Formulae which Highlight Differences between Temporal Logic and Dynamic Logic Provers

Ullrich Hustadt[1] and Renate A. Schmidt[2]

[1] Department of Computer Science, University of Liverpool, UK.
U.Hustadt@csc.liv.ac.uk
[2] Department of Computer Science, University of Manchester, UK.
schmidt@cs.man.ac.uk

Abstract. In this Note we compare different inference methods for propositional temporal logic by empirical analysis. We define a class of randomly generated temporal logic formulae which we use to investigate the behaviour of a tableaux-based temporal logic approach using the Logics Workbench, a tableaux-based approach for propositional dynamic logic using an appropriate translation and the prover DLP, and temporal resolution using TRP.

1 Introduction

For propositional linear time logic PLTL there are three major, different approaches: (i) tableaux- and sequent-based calculi, (ii) automata-based approaches, and (iii) temporal resolution calculi. In most publications on tableaux-based calculi for PLTL [6, 10] the decision procedure proceeds in two phases: Given a PLTL formula φ which has to be tested for satisfiability, first, the procedure creates a pre-model $\mathcal{M}$ for the PLTL by applying tableaux expansion rules to φ, then, in the second phase the procedure checks whether $\mathcal{M}$ satisfies all so-called eventuality formulae. In contrast, Schwendimann [9] presents a one-phase tableau calculus which checks locally, on-the-fly, for the fulfillment of eventuality formulae on a branch-by-branch basis. A C++ implementation of this calculus is incorporated into the Logics Workbench [7].

Also most automata-based procedures proceed in two phases: Given a PLTL formula φ which has to be tested for satisfiability, first, an automaton $\mathcal{A}$ is constructed which accepts all models of φ, then, in the second phase the procedure checks whether the set of models, that is, the language accepted by $\mathcal{A}$, is empty. Essentially, this can be done by a simple cycle detection scheme. For our experiments we have used a system written by Klaus Schneider at the University of Karlsruhe which combines a PLTL-to-SMV translator (written in Java) and the SMV model checker (written in C). In the following this system is called TLSMV.

Finally, the resolution method for PLTL proposed by Fisher [4] involves the translation of PLTL formulae to a normal form, classical resolution within states (known as initial and step resolution) and temporal resolution over states between eventuality formulae like $\Diamond \neg p$ and formulae that together imply $\Box p$. For a detailed description of the temporal resolution calculus for PLTL see [4, 2, 3]. We have used an experimental implementation called TRP (written in SICStus Prolog 3.8.5) for our experiments. TRP is an extension of a first-order resolution theorem prover by temporal resolution.

In this paper we consider a class of random formulae with the following characteristics: (i) it should be able to highlight differences between tableaux-based or automata-based approaches to PLTL and temporal resolution, (ii) it should also allow for a 'cross-logic' comparison of decision procedures, that is, the comparison of decision procedures for PLTL and PDL, for example, and (iii) it should provide us with some insight into possible improvements of decision procedures for PLTL and PDL.

2 Basic notions

Let P be a set of propositional variables. The set of formulae of *propositional linear time logic* PLTL (over P) is inductively defined as follows: (i) $\top$ is a formula of PLTL, (ii) every propositional variable of P is a formula of PLTL, (iii) if φ and ψ are formulae of PLTL, then $\neg\varphi$ and $\varphi \vee \psi$ are formulae of PLTL, and (iv) if φ and ψ are formulae of PLTL, then $\bigcirc\varphi$ (*in the next moment of time* φ *is true*), $\Diamond\varphi$ (*sometimes in the future* φ *is true*), $\Box\varphi$ (*always in the future* φ *is true*), $\varphi\,\mathcal{U}\,\psi$ (φ *is true until* ψ *is true*), and $\varphi\,\mathcal{W}\,\psi$ (φ *is true unless* ψ *is true*) are formulae of PLTL. Implicit connectives include $\bot$, $\wedge$, $\rightarrow$, and $\leftrightarrow$.

We also use $\top$ and $\bot$ to denote the empty conjunction and disjunction, respectively. A literal is either p or $\neg p$ where p is a propositional variable. For any (modal) literal L we denote by $\sim L$ the negation normal form of $\neg L$.

PLTL-formulae are interpreted over ordered pairs $\mathcal{I} = \langle \mathcal{S}, \iota\rangle$ where (i) $\mathcal{S}$ is an infinite sequence of *states* $(s_i)_{i\in\mathbf{N}}$ and (ii) ι is a function assigning to each state a subset of P. An interpretation $\mathcal{I}$ *satisfies* a formula φ iff φ holds at s_0 in $\mathcal{I}$ and it *satisfies* a set N of formulae iff for every formula $\psi \in N$, $\mathcal{I}$ satisfies ψ. In this case, $\mathcal{I}$ is a *model* for φ and N, respectively, and we say φ and N are (PLTL-)satisfiable. The satisfiability problem of PLTL formulae is PSPACE-complete.

Let P be a set of propositional variables and A be a set of *atomic actions*. The set of action formulae and formulae of *propositional dynamic logic* PDL is inductively defined as follows: (i) $\top$ is a formula of PDL, (ii) every propositional variable of P is a formula of PDL, (iii) if φ and ψ are formulae of PDL, then $\neg\varphi$ and $\varphi \vee \psi$ are formulae of PDL, and (iv) if φ is a formula of PDL and α is an action formula of PDL, then $[\alpha]\varphi$ and $\langle\alpha\rangle\varphi$ are formulae of PDL, (v) `id` (skip) is an action formula of PDL, (vi) every atomic action of A is an action formula of PDL, (vii) if φ is a formula of PDL, then $\varphi!$ (confirmation or test) is an action formula of PDL, and (viii) if α and β are action formulae of PDL, then $\alpha \vee \beta$ (non-deterministic choice), $\alpha\,;\beta$ (sequencing), and $\alpha^\star$ (unbounded repetition) are action formulae of PDL. Implicit connectives again include $\bot$, $\wedge$, $\rightarrow$, and $\leftrightarrow$.

The semantics of PDL is defined as follows. A PDL-*interpretation* $\mathcal{I}$ is an ordered tuple $\langle \mathcal{W}, \mathcal{R}, \iota\rangle$ where (i) $\mathcal{W}$ is a set of *worlds*, (i) $\mathcal{R}$ contains a binary relation r_a on $\mathcal{W}$ for each atomic action a in A, and (iii) ι is a function assigning to each world a subset of P.

The notions of satisfiability and PDL-satisfiable formulae are defined analogous to the corresponding notion in PLTL. The satisfiability problem of PDL formulae is EXPTIME-complete. For a description of a tableau calculus for PDL (with converse) consult [1].

Deterministic propositional dynamic logic DPDL is a variant of PDL with the same set of formulae and action formulae, and the same semantics except that in a DPDL-*interpretation* $\langle \mathcal{W}, \mathcal{R}, \iota \rangle$ all the binary relations r_a in $\mathcal{R}$ have to be functions.

Arbitrary PLTL-formulae can be transformed into a *separated normal form* (SNF) in a satisfiability equivalence preserving way using a renaming technique replacing non-atomic subformulae by new propositions and removing all occurrences of the $\mathcal{U}$ and $\mathcal{W}$ operator [4, 5]. The result is a set of *SNF clauses* of the following form (which differs slightly from [4, 5]).

(initial clause)	$\bigvee_{i=1}^{n} L_i$
(global clause)	$\Box((\bigvee_{j=1}^{m} K_j) \vee (\bigvee_{i=1}^{n} \bigcirc L_i))$
(sometime clause)	$\Box((\bigvee_{j=1}^{m} K_j) \vee \Diamond L)$

Here, K_j, L_i, and L (with $1 \leq j \leq m$ and $1 \leq i \leq n$) denote literals.

Given a PLTL-formula φ the transformation to SNF starts with a set $\{q, \neg q \vee \varphi\}$, where q is a new propositional variable, and exhaustively applies a set of transformation rules to this set. We do not give a complete description of the transformation rules. The result of the transformation for a formula φ is denoted by $\mathrm{snf}(\varphi)$. For a complete description of snf see [4, 5]. We need some additional terminology for SNF clauses. If $C = \Box((\bigvee_{j=1}^{m} K_j) \vee (\bigvee_{i=1}^{n} \bigcirc L_i)$ is a global clause, then by $\mathrm{now}(C)$ we denote $\bigvee_{j=1}^{m} K_j$ and by $\mathrm{next}(C)$ we denote $\bigvee_{i=1}^{n} L_i$. If $C = \Box(\bigvee_{i=1}^{n} L_i)$, then let $\mathrm{now}(C)$ denote the empty clause and $\mathrm{next}(C)$ denote $\bigvee_{i=1}^{n} L_i$.

Lemma 1 ([4, 5]). *Let φ be a formula of* PLTL. *Then, φ is* PLTL*-satisfiable if and only if* $\mathrm{snf}(\varphi)$ *is* PLTL*-satisfiable, and* $\mathrm{snf}(\varphi)$ *is computable in time polynomial in the size of φ.*

3 Random PLTL-formulae

We now introduce a class of random formulae with the three characteristics identified in the Introduction.

Based on Lemma 1 it is clear that we can restrict ourselves to sets of SNF clauses instead of arbitrary PLTL-formulae. The similarity of SNF clauses to propositional clauses makes it easy to generalise the standard random generator for uniform propositional kSAT clauses to SNF clauses. This can be done in a number of ways. The random generator we have used in our experiments takes as input the number of propositional variables n, the number of clauses l, the number of literals per step clause k, and a probability p. A randomly generated formula φ is a conjunction of global and sometime clauses of the following form

$$\begin{aligned} &\Box(\bigcirc L_1^1 \vee \ldots \vee \bigcirc L_k^1) \wedge \ldots \wedge \Box(\bigcirc L_1^l \vee \ldots \vee \bigcirc L_k^l) \\ \wedge\, &\Box(\neg p_1 \vee \Diamond p_2) \wedge \Box(\neg p_2 \vee \Diamond p_3) \wedge \ldots \wedge \Box(\neg p_n \vee \Diamond p_1) \end{aligned}$$

where for each global clause, the literals $L_1^i, \ldots, L_k^i$ are generated by choosing k distinct variables randomly from the set $\{p_1, \ldots, p_n\}$ of n propositional variables and by determining the polarity of each literal with probability p. The sometime clauses included in φ only depend on the parameter n. In the following we call the formulae generated in this way *random SNF formulae* and denote the class of all random SNF formulae by $\mathcal{C}_{ran}$. In all our experiments the parameters k and p have been fixed to 3 and 0.5, respectively.

The absence of initial clauses and the restricted form of global clauses in random SNF formulae indicates that they form a strict subclass of the class of sets (or conjunctions) of SNF clauses. Thus, not every PLTL-formula is equivalent to some random SNF formula. It turns out that the satisfiability problem of $\mathcal{C}_{ran}$ has reduced complexity.

Theorem 2. *The satisfiability problem for $\mathcal{C}_{ran}$ is in* NP.

Proof (Sketch). Let φ be a random SNF formula over n propositional variables with global clauses $C_1, \ldots, C_l$. Let ψ' be the propositional formula $\mathrm{next}(C_1) \wedge \ldots \wedge \mathrm{next}(C_l)$. Let ψ_0 be the propositional formula $\psi' \wedge \neg p_0 \wedge \ldots \wedge \neg p_n$, and let ψ_i be the propositional formula $\psi' \wedge p_i$ for every i, $1 \leq i \leq n$. Then φ is satisfiable iff ψ_0 is satisfiable or for every i, ψ_i is satisfiable.

Underlying this theorem are considerations about the satisfiability equivalence of formulae. It should be stressed that the fact that a formula φ can be transformed into an 'easy' satisfiability equivalent formula φ' does not indicate that φ will be easy for a particular theorem prover. We can only do so if we know that a particular theorem prover actually performs this particular transformation. However, the proof of Theorem 2 is the key to understanding why TRP has an advantage over the Logics Workbench as we will see below.

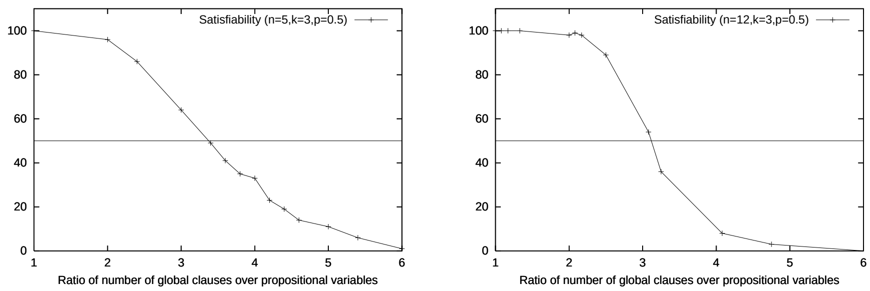

Fig. 1. Satisfiability of random PLTL-formulae

Concerning the satisfiability of random SNF formulae we observe that similar to random kSAT formulae, a random SNF formula is likely to be satisfiable if the number l of clauses is small and it is likely to be unsatisfiable if the number l of clauses is large. Figure 1 shows the percentage of satisfiable random SNF formulae for two values of the parameter n, namely, $n = 5$ and $n = 12$. For each value of n and particular values of l we have generated 100 random SNF formulae which were tested for satisfiability. Both graphs show that for a ratio l/n smaller than 2, a random SNF formula is most likely satisfiable while for a ration l/n greater than 4, it is most likely unsatisfiable. For ratios l/n between 2 and 4 we note a phase transition in the satisfiability of random SNF formula which becomes more abrupt with greater values of n.

4 Experiments

We will now see how the Logics Workbench, TRP, and TLSMV perform on random SNF formulae. The test have been performed on a PC with a 800MHz Pentium III processor, 256MB main memory, and 512MB virtual memory running RedHat Linux 6.2. For each individual satisfiability test a time-limit of 1000 CPU seconds was used. Figure 2 shows the CPU time percentile graphs of the Logics Workbench on random SNF formulae for $n = 5$ and $n = 12$. We observe that all the random SNF formulae for $n = 5$ are very easy for the Logics Workbench. Quite the opposite is true for $n = 12$. For random SNF formulae generated for l between 10 and 19 the median CPU time exceeds 100 seconds. However, the three most important characteristics are the following: (i) a steep incline in the median CPU time from $l/n = 0.75$ to $l/n = 0.83$, (ii) a peak in the median CPU time at $l/n = 1.17$ and a gentle decline of the median CPU time for $1.17 < l/n \leq 4.75$, and (iii) a steep decline in the median CPU time for $l > 4.75$. Thus, the median CPU time peaks on a collection of formulae which are all satisfiable and there seems to be no apparent connection between the phase transition in the satisfiability of random SNF formulae and the time needed by the Logics Workbench to decide these formulae. It should be mentioned that the Logics Workbench fails a number of random SNF formulae due to memory allocation problems (not the time-limit). In fact, for $n > 12$ this becomes a major problem and for a considerable number of values for l the majority of random SNF formulae can no longer be decided.

To explain the characteristics of the performance of the Logics Workbench, let us take a closer look at random SNF formulae and how a tableau calculus constructs models for them. Let φ be a random SNF formula over n propositional variables and let $C_1, \ldots, C_l$ be the global clauses in φ. First note, that $C_i = \Box(\bigcirc L_1^i \vee \bigcirc L_2^i \vee \bigcirc L_3^i)$ is equivalent to $\bigcirc\Box(L_1^i \vee L_2^i \vee L_3^i)$. Thus, C_i puts no constraint on the state s_0, in a potential model $\mathcal{I}$ for φ, and enforces the constraint $L_1^i \vee L_2^i \vee L_3^i = \text{next}(C_i)$ on the remaining states s_j, $j > 0$, of $\mathcal{I}$. With increasing l, the number of distinct valuations which satisfy $\text{next}(C_i)$ for all i, $1 \leq i \leq l$, continuously decreases. For $n = 12$ and $l/n > 4.75$, no such satisfying valuation exists for more than 50% of the random SNF formulae. This

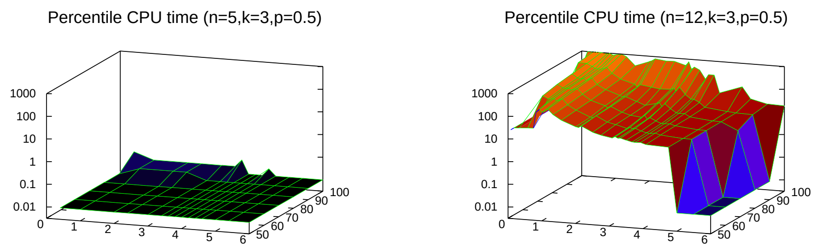

Fig. 2. CPU time percentiles of the Logics Workbench

is easy to detect with the Logics Workbench and leads to the good performance on these random SNF formulae. For all ratios l/n smaller than 4.75 the sometime clauses in φ have to be taken into account. Consider an arbitrary sometime clause $\Box(\neg p_i \vee \Diamond p_{i+1})$. This clause imposes the constraint $\neg p_i \vee \Diamond p_{i+1}$ on all states in $\mathcal{I}$. To satisfy this disjunction, for s_0, the Logics Workbench will always choose to make $\neg p_i$ true, but for all other states this will only be possible if p_i is not already true due to the constraints imposed by the global clauses. Thus, as long as there is a satisfying valuation which makes all $\mathrm{next}(C_i)$ true and also all p_i false, $\mathcal{I}$ can be constructed easily. For $n = 12$ we find that for $l \leq 0.33$ this is true for more than 50% of the formulae and for $l \leq 0.58$ it is still true for more than 40% of the formulae. The percentile graph for $n = 12$ shows quite a good correlation between the sudden rise in the CPU time consumption and the decline of the percentage of formulae for which models can be found in the way described above. To explain the behaviour of the Logics Workbench for $n = 12$ and ratios $0.83 \leq l/n \leq 4.75$ it is useful to take Theorem 2 into account.

The proof of Theorem 2 tells us something about the structure of PLTL-models for random SNF formulae. There are basically two possibilities. Either the model consists of an infinite sequence of states whose identical valuations satisfy the formula ψ_0 defined in the proof of Theorem 2, or the model consists of an infinite repetition of the sequence of states $s_1, \ldots, s_n$ where each s_i satisfies the formulae ψ_i defined above. The first possibility provides an explanation of the good behaviour of the Logics Workbench for ratios $l/n \leq 0.58$. However, for ratios $0.83 \leq l/n \leq 4.75$ almost all models will be structured according to the second possibility. Since the Logics Workbench is not aware that the states $s_1, \ldots, s_n$ have to satisfy particular valuations, it searches through up to n^{2^n} combinations of possible valuations, until a model has been found. Since the global clauses put ever stronger constraints on satisfying constraints with increasing ratio l/n, the search space is getting continuously smaller. Thus, the continuous decrease in the median CPU time observed after the peak at $l = 1.17$.

We now turn to TRP, an experimental implementation of the temporal resolution calculus for PLTL. Figure 3 shows the percentile CPU time graphs of TRP on random SNF formulae for $n = 5$ and $n = 12$. While the performance of TRP for $n = 5$ is not significantly different to that of Logics Workbench, there

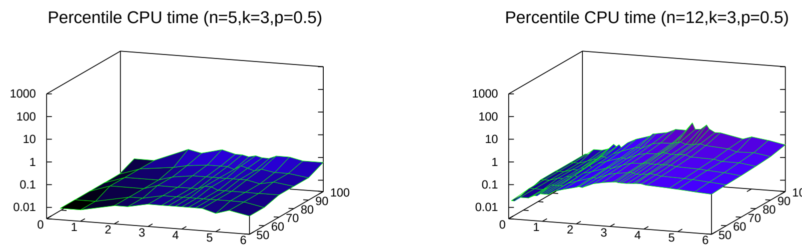

Fig. 3. CPU time percentiles of TRP

is an obvious difference for $n = 12$. There is a slow and steady increase in the median CPU time for every l/n between 0 and 4.75 followed by a small decrease for $l/n > 4.75$. Except for $l/n > 4.75$, TRP is considerably faster than the Logics Workbench.

To explain the behaviour of TRP we take a closer look at the temporal resolution calculus. Besides the standard resolution rule for propositional clause logic (which can be applied to initial clauses), it contains two inference rules, *step resolution* and *temporal resolution*. While the step resolution rule resolves two global clauses C_1 and C_2 on complementary literals in $\text{next}(C_1)$ and $\text{next}(C_2)$, the temporal resolution rule resolves a set of global clauses with a sometime clause to derive a set of initial and global clauses. Determining a set of global clauses which can successfully be resolved with a sometime clause is a non-trivial problem, and our implementation follows the approach described in [3]. In TRP propositional and step resolution take precedence over temporal resolution. That is, TRP will only try to apply the temporal resolution rule when no further applications of the other two rules are possible. As pointed out above, for $n = 12$ and $l/n > 4.75$, the conjunction of global clauses in a random SNF formula is unsatisfiable. Thus, TRP will derive a contradiction for these formulae without trying to apply the temporal resolution rule. For all random SNF formulae where the conjunction of their global clauses is satisfiable, we have to try to apply the temporal resolution rule to each of the n sometime clauses. Now, if the random SNF formula is unsatisfiable, we will be able to successfully apply the temporal resolution rule to at least one of the sometime clauses. Already after the first successful application, the additional clauses derived will be sufficient to refute the random SNF formula. On the other hand, if the random SNF formula under consideration is satisfiable, then we will either fail to apply the temporal resolution rule to any of the n sometime clauses or the clauses we derive will not lead to the derivation of a contradiction. When no further clauses can be derived, then TRP concludes that the random SNF formula is satisfiable.

The slow and steady increase in the time needed to decide the random SNF formulae for $l/n \geq 4.75$ is due to the increase in the number of global clauses in the formulae which we need to saturate under the three inference rules. For $n = 12$ and $l/n < 0.83$ the number of resolvents generated is below 10, reaching

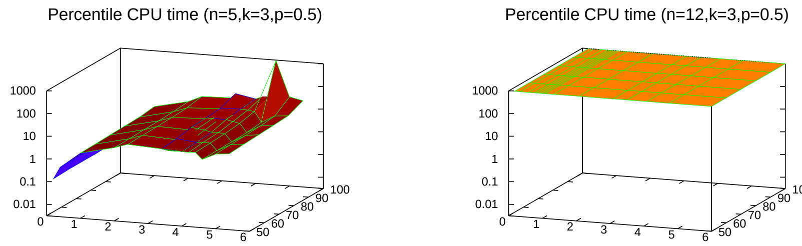

Fig. 4. CPU time percentiles of TLSMV

about 1000 resolvents for $l/n = 4.75$, and dropping to below 300 resolvent for $l/n = 6$.

Finally, we have done some preliminary experiments with TLSMV on random SNF formulae. Figure 4 shows the percentile CPU time graphs of TLSMV on random SNF formulae for $n = 5$ and $n = 12$.

For $n = 5$ the shape of the percentile graph for TLSMV is similar to that for TRP, although TLSMV is about 200 times slower than TRP. If this difference in performance would be a constant factor between the two provers, we would expect that TLSMV is able to decide most of the SNF formulae for $n = 12$. However, it fails completely.

Another interesting property of random SNF formulae is that they can be embedded into PDL in a satisfiability equivalence preserving way. Let τ' be the following translation from PLTL (without $\mathcal{U}$ and $\mathcal{W}$) to PDL and let $\tau'_{\mathsf{PDL}}(\varphi) = [a^\star]\langle a\rangle\top \wedge \tau'(\varphi)$.

$$
\begin{aligned}
\tau'(p) &= p & \tau'(\bigcirc\varphi) &= [a]\tau'(\varphi)\\
\tau'(\neg\varphi) &= \neg\tau'(\varphi) & \tau'(\Box\varphi) &= [a^\star]\tau'(\varphi)\\
\tau'(\varphi\vee\psi) &= \tau'(\varphi)\vee\tau'(\psi) & \tau'(\Diamond\varphi) &= \langle a^\star\rangle\tau'(\varphi)
\end{aligned}
$$

Theorem 3. *Let φ be a random SNF formula. Then φ is* PLTL*-satisfiable iff $\tau'_{\mathsf{PDL}}(\varphi)$ is* PDL*-satisfiable.*

Proof (Sketch). Constructing a PDL-model for $\tau'_{\mathsf{PDL}}(\varphi)$ from a given PLTL-model for φ is straightforward. The task of constructing a PLTL-model for φ from a PDL-model $\mathcal{I}$ for $\tau'_{\mathsf{PDL}}(\varphi)$ seems less straightforward, since the PDL-model $\mathcal{I}$ might contain more than one r_a-successor for a world w in $\mathcal{I}$. However, we can show that in this case we can arbitrarily choose one r_a-successor for incorporation into the PLTL-model and can ignore the remaining ones.

Thus, random SNF formulae allow a 'cross-logic' comparison of decision procedures for PLTL, PDL, and DPDL, which was one of our design goals.

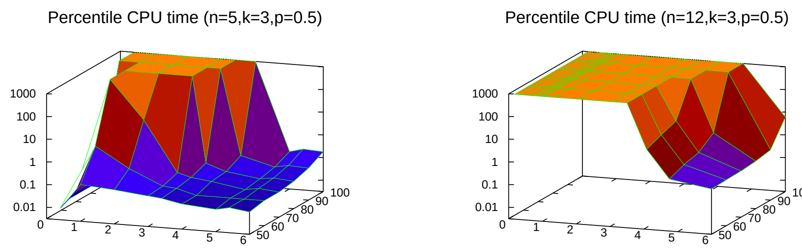

Fig. 5. CPU time percentiles of DLP

We have done some preliminary[1] experiments with DLP (version 3.3) [8] on translated random SNF formulae. Figure 5 shows the CPU time percentile graphs, respectively, on random SNF formulae for $n = 5$ and $n = 12$. The most obvious observation is that the translated random SNF formulae are more difficult for DLP than the original formulae were for the Logics Workbench. This might be due to the following problem: Suppose that in a particular world w two formulae $\langle a^\star\rangle p_1$ and $\langle a^\star\rangle p_2$ have to be true, but p_1 and p_2 are false at w. A tableaux-based algorithm will conclude that the only way to make $\langle a^\star\rangle p_1$ and $\langle a^\star\rangle p_2$ true is to make $\langle a\rangle\langle a^\star\rangle p_1$ and $\langle a\rangle\langle a^\star\rangle p_2$ true. To investigate whether this is possible, the algorithm will create and consider two direct r_a-successor worlds w_1 and w_2 of w, where $\langle a^\star\rangle p_1$ and $\langle a^\star\rangle p_2$ have to be true in w_1 and w_2, respectively. However, due to the particular structure of the sometime clauses in random SNF formulae, it turns out that in some r_a-successor of w_1 or w_1 itself also $\langle a^\star\rangle p_2$ has to be true. The investigation of w_2 and its successors is wasted effort which can exponentially worsen the performance of the algorithm.

A question which cannot be answered with certainty based on the experimental results is whether DLP shows, in principle, the same behaviour as the Logics Workbench. The median CPU time percentile for $n = 5$ seems to suggest a peak at the ratio $l/n = 1$ and a steady decrease in the median CPU time from this point onwards. Unfortunately, already for $n = 6$ we observed a drastic increase in the difficulty of random SNF formulae for DLP leading to a similar graph as for $n = 12$ which makes it difficult to come to any conclusions. However, two interesting observations can be made. Note that for $n = 12$ the ratio l/n there is a significant decrease in the median CPU time percentile for $l/n = 4$ and a further decrease for $l/n = 4.75$. The second decrease coincides with the ratio for which for more than 50% of the formulae we are no longer able to find a satisfying valuation for their global clauses. The second observation is that for

[1] During the analysis of the experiments we found that the results of DLP and the Logics Workbench were not always identical. The discrepancies disappear if caching is disabled in DLP, which, however, dramatically worsens the performance of DLP. Since the number of discrepancies is small, we expect that the behaviour of a corrected version of DLP will still be similar as reported in this paper.

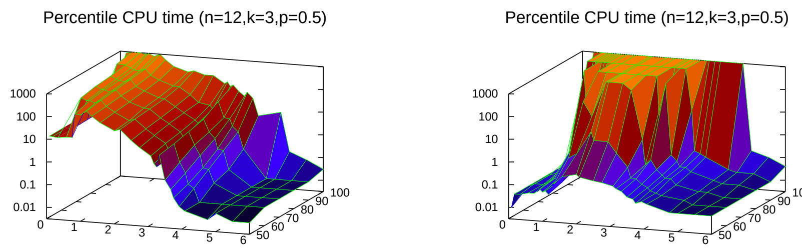

Fig. 6. CPU time percentiles of the Logics Workbench and DLP

$n = 5$ the behaviour of DLP is much less uniform than the behaviour of the Logics Workbench as can be seen in the percentile graphs.

Finally, we turn to the question of possible improvements of the theorem provers tested. For example, we have mentioned in the discussion of the results for the Logics Workbench that $\Box(\bigcirc L_1^i \vee \bigcirc L_2^i \vee \bigcirc L_3^i)$ is equivalent to $\bigcirc\Box(L_1^i \vee L_2^i \vee L_3^i)$. Figure 6 show the percentile graphs of the Logics Workbench and DLP if we apply this equivalence (from left to right) to random SNF formulae for $n = 12$ before they are given to these provers. Both provers show an significant improvement in their performance. In the case of the Logics Workbench one might have expected that this equivalence is exploited, while in the case of DLP this cannot be expected (since the equivalence does not hold after the translation).

References

1. G. De Giacomo and F. Massacci. Tableaux and algorithms for propositional dynamic logic with converse. In *Proc. CADE'96*, LNAI 1104, pages 613–628, 1996.
2. C. Dixon. Search strategies for resolution in temporal logics. In *Proc. CADE-13*, LNAI 1104, pages 673–687. Springer, 1996.
3. C. Dixon. Using Otter for temporal resolution. In *Advances in Temporal Logic*, volume 16 of *Applied Logic Series*, pages 149–166. Kluwer, 2000.
4. M. Fisher. A resolution method for temporal logic. In *Proc. IJCAI'91*, pages 99–104. Morgan Kaufman, 1991.
5. M. Fisher, C. Dixon, and M. Peim. Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, 2(1), 2001.
6. G. D. Gough. Decision procedures for temporal logic. Technical Report UMCS-89-10-1, Department of Computer Science, University of Manchester, UK, 1989.
7. A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. The Logics Workbench LWB: A snapshot. *Euromath Bulletin*, 2(1):177–186, 1996.
8. P. Patel-Schneider. System description: DLP. In *Proc. CADE-17*, LNAI 1831, pages 297–301. Springer, 2000.
9. S. Schwendimann. *Aspects of Computational Logic*. PhD thesis, Universität Bern, Switzerland, 1998.
10. P. Wolper. The tableau method for temporal logic: An overview. *Logique et Analyse*, 28:119–136, 1985.