TeMP: A Temporal Monodic Prover*

Ullrich Hustadt[1], Boris Konev[1**], Alexandre Riazanov[2], and Andrei Voronkov[2]

[1] Department of Computer Science, University of Liverpool, UK
{U.Hustadt, B.Konev}@csc.liv.ac.uk
[2] Department of Computer Science, University of Manchester, UK
{riazanov, voronkov}@cs.man.ac.uk

Abstract. We present **TeMP**—the first experimental system for testing validity of *monodic* temporal logic formulae. The prover implements *fine-grained temporal resolution*. The core operations required by the procedure are performed by an efficient resolution-based prover for classical first-order logic.

1 Monodic First-Order Temporal Logic

First-Order Temporal Logic, FOTL, is an extension of classical first-order logic by temporal operators for a discrete linear model of time (isomorphic to $\mathbb{N}$, that is, the most commonly used model of time). Formulae of this logic are interpreted over structures that associate with each element n of $\mathbb{N}$, representing a moment in time, a first-order structure (D_n, I_n) with its own non-empty domain D_n. In this paper we make the *expanding domain assumption*, that is, $D_n \subseteq D_m$ if $n < m$. The set of valid formulae of this logic is not recursively enumerable. However, the set of valid *monodic* formulae is known to be finitely axiomatisable [13].

A formula ϕ in a FOTL language *without equality and function symbols* (constants are allowed) is called *monodic* if any subformulae of ϕ of the form $\bigcirc\psi$, $\Box\psi$, $\Diamond\psi$, $\psi_1 \mathsf{U} \psi_2$ or $\psi_1 \mathsf{W} \psi_2$ contains at most one free variable. For example, the formulae $\forall x \Box \exists y P(x,y)$ and $\forall x \Box P(x,c)$ are monodic, while $\forall x,y(P(x,y) \Rightarrow \Box P(x,y))$ is not monodic.

The monodic fragment has a wide range of novel applications, for example in spatio-temporal logics [14, 5] and temporal description logics [1].

In this abstract we describe **TeMP**, the first automatic theorem prover for the monodic fragment of FOTL.

2 Monodic fine-grained temporal resolution

Our temporal prover is based on *fine-grained temporal resolution* [9] which we briefly describe in this section. Every monodic temporal formula can be translated in a satisfiability equivalence preserving way into a clausal form. The calculus operates on four kinds of temporal clauses, called *initial*, *universal*, *step*, and *eventuality* clauses. Essentially, initial clauses hold only in the initial moment in time, all other kinds of clause hold in every moment in time. Initial and universal are ordinary first-order clauses, containing no temporal operators. *Step* clauses in the clausal form of monodic temporal

* Work supported by EPSRC grant GR/L87491.

** On leave from Steklov Institute of Mathematics at St.Petersburg formulae are of the form $p \Rightarrow \bigcirc q$, where p and q are propositions, or of the form $P(x) \Rightarrow \bigcirc Q(x)$, where P and Q are unary predicate symbols and x a variable. During a derivation more general *step* clauses can be derived, which are of the form $C \Rightarrow \bigcirc D$, where C is a *conjunction* of propositions, atoms of the form $P(x)$ and ground formulae of the form $P(c)$, where P is a unary predicate symbol and c is a constant such that c occurs in the input formula, D is a *disjunction* of arbitrary literals, such that C and D have at most one free variable in common. The *eventuality* clauses are of the form $\Diamond L(x)$, where $L(x)$ is a literal having at most one free variable.

Monodic fine-grained temporal resolution consists of the *eventuality resolution rule*:

$$\frac{\forall x(\mathcal{A}_1(x) \Rightarrow \bigcirc(\mathcal{B}_1(x))) \quad \dots \quad \forall x(\mathcal{A}_n(x) \Rightarrow \bigcirc(\mathcal{B}_n(x))) \qquad \Diamond L(x)}{\forall x \bigwedge_{i=1}^{n} \neg \mathcal{A}_i(x)} \ (\Diamond^{\mathcal{U}}_{res}),$$

where $\mathcal{U}$ is the current set of all universal clauses, $\forall x(\mathcal{A}_i(x) \Rightarrow \bigcirc \mathcal{B}_i(x))$ are complex combinations of step clauses, called *full merged step clauses* [9], such that for all $i \in \{1, \dots, n\}$, the *loop* side conditions $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \neg L(x))$ and $\forall x(\mathcal{U} \wedge \mathcal{B}_i(x) \Rightarrow \bigvee_{j=1}^{n}(\mathcal{A}_j(x)))$ are both valid; and the following five rules of *fine-grained step resolution*

1. *First-order resolution between two universal clauses and factoring on a universal clause.* The result is a universal clause.
2. *First-order resolution between an initial and a universal clause, between two initial clauses, and factoring on an initial clause*. The result is again an initial clause.
3. *Fine-grained (restricted) step resolution*.

$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee L) \quad C_2 \Rightarrow \bigcirc(D_2 \vee \neg M)}{(C_1 \wedge C_2)\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma} \qquad \frac{C_1 \Rightarrow \bigcirc(D_1 \vee L) \quad D_2 \vee \neg M}{C_1\sigma \Rightarrow \bigcirc(D_1 \vee D_2)\sigma}$$

4. *(Step) factoring*.

$$\frac{C_1 \Rightarrow \bigcirc(D_1 \vee L \vee M)}{C_1\sigma \Rightarrow \bigcirc(D_1 \vee L)\sigma} \qquad \frac{(C_1 \wedge L \wedge M) \Rightarrow \bigcirc D_1}{(C_1 \wedge L)\sigma \Rightarrow \bigcirc D_1\sigma}$$

5. *Clause conversion*.
   A step clause of the form $C \Rightarrow \bigcirc \textbf{false}$ is rewritten into the *universal clause* $\neg C$.

In rules 1 to 5, we assume that different premises and conclusions of the deduction rules have no variables in common; variables may be renamed if necessary. In rules 3 and 4, σ is a most general unifier of the literals L and M *such that* σ *does not map variables from* C_1 *or* C_2 *into a constant or a functional term*.

The input formula is unsatisfiable over expanding domains if and only if fine-grained temporal resolution derives the empty clause (see [9], Theorem 8).

3 Implementation

The deduction rules of fine-grained step resolution are close enough to classical first-order resolution to allow us to use first-order resolution provers to provide an implementation of our calculus.

Let **S** be a temporal problem in clausal form. For every k-ary predicate, P, occurring in **S**, we introduce new $(k+1)$-ary predicate $\widetilde{P}$. We will also use the constant 0 (representing the initial moment in time), and unary function symbols s (representing the successor function on time) and h, which we assume not to occur in $\mathbf{S}$. Let ϕ be a first-order formula in the vocabulary of $\mathbf{S}$. We denote by $[\phi]^T$ the result of replacing all occurrences of predicates in ϕ by their "tilded" counterparts with T as the first argument (e.g. $P(x,y)$ is replaced with $\widetilde{P}(T,x,y)$). The term T will either be the constant 0 or the variable t (intuitively, t represents a moment in time). The variable t is assumed to be universally quantified.

Now, in order to realise fine-grained step resolution by means of classical first-order resolution, we define a set of first-order clauses $\mathbf{FO}(\mathbf{S})$ as follows.

– For every initial clause C from $\mathbf{S}$, the clause $[C]^0$ is in $\mathbf{FO}(\mathbf{S})$.
– For every universal clause D from $\mathbf{S}$, the clause $[D]^t$ is in $\mathbf{FO}(\mathbf{S})$.
– For every step clause $p \Rightarrow \bigcirc q$ from $\mathbf{S}$, the clause $\neg\widetilde{p}(t) \vee \widetilde{q}(t)$ is in $\mathbf{FO}(\mathbf{S})$, and for every step clause $P(x) \Rightarrow \bigcirc Q(x)$, the clause $\neg P(t,x) \vee Q(s(t),h(x))$ is in $\mathbf{FO}(\mathbf{S})$.

It is a straightforward routine to check that fine-grained step resolution on $\mathbf{S}$, including (implicitly) the clause conversion rule, is modelled by classical ordered first-order resolution with selection (see, e.g., [2]) on $\mathbf{FO}(\mathbf{S})$ together with an additional requirement: if a clause contains a *next-state* literal, i.e., a literal whose first argument starts with the function symbol s, this clause can only be resolved on next-state literals. This requirement can be enforced by an appropriate literal selection discipline. Note that standard redundancy deletion mechanisms, such as subsumption and tautology deletion, are also compatible with fine-grained step resolution (for details see [9]).

As for the eventuality resolution rule, note that finding full merged clauses which satisfy the side conditions of the eventuality resolution rule is a non-trivial problem [9]. We find such merged clauses by means of a search algorithm presented in [9] which is again based on step resolution. Hence, the performance of the step resolution inference engine is critical.

In our implementation, we extended the propositional temporal prover, **TRP**++ [6], to deal with monodic formulae. The main procedure of our implementation of this calculus consists of a loop where at each iteration (i) the set of temporal clauses is saturated under application of the step resolution rules, and (ii) then for every eventuality clause in the clause set, an attempt is made to find a set of premises for an application of the eventuality resolution rule. If we find such a set, the set of clauses representing the conclusion of the application is added to the current set of clauses. The main loop terminates if the empty clause is derived, indicating that the initial set of clauses is unsatisfiable, or if no new clauses have been derived during the last iteration of the main loop, which in the absence of the empty clause indicates that the initial set of clauses is satisfiable.

The task of saturating clause sets with classical resolution simulating step resolution is delegated to the **Vampire** kernel [11], which is linked to the whole system as a C++ library. Minor adjustments have been made in the functionality of **Vampire** to accommodate step resolution: a special mode for literal selection has been introduced such that in a clause containing a next-state literal only next-state literals can be selected. At the moment, the result of a previous saturation step, augmented with the result of an eventuality resolution application, is resubmitted to the **Vampire** kernel, although no inferences are performed between the clauses from the already saturated part. This is only a temporary solution, and in the future **Vampire** will support incremental input in order to reduce communication overhead.

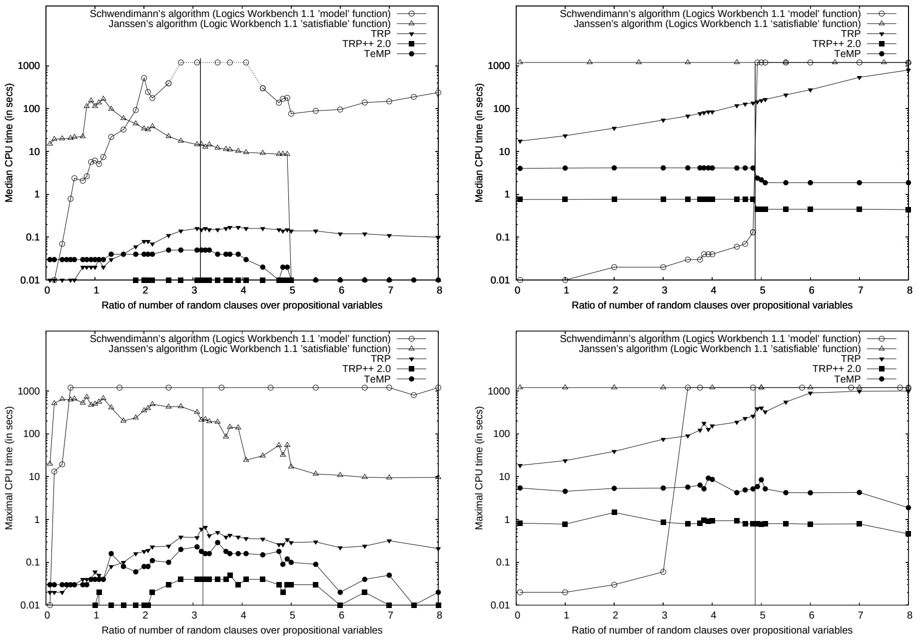

Fig. 1. Performance of the systems on $\mathcal{C}^1_{ran}$ (left) and $\mathcal{C}^2_{ran}$ (right)

4 Performance evaluation

It is difficult to evaluate the performance of **TeMP** for two reasons. First, there are no established monodic benchmark sets. Second, there is no other monodic temporal provers to compare with.

However, we find it worthwhile to compare the performance of **TeMP** to that of **TRP**++ 2.0 [6] on *propositional* temporal logic (PLTL) formulae. Both provers perform essentially the same inference steps on such formulae, since the rules of fine-grained temporal resolution presented in Section 2 coincide with those of propositional temporal resolution [3] on propositional temporal logic formulae.

Besides **TeMP** and **TRP**++ 2.0 we have also included two tableau-based procedures for PLTL implemented in the Logics Workbench 1.1, described respectively in [8] and [12], and **TRP**, a prototype implementation of temporal resolution in SICStus Prolog by the first author.

We have compared the systems on two classes of semi-randomly generated PLTL-formulae, called $\mathcal{C}^1_{ran}$ and $\mathcal{C}^2_{ran}$, introduced in [7], for parameters $n = 12$, $k = 3$, and $p = 0.5$. The tests were performed on a PC with a 1.3GHz AMD Athlon processor, 512MB main memory, and 1GB virtual memory running Red Hat Linux 7.1. For each individual satisfiability test of a formula a time-limit of 1000 CPU seconds was used.

The left- and right-hand sides of Figure 1 depict the behaviour of the systems on $\mathcal{C}^1_{ran}$ and on $\mathcal{C}^2_{ran}$, respectively. All the graphs contain a vertical line. On the left of the line most formulae are satisfiable, on the right most formulae are unsatisfiable.

The upper part of the figure shows the resulting graphs for the median CPU time consumption of each of the systems, while the lower part shows the graphs for the maximal CPU time consumption. In all performance graphs, a point for a system above the 1000 CPU second mark indicates that the median or maximal CPU time required by the system exceeded the imposed time limit.

We can see from these graphs that **TeMP** is about an order of magnitude slower than **TRP**++ 2.0, but still faster than the prototypical system **TRP**. This can be explained by high overheads in communications with **Vampire**. **TeMP** is also faster than the two tableau-based procedures on $\mathcal{C}^1_{ran}$ and is only outperformed by the procedure of [12] on satisfiable formulae in $\mathcal{C}^2_{ran}$. In our opinion, the results shows the strength of **TeMP**, since it is not specialised for propositional reasoning.

We are aware of new *tableau-based* systems [10] for monodic temporal logic being under development. When these systems are available, we will be able to perform a systematic comparison with **TeMP**. We also intend to look at more realistic formulae coming from verification problems [4], instead of randomly generated formulae.

References

1. A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning over conceptual schemas and queries. In *Proc. JELIA'02*, volume 2424 of *LNCS*, pages 98–110. Springer, 2002.
2. L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 19–99. Elsevier, 2001.
3. M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Transactions on Computational Logic*, 2(1):12–56, 2001.
4. M. Fisher and A. Lisitsa. Temporal verification of monodic abstract state machines. Technical Report ULCS-03-011, Department of Computer Science, University of Liverpool, 2003.
5. D. Gabelaia, R. Kontchakov, A. Kurucz, F. Wolter, and M. Zakharyaschev. On the computational complexity of spatio-temporal logics. In *Proc. FLAIRS 2003*, pages 460–464. AAAI Press, 2003.
6. U. Hustadt and B. Konev. TRP++ 2.0: A temporal resolution prover. In *Proc. CADE-19*, volume 2741 of *LNAI*, pages 274–278. Springer, 2003.
7. U. Hustadt and R. A. Schmidt. Scientific benchmarking with temporal logic decision procedures. In *Proc. KR2002*, pages 533–544. Morgan Kaufmann, 2002.
8. G. Janssen. *Logics for Digital Circuit Verification: Theory, Algorithms, and Applications*. PhD thesis, Eindhoven University of Technology, The Netherlands, 1999.
9. B. Konev, A. Degtyarev, C. Dixon, M. Fisher, and U. Hustadt. Mechanising first-order temporal resolution. Technical Report ULCS-03-023, University of Liverpool, Department of Computer Science, 2003. `http://www.csc.liv.ac.uk/research/`.
10. R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalising tableaux. *Studia Logica*, 76(1):91–134, 2004.
11. A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.
12. S. Schwendimann. *Aspects of Computational Logic*. PhD thesis, Universität Bern, Switzerland, 1998.
13. F. Wolter and M. Zakharyaschev. Axiomatizing the monodic fragment of first-order temporal logic. *Annals of Pure and Applied logic*, 118:133–145, 2002.
14. F. Wolter and M. Zakharyaschev. Qualitative spatio-temporal representation and reasoning: a computational perspective. In *Exploring Artificial Intelligence in the New Millenium*, pages 175–216. Morgan Kaufmann, 2002.