfiable formula can easily be found. Only pdlProver could be 'guided' to behave in the expected way (by inputting formulae in the 'right' form, that is, exactly the form given on page 68; changing the order of conjuncts or the order of disjuncts within each conjunction seems to lead to worse results) and to make the right choices in the model construction up to $\ell/n \leq 5.4$ that is almost 'optimal'. In contrast, the Tableau Workbench and MLSolver fail to show a similar behaviour. For MLSolver we also observe a marked difference between $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$. While for $\mathcal{C}^1_{\mathrm{PDL}}$ MLSolver was able to solve the majority of formulae for each ratio ℓ/n, on $\mathcal{C}^2_{\mathrm{PDL}}$ the opposite is true and it solved not a single formula in this class. On both classes the behaviour of the Tableau Workbench and MLSolver is as expected.

Figure 3 shows the CPU time percentile graphs for the three systems on $\mathcal{C}^1_{\mathrm{PDL}}$ and $\mathcal{C}^2_{\mathrm{PDL}}$. The graphs provide additional insight into their behaviour. The x-axis indicates the ratio ℓ/n as in previous figures. The y-axis indicates the percentile, from 10th percentile up to the 100th percentile. The 50th percentile corresponds to the median shown in Figure 2. The z-axis indicates the CPU time. In particular, for MLSolver and pdlProver the graphs confirm our expectations. As a two stage procedure, the performance of MLSolver does not greatly

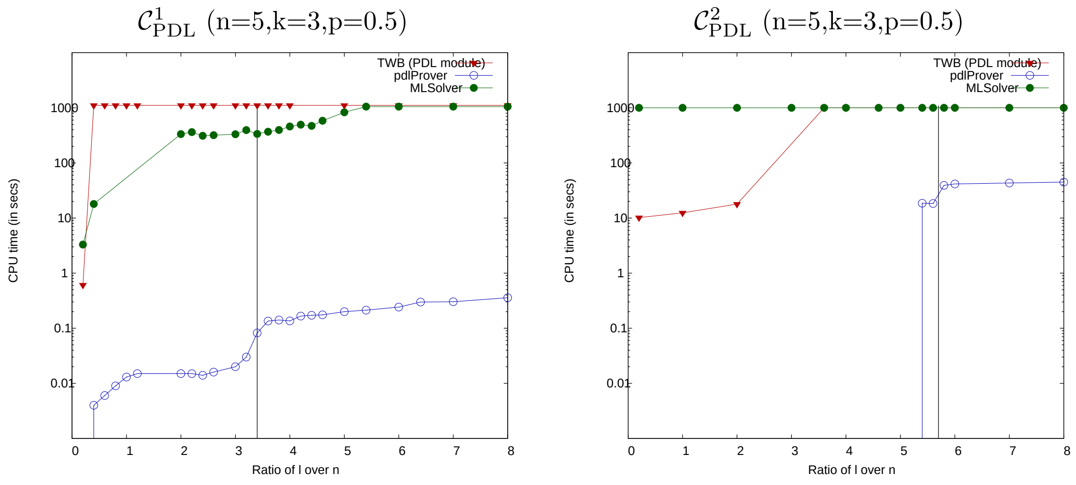

Figure 2: Performance of the decision procedures

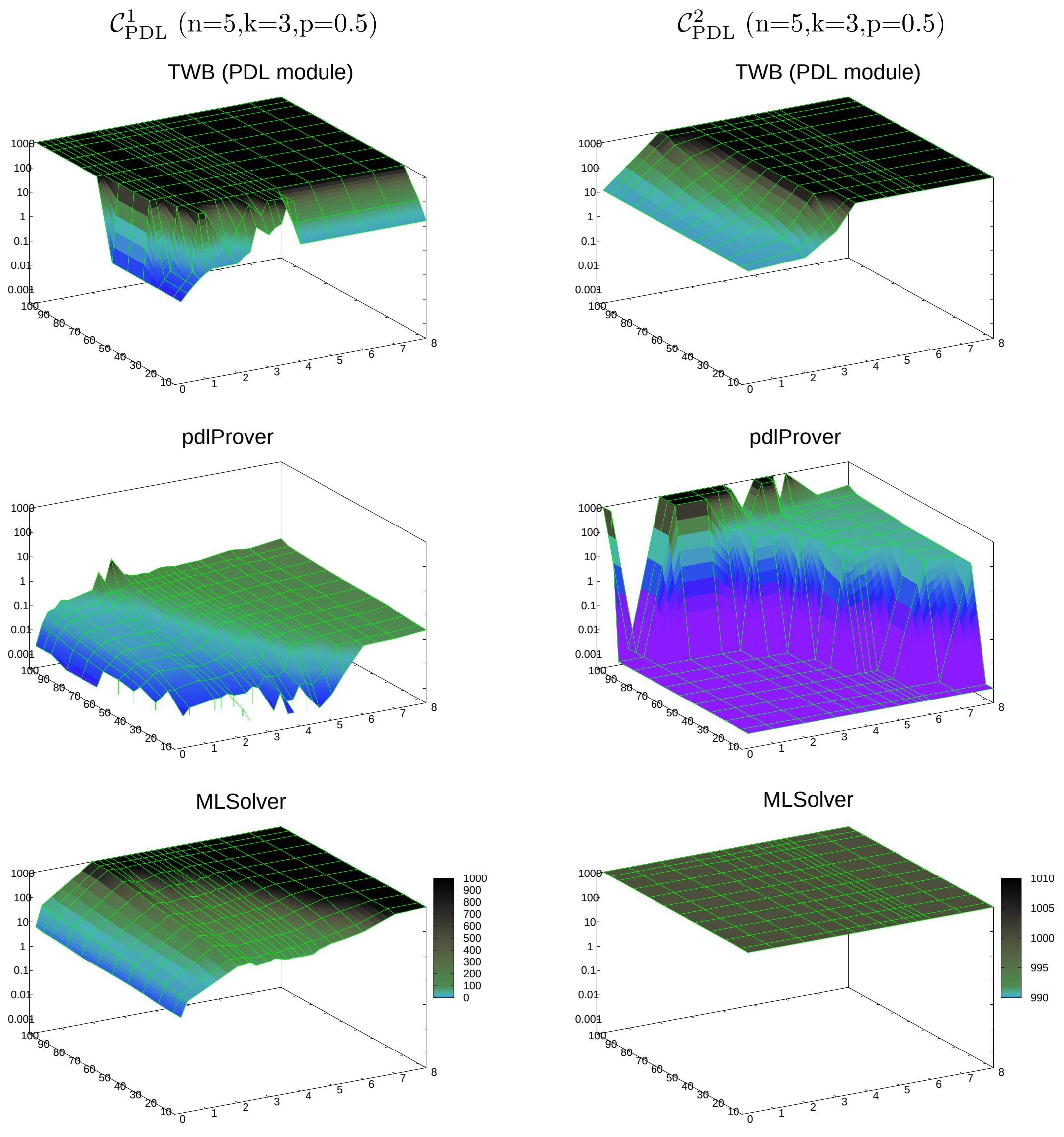

Figure 3: CPU time percentile graphs

depend on whether a formula is satisfiable or unsatisfiable. Similarly, for pdlProver on $\mathcal{C}^1_{\mathrm{PDL}}$, there is little variation in the performance of the system. However, caching allows pdlProver to perform much better than MLSolver. In contrast, on $\mathcal{C}^2_{\mathrm{PDL}}$ the performance of pdlProver is closely related to whether a formula is satisfiable or not. We clearly see that in Figure 3 that as the percentage of unsatisfiable formulae increases so does the percentage of formulae for which pdlProver needs non-negligible time (more than 40 CPU seconds) to solve them.

Overall, pdlProver shows the best performance on these two classes of PDL formulae. The experiments illustrate the importance of caching and of detecting satisfiability as early as possible. In addition, the experiments show that the two classes of benchmark formulae originally devised for PLTL are also useful for 'black-box' performance evaluations of PDL solvers.

7 Conclusion

In this paper we presented benchmarking results for three implemented system for the satisfiability problem in propositional dynamic logic following the hypothesis-driven benchmarking methodology.

The benchmarks presented were intended to test two hypotheses for PDL solvers, namely, (i) that caching is important to control the search space of a system, and (ii) that the possibility of early detection of satisfiability is an essential feature of an efficient PDL solver. The benchmark results seem to support the validity of both hypotheses.

An additional aim of the hypothesis-driven benchmarking methodology is to highlight strengths and weaknesses of particular methods or systems and the benchmark results clearly do so as well.

Finally, the benchmarking approach is intended to motivate implementers to improve their systems. By using formulae for benchmarking whose satisfiability or unsatisfiability is far easier to detect than the worst-case complexity of the satisfiability problem for PDL suggests, there is little excuse for a system to perform badly on these.

References

[1] P. Abate and R. Goré. The Tableau Workbench (TWB). `http://twb.rsise.anu.edu.au/`.

[2] P. Abate and R. Goré. The Tableau Workbench. *Electron. Notes Theor. Comput. Sci.*, 231:55–67, 2009.

[3] P. Abate, R. Goré, and F. Widmann. An on-the-fly tableau-based decision procedure for PDL-satisfiability. *Electr. Notes Theor. Comput. Sci.*, 231:191–209, 2009.

[4] 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)*, volume 1397 of *LNAI*, pages 25–26. Springer, 1998.

[5] P. Balsiger, A. Heuerding, and S. Schwendimann. A benchmark method for the propositional modal logics k, kt, s4. *J. Autom. Reasoning*, 24(3):297–317, 2000.

[6] G. De Giacomo and F. Massacci. Combining deduction and model checking into tableaux and algorithms for converse-PDL. *Info. and Comp.*, 162:117–137, 2000.

[7] M. J. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. Comp. and System Sci.*, 18:194–211, 1979.

[8] O. Friedmann and M. Lange. MLSolver. `http://www2.tcs.ifi.lmu.de/mlsolver/`.

[9] O. Friedmann and M. Lange. PGSolver. `http://www2.tcs.ifi.lmu.de/pgsolver/`.

[10] O. Friedmann and M. Lange. A solver for modal fixpoint logics. In *Prelim. Proc. M4M-6*, pages 176–187. Roskilde University, Denmark, 2009.

[11] O. Gasquet, A. Herzig, D. Longin, and M. Sahade. LoTREC: Logical tableaux research engineering companion. In *Proc. TABLEAUX'05*, volume 3702 of *LNAI*, pages 318–322. Springer, 2005.

[12] R. Goré and F. Widmann. pdlProver. `http://users.cecs.anu.edu.au/~rpg/PDLProvers/`.

[13] R. Goré and F. Widmann. An optimal on-the-fly tableau-based decision procedure for PDL-satisfiability. In *Proc. CADE-22*, volume 5663 of *LNCS*, pages 437–452, 2009.

[14] U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In *Proc. KR2002*, pages 533–544. Morgan Kaufmann, 2002.

[15] L. A. Nguyen and A. Szalas. Optimal tableau decision procedures for pdl. *CoRR*, abs/0904.0721, 2009.

[16] V. R. Pratt. A near-optiomal method for reasoning about actions. *J. Comp. and System Sci.*, 20:231–254, 1980.

[17] B. Said. LoTREC generic tableau prover. `http://www.irit.fr/Lotrec/`.