Propositional Temporal Proving with Reductions to a SAT Problem

Richard Williams and Boris Konev

Department of Computer Science, University of Liverpool, Liverpool L69 7ZF
{R.M.Williams1, Konev}@liv.ac.uk

Abstract. We present a new approach to reasoning in propositional linear-time temporal logic (PLTL). The method is based on the simplified temporal resolution calculus. We prove that the search for premises to apply the rules of simplified temporal resolution can be re-formulated as a search for minimal unsatisfiable subsets (MUS) in a set of classical propositional clauses. This reformulation reduces a large proportion of PLTL reasoning to classical propositional logic facilitating the use of modern tools. We describe an implementation of the method using the CAMUS system for MUS computation and present an in-depth comparison of the performance of the new solver against a clausal temporal resolution prover.

1 Introduction

Propositional Linear-time Temporal Logic (PLTL) is an extension of classical propositional logic with operators that deal with time. PLTL, and its extensions, have been used in various areas of computer science, for example, for the specification of distributed and concurrent systems and verification of their properties through temporal reasoning [20], for synthesis of programs from temporal specifications [23], in temporal databases [27] and for knowledge representation and reasoning [15]. PLTL is notable for its widespread use as a specification language in software and hardware verification via model checking [6].

In recent years, we witness a renewed interest in PLTL theorem proving. Among other reasons, it can be explained by the fact that PLTL specifications, used in verification of software and hardware systems, often go far beyond simple safety and liveness conditions. In fact, temporal specifications became so complicated that a need arises for an automated check if they are (un)satisfiable. Indeed, it does not make sense to check whether a PLTL formula is true in a model if the formula is unsatisfiable or valid [12, 25, 24].

Satisfiability of PLTL formulae can be established with a number of techniques including automata-based approaches [28], tableau methods [30] and clausal temporal resolution [11]. Clausal temporal resolution has been successfully implemented [18, 17] and shown to perform well in practice [17, 25].

Clausal temporal resolution is a machine-oriented calculus that operates on temporal formulae in a clausal form called SNF and uses a small number of resolution rules. In a nutshell, clausal temporal resolution propagates conflicts 'backward in time' until a contradiction is derived in the initial state [11]. For example, consider a conjunction of the following three temporal clauses

$$1\colon a \Rightarrow \bigcirc(x \vee y) \quad 2\colon b \Rightarrow \bigcirc\neg x \quad 3\colon c \Rightarrow \bigcirc\neg y,$$

where $\bigcirc$ denotes 'at the next moment of time'. Similar to classical resolution, clausal temporal resolution derives a new clause 4: $a \wedge b \Rightarrow \bigcirc y$ [`from 1 and 2`]. Then the derived clause 4 can be combined with clause 3 to derive 5: $a \wedge b \wedge c \Rightarrow \bigcirc\mathbf{false}$ [`from 4 and 3`]. This latter clause can be rewritten as 6: $\neg a \vee \neg b \vee \neg c$ [`from 5`]. Indeed, if it is not the case that at least one of a, b or c is false, we inevitably get a contradiction at the next moment of time.

Simplified temporal resolution introduced in [7] derives clause 6 in one go by noticing that the (pure classical, that is, containing no temporal operators) conjunction of the right-hand sides of the given clauses, $(x \vee y) \wedge \neg x \wedge \neg y$ is unsatisfiable. Thus, an application of the temporal resolution rule can be characterised in an abstract way as a multi-premise rule with a purely classical side condition.

The biggest challenge in implementing the simplified calculus is that the abstract characterisation of the inference rules gives no hint on which temporal clauses need to be combined. A straightforward implementation of simplified temporal resolution enumerates all combinations of temporal clauses in order to find those satisfying the classical side conditions. The resulting procedure is best-case exponential. In fact, simplified temporal resolution was never intended to be implemented. The calculus has primarily been introduced to provide a cleaner separation between temporal and classical reasoning, to simplify the proof of completeness and to explore variations of the clausal normal form [7].

In this paper we present a new approach to PLTL reasoning based on simplified temporal resolution, which tackles the challenge of determining which clauses need to be combined by reducing it to the propositional Minimal Unsatisfiable Subset (MUS) problem. A set of propositional clauses is an MUS if it is both usatisfiable and any proper subset is satisfiable. We prove that when searching for temporal clauses to combine for simplified temporal reasoning, it suffices to consider those whose right-hand side (together with some universal clauses) forms an MUS. This reduces a large proportion of PLTL reasoning to classical propositional logic. We report on a rigorous experimental evaluation of our prototype implementation of the calculus, which shows that this simple and elegant idea works well in practice.

2 Preliminaries

The set of PLTL formulae is the smallest set containing the set of (atomic) propositions *Prop* and such that if ϕ and ψ are in PLTL formulae, then so are **true**, $\neg\phi$, $\phi \vee \psi$, $\bigcirc\phi$ ('ϕ is true in the next moment'), and $\phi \mathsf{U} \psi$ ('ϕ is true until ψ becomes true'). As usual, we introduce other Boolean and temporal operators ($\Box$ 'always in the future', $\Diamond$ 'sometime in the future' and W 'unless') as abbreviations: **false** $= \neg$**true**, $\phi \wedge \psi = \neg(\neg\phi \vee \neg\psi)$, $\phi \Rightarrow \psi = \neg\phi \vee \psi$, $\phi \Leftrightarrow \psi = (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$, $\Diamond\phi =$ **true** $\mathsf{U}\ \phi$, $\Box\phi = \neg\Diamond\neg\phi$ and $\phi\ \mathsf{W}\ \psi = (\phi\ \mathsf{U}\ \psi) \vee (\Box\phi)$.

A PLTL formula that does not contain any temporal operators is called a (classical) *propositional formula*. A *literal* is a proposition or a negation of a proposition. A *clause* is disjunction of literals. A propositional CNF formula is a conjunction of clauses. We do not make a distinction between a propositional CNF formula ϕ and the set of clauses S such that $\phi = \bigwedge_{C \in S} C$.

A model for a PLTL formula ϕ can be characterised as a sequence of *states* of the form $\sigma = s_0, s_1, s_2, \ldots$, where each state s_i is a set of propositions that are satisfied at the i^{th} moment in time. We call every such sequence of states an interpretation. We define the relation $(\sigma, i) \models \phi$ (at time instance i, interpretation σ satisfies PLTL formula ϕ) by induction on the structure of the formula as follows:

$$\begin{array}{lll}
(\sigma, i) \models p & \text{iff} & p \in s_i \quad [\text{for } p \in \mathit{Prop} \text{ and } \sigma = s_0, s_1, \ldots] \\
(\sigma, i) \models \bigcirc\psi & \text{iff} & (\sigma, i+1) \models \psi \\
(\sigma, i) \models \phi\ \mathsf{U}\ \psi & \text{iff} & \text{iff } \exists k \in \mathbb{N}.\ k \geq i \text{ and } (\sigma, k) \models \psi \text{ and} \\
 & & \forall j \in \mathbb{N}, \text{ if } i \leq j < k \text{ then } (\sigma, j) \models \phi
\end{array}$$

We say that a formula ϕ is *satisfiable* if, and only if, there exists an interpretation σ such that $(\sigma, 0) \models \phi$. We also say that σ is a model of ϕ in this case. A formula ϕ is *valid* if, and only if, it is satisfied in every possible interpretation, i.e. for each σ, $(\sigma, 0) \models \phi$. A formula ϕ is unsatisfiable if, and only if, it is not satisfiable.

Simplified temporal resolution introduced in [7] operates on temporal problems in divided separated normal form (DSNF). A DSNF problem is a quadruple $\langle \mathcal{I}, \mathcal{U}, \mathcal{S}, \mathcal{E} \rangle$, where

- $\mathcal{I}$ (the initial part) and $\mathcal{U}$ (the universal part) are sets of propositional clauses;
- $\mathcal{S}$ (the step part) is a set of *step clauses* of the form

$$P \Rightarrow \bigcirc Q,$$

  where P is a conjunction of literals and Q is a disjunction of literals;
- and $\mathcal{E}$ (the eventuality part) is a set of *eventualities* of the form $\Diamond l$, where l is a literal.

The intended meaning of a DSNF problem is given by

$$\mathcal{I} \wedge \Box\mathcal{U} \wedge \Box\mathcal{S} \wedge \Box\mathcal{E}.$$

When we talk about particular properties of temporal problems (e.g., satisfiability, validity, logical consequences etc) we mean properties of the associated formula. (As above, we do not make a distinction between a finite set of formulae and a conjunction of formulae in this set.)

Arbitrary PLTL-formulae can be transformed into satisfiability equivalent DSNF problems using a renaming technique replacing non-atomic subformulae with new propositions and replacing all occurrences of the U ('until'), $\Box$ ('always') and W ('unless') operators with their fixpoint definitions. The size of the resulting temporal problem in DSNF is at most linear in the size of the given formula [11, 7, 8]. We illustrate DSNF transformation with an example.

Example 1. Consider temporal formula $\phi = \Diamond\Box a \wedge \Diamond\Box\neg a$. First, we satisfiability equivalently rewrite it as $x_1 \wedge \Box(x_1 \Rightarrow \Diamond\Box a) \wedge \Box(x_1 \Rightarrow \Diamond\Box\neg a)$, where x_1 a fresh proposition. Then we rename the occurrences of the always operator and then 'unwind' the always operator using its fixpoint definition to give

$$\begin{array}{r}x_1 \wedge \Box(x_1 \Rightarrow \Diamond x_2) \wedge \Box(x_1 \Rightarrow \Diamond x_3) \wedge \\ \Box(x_2 \Rightarrow \bigcirc x_2) \wedge \Box(x_2 \Rightarrow a) \wedge \\ \Box(x_3 \Rightarrow \bigcirc x_3) \wedge \Box(x_3 \Rightarrow \neg a),\end{array}$$

where x_2 and x_3 are fresh propositions. Then we replace conditional eventualities $\Box(x_1 \Rightarrow \Diamond x_2)$ and $\Box(x_1 \Rightarrow \Diamond x_3)$ with unconditional ones. Formula $\Box(x_1 \Rightarrow \Diamond x_2)$ is satisfiability equivalent to $\Box(x_1 \Rightarrow (x_2 \vee w_1)) \wedge \Box(w_1 \Rightarrow \bigcirc(w_1 \vee x_2)) \wedge \Box\Diamond\neg w_1$, where w_1 is a fresh proposition, which, intuitively, is true 'while we are waiting for x_2 to become true'; the other eventuality is treated similarly. All in all, ϕ is satisfiability equivalent to the following temporal DSNF problem.

$$\mathcal{I} = \{\mathsf{i1}\colon x_1\}; \qquad \mathcal{U} = \left\{\begin{array}{l}\mathsf{u1}\colon \neg x_2 \vee a, \\ \mathsf{u2}\colon \neg x_3 \vee \neg a, \\ \mathsf{u3}\colon \neg x_1 \vee x_2 \vee w_1, \\ \mathsf{u4}\colon \neg x_1 \vee x_3 \vee w_2\end{array}\right\}$$

$$\mathcal{S} = \left\{\begin{array}{l}\mathsf{s1}\colon x_2 \Rightarrow \bigcirc x_2, \\ \mathsf{s2}\colon x_3 \Rightarrow \bigcirc x_3, \\ \mathsf{s3}\colon w_1 \Rightarrow \bigcirc(w_1 \vee x_2), \\ \mathsf{s4}\colon w_2 \Rightarrow \bigcirc(w_2 \vee x_3)\end{array}\right\}; \mathcal{E} = \{\mathsf{e1}\colon \Diamond\neg w_1, \mathsf{e2}\colon \Diamond\neg w_2\}.$$

The added labels i1, u1,... have no special meaning and are not part of DSNF; we use them for reference when we return to this example. □

Simplified temporal resolution consists of an (implicit) *merging operation*

$$\frac{P_1 \Rightarrow \bigcirc Q_1, \ldots, P_n \Rightarrow \bigcirc Q_n}{\bigwedge_{j=1}^{n} P_i \Rightarrow \bigcirc \bigwedge_{j=1}^{n} Q_i},$$

and resolution and termination rules defined below. To simplify the presentation, we denote the result of merging of step clauses as $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$ (or $\mathcal{A}_i \Rightarrow \bigcirc\mathcal{B}_i$ if a rule operates several merged step clauses). Thus, in what follows $\mathcal{A}$, $\mathcal{B}$, $\mathcal{A}_i$ and $\mathcal{B}_i$ are conjunctions of propositional literals. As $\mathcal{U}$ contains no temporal operators, all side conditions in the rules are purely propositional.

- *Step resolution rule*:

$$\frac{\mathcal{A} \Rightarrow \bigcirc\mathcal{B}}{\neg\mathcal{A}},$$

where $\mathcal{U} \cup \{\mathcal{B}\}$ is unsatisfiable.

```
function SRes($\mathcal{U}$, $\mathcal{S}$)
    $New = \emptyset$
    for each $mus \in$ AllMus ($\{B \mid A \Rightarrow \bigcirc B \in \mathcal{S}\} \cup \mathcal{U}$) do
        $\mathcal{A} = \bigwedge_{B \in mus,\, A \Rightarrow \bigcirc B \in \mathcal{S}} A$
        $New = New \cup \{\neg\mathcal{A}\}$
    end for
    return $New$
end function
```

Fig. 1: Step resolution

- *Eventuality resolution rule*:

$$\frac{\mathcal{A}_1 \Rightarrow \bigcirc\mathcal{B}_1, \quad \dots, \quad \mathcal{A}_n \Rightarrow \bigcirc\mathcal{B}_n \qquad \Diamond l}{(\bigwedge_{i=1}^{n} \neg\mathcal{A}_i)} \,,$$

  where $\mathcal{U} \cup \{\mathcal{B}_i, l\}$ and $\mathcal{U} \cup \{\mathcal{B}_i, \bigwedge_{j=1}^{n} \neg\mathcal{A}_j\}$, for all i, are unsatisfiable.
- *Termination rule*: **false** is derived if $\mathcal{U} \cup \mathcal{I}$, or $\mathcal{U} \cup \{l\}$ are unsatisfiable, where l is an eventuality literal.

A *derivation* is a sequence of universal parts, $\mathcal{U} = \mathcal{U}_0 \subset \mathcal{U}_1 \subset \mathcal{U}_2 \subset \dots$, extended little by little by the conclusions of the inference rules. Notice that, as the left-hand sides of (merged) step clauses are conjunctions of literals, the step resolution rule generates clauses and the eventuality resolution rule generates sets of clauses. The $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{E}$ parts of the temporal problem are not changed during a derivation. A derivation *terminates* if, and only if, either **false** is derived, in which case we say that the derivation *successfully terminates*, or if no new formulae can be derived by further inference steps. A derivation $\mathcal{U} = \mathcal{U}_0 \subset \mathcal{U}_1 \subset \mathcal{U}_2 \subset \dots \subset \mathcal{U}_n$ is called *fair* if for any $i \geq 0$ and formula ϕ derivable from $\langle \mathcal{U}_i, \mathcal{I}, \mathcal{S}, \mathcal{E} \rangle$ by the rules above, there exists $j \geq i$ such that $\phi \in \mathcal{U}_j$.

Theorem 1 ([7]). *If a DSNF problem $\langle \mathcal{I}, \mathcal{U}, \mathcal{S}, \mathcal{E} \rangle$ is unsatisfiable then any fair derivation by temporal resolution successfully terminates.*

3 Temporal Reasoning with Reductions to MUS

As the side conditions of the inference rules are purely propositional problems, they can be tested with an external SAT Solver. All that remains is to find the appropriate merged step clause, or clauses, which satisfy the side conditions. This straightforward approach has been implemented in [29]; however, in practice, the necessity to try all possibilities to merge clauses led to inability to handle problems with a sizeable step part. In this paper we investigate a possibility to delegate the search for step clauses to merge to an MUS solver.

For an unsatisfiable set of propositional clauses S, its subset $S' \subseteq S$ is called a *minimal unsatisfiable subset* (MUS) if S' is unsatisfiable and every proper

```
function ERES($\mathcal{U}$, $\mathcal{S}$, $\Diamond l$)
    $H = \mathrm{SRES}(\mathcal{U} \cup \{l\}, \mathcal{S})$
    repeat
        $H' = H$;   $H = \mathrm{SRES}(\mathcal{U} \cup \{(l \vee \neg\mathcal{A}') \mid \neg\mathcal{A}' \in H'\}, \mathcal{S})$
        if $H = \emptyset$ then
            return $\emptyset$
        end if
    until $(\bigwedge_{\neg\mathcal{A} \in H} \neg\mathcal{A} \Rightarrow \bigwedge_{\neg\mathcal{A}' \in H'} \neg\mathcal{A}')$ is valid
    return $H$
end function
```

Fig. 2: Eventuality resolution

subset of S' is satisfiable. The number of MUSes for a set of clauses S can be exponential in the size of S. Propositional minimal unsatisfiability has been extensively studied (often under different names) in the literature, and a number of empirically efficient implementations of algorithms enumerating all MUSes for a given set of propositional clauses is available (see the survey [21] and references within).

The step resolution procedure is given in Figure 1. The ALLMUS procedure called returns all MUSes for a set of propositional clauses. By definition, every $\neg\mathcal{A} \in \mathrm{SRES}(\mathcal{S},\mathcal{U})$ is obtained from $\langle\mathcal{I},\mathcal{U},\mathcal{S},\mathcal{E}\rangle$ by an application of the step resolution rule; conversely we have the following.

Lemma 1. *For any DNSF problem $\mathcal{P} = \langle\mathcal{I},\mathcal{U},\mathcal{S},\mathcal{E}\rangle$ such that $\mathcal{U}$ is satisfiable, if $\neg\mathcal{A}$ can be obtained by an application of the step resolution rule from $\mathcal{P}$, then there exists $\neg\mathcal{A}' \in \mathsf{SRes}(\mathcal{S},\mathcal{U})$ such that $(\neg\mathcal{A}' \Rightarrow \neg\mathcal{A})$ is a valid formula.*

Proof. Let $\mathcal{B} = \bigwedge_{i \in I} B_i$. As $\mathcal{U}$ is satisfiable and $\mathcal{B} \wedge \mathcal{U}$ is not, there exists $J \subseteq I$ such that for some MUS mus we have $B_j \in mus$, for every $j \in J$, so $\neg(\bigwedge_{j \in J} A_j)$ will be returned by $\mathrm{SRES}(\mathcal{U},\mathcal{S})$. Clearly, $(\neg(\bigwedge_{j \in J} A_j) \Rightarrow \neg\mathcal{A})$ is valid. □

It has been noticed already in [9] that the search for premises of the eventuality resolution rule can be performed with the help of step resolution. Our algorithm for eventuality resolution given in Figure 2 is based on the BFS algorithm as described in [8]. Notice that every element of the set H in the $\mathrm{ERES}(\mathcal{U}$, $\mathcal{S}$, $\Diamond l)$ procedure is of the form $\neg\mathcal{A}$, where $\mathcal{A}$ is the left-hand side of some merged step clause $\mathcal{A} \Rightarrow \bigcirc\mathcal{B}$.

We demonstrate the working of the ERES procedure by proving its correctness; the proof of completeness of simplified temporal resolution with the eventuality rule applications restricted to the outputs of ERES can be obtained by adapting the proof of completeness in [8] using arguments similar to those used in the proof of Lemma 1.

Lemma 2. *For any DNSF problem $\mathcal{P} = \langle\mathcal{I},\mathcal{U},\mathcal{S},\mathcal{E}\rangle$ let H be returned by $\mathrm{ERES}(\mathcal{U},\mathcal{S},\Diamond l)$. Then H can be obtained from $\langle\mathcal{I},\mathcal{U},\mathcal{S},\mathcal{E}\rangle$ by an application of the eventuality resolution rule.*

```
function PLTL($\mathcal{I}$,$\mathcal{U}$,$\mathcal{S}$,$\mathcal{E}$)
    repeat
        if ($\mathcal{U} = \mathcal{U} \cup \mathrm{SRes}(\mathcal{U},\mathcal{S})$ changes $\mathcal{U}$) then
            check for termination
        else if ($\mathcal{U} = \mathcal{U} \cup \mathrm{ERes}(\mathcal{U},\mathcal{S},\Diamond l)$, for some $\Diamond l \in \mathcal{E}$, changes $\mathcal{U}$) then
            check for termination
        end if
    until There is no change in $\mathcal{U}$
    check for termination
    return 'satisfiable'
end function
```

Fig. 3: Reasoning procedure

Proof. Suppose that $\mathrm{ERes}(\mathcal{U},\mathcal{S},\Diamond l)$ returns a non-empty set of clauses H and let H' be from the last iteration of the loop. Let a set of indices I be such that $H = \{\neg\mathcal{A}_i \mid i \in I\}$ and let $\mathcal{B}_i$ be such that $\mathcal{A}_i \Rightarrow \bigcirc\mathcal{B}_i$ is a merged step clause, for $i \in I$. Then, by properties of $\mathrm{SRes}(\mathcal{U},\mathcal{S})$, for every $i \in I$ the set $\{\mathcal{B}_i\} \cup \{(l \vee \neg\mathcal{A}') \mid \neg\mathcal{A}' \in H'\} \cup \mathcal{U}$ is unsatisfiable. As $(\bigwedge_{\neg\mathcal{A}\in H} \neg\mathcal{A} \Rightarrow \bigwedge_{\neg\mathcal{A}'\in H'} \neg\mathcal{A}')$ is valid, the set $\{\mathcal{B}_i\} \cup \{(l \vee \neg\mathcal{A}) \mid \neg\mathcal{A} \in H\} \cup \mathcal{U}$ is also unsatisfiable. Equivalently, $\mathcal{U} \cup \{\mathcal{B}_i, l\}$ and $\mathcal{U} \cup \{\mathcal{B}_i, \bigwedge_{j\in I} \neg\mathcal{A}_j\}$, for all $i \in I$, are unsatisfiable. But then H can be obtained by an application of the eventuality resolution rule. □

Finally, the overall proof procedure is given in Figure 3. Note in the procedure *check for termination* stands for checking if $\mathcal{U} \cup \mathcal{I}$ or $\mathcal{U} \cup \{l\}$ (for some $\Diamond l \in \mathcal{E}$) are unsatisfiable, in which case proof search terminates returning '*unsatisfiable*'. Combining Lemmata 1 and 2 with results from [8] we obtain the following result.

Theorem 2. PLTL($\mathcal{I}$,$\mathcal{U}$,$\mathcal{S}$,$\mathcal{E}$) *always terminates. DSNF problem* $\langle\mathcal{I},\mathcal{U},\mathcal{S},\mathcal{E}\rangle$ *is satisfiable if, and only if,* PLTL($\mathcal{I}$,$\mathcal{U}$,$\mathcal{S}$,$\mathcal{E}$) *returns 'satisfiable'.*

Example 2 (Example 1 continued). We apply our algorithm to the DSNF problem from Example 1. To simplify the notation, we refer to clauses just by their label. Additionally, we refer to the propositional clause in the right-hand side of a step clause by adding the suffix 'r' to the label of the step clause. For example, s3r denotes $w_1 \vee x_2$, the right-hand side of step clause s3.

When the algorithms starts $\mathcal{U} = \{\mathsf{u1}, \mathsf{u2}, \mathsf{u3}, \mathsf{u4}\}$. We apply PLTL step by step.

SRes($\mathcal{U}$, $\mathcal{S}$). AllMus($\{\mathsf{u1}, \mathsf{u2}, \mathsf{u3}, \mathsf{u4}, \mathsf{s1r}, \mathsf{s2r}, \mathsf{s3r}, \mathsf{s4r}\}$), returns just one MUS $\{\mathsf{u1}, \mathsf{u2}, \mathsf{s1r}, \mathsf{s2r}\}$. As $\{\mathsf{s1r}, \mathsf{s2r}\} \cup \mathcal{U}$ is unsatisfiable, the step resolution rule applies to the result of merging of s1 and s2 generating new universal clause: u5: $\neg x_2 \vee \neg x_3$, which is returned by the procedure and added to $\mathcal{U}$. As $\mathcal{U}$ is changed, the *check for termination* is employed, but it does not succeed. Then $\mathrm{SRes}(\mathcal{U}, \mathcal{S})$ is called again, but one can see that the second call leads to no change in $\mathcal{U}$.

ERes$(\mathcal{U}, \mathcal{S}, \Diamond\neg w_1)$**.** In order to compute H, SRes$(\mathcal{U} \cup \{\mathsf{t1}\colon \neg w_1\}, \mathcal{S})$ is called, where t1 is a label used to denoted the temporary universal clause. AllMus$(\{\mathsf{u1}, \mathsf{u2}, \mathsf{u3}, \mathsf{u4}, \mathsf{u5}, \mathsf{t1}, \mathsf{s1r}, \mathsf{s2r}, \mathsf{s3r}, \mathsf{s4r}\})$ returns a set containing 4 MUSes, $\{\{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u1}, \mathsf{u2}\}, \{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u5}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{t1}, \mathsf{u1}, \mathsf{u2}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{t1}, \mathsf{u5}\}\}$ however H only contains 2 clauses $\{\neg x_2 \vee \neg x_3, \neg x_3 \vee \neg w_1\}$ as the two first and the two last MUSes contain the same right-hand sides of step clauses. We set $H' = H$ and run the loop body. The call of SRes$(\mathcal{U} \cup \{\mathsf{t2}\colon \neg w_1 \vee \neg x_2 \vee \neg x_3, \mathsf{t3}\colon \neg w_1 \vee \neg x_3\}, \mathcal{S})$ passes $\{\mathsf{u1}, \mathsf{u2}, \mathsf{u3}, \mathsf{u4}, \mathsf{u5}, \mathsf{t2}, \mathsf{t3}, \mathsf{s1r}, \mathsf{s2r}, \mathsf{s3r}, \mathsf{s4r}\}$ to AllMus which returns $\{\{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u1}, \mathsf{u2}\}, \{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u5}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{t3}, \mathsf{u1}, \mathsf{u2}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{t3}, \mathsf{u5}\}\}$, so $H = \{\neg x_2 \vee \neg x_3, \neg x_3 \vee \neg w_1\}$. As $H = H'$, ERes terminates and returns two universal clauses u6: $\neg x_2 \vee \neg x_3$ and u7: $\neg x_3 \vee \neg w_1$. Only u7 is new and is added to $\mathcal{U}$; u6 is discarded as redundant. As $\mathcal{U}$ is changed, the *check for termination* is employed, but it does not succeed.

ERes$(\mathcal{U}, \mathcal{S}, \Diamond\neg w_2)$**.** H is computed. SRes$(\mathcal{U} \cup \{\mathsf{t5}\colon \neg w_2\}, \mathcal{S})$ is called and AllMus$(\{\mathsf{u1}, \mathsf{u2}, \mathsf{u3}, \mathsf{u4}, \mathsf{u5}, \mathsf{t5}, \mathsf{s1r}, \mathsf{s2r}, \mathsf{s3r}, \mathsf{s4r}\})$ returns $\{\{\mathsf{s3r}, \mathsf{s4r}, \mathsf{u1}, \mathsf{u2}, \mathsf{u7}, \mathsf{t5}\}, \{\mathsf{s3r}, \mathsf{s4r}, \mathsf{u7}, \mathsf{t5}, \mathsf{u5}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{u1}, \mathsf{u2}, \mathsf{u7}\}, \{\mathsf{s2r}, \mathsf{s3r}, \mathsf{u7}, \mathsf{u5}\}, \{\mathsf{s1r}, \mathsf{s4r}, \mathsf{u1}, \mathsf{u2}, \mathsf{t5}\}, \{\mathsf{s1r}, \mathsf{s4r}, \mathsf{t5}, \mathsf{u5}\}, \{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u1}, \mathsf{u2}\}, \{\mathsf{s1r}, \mathsf{s2r}, \mathsf{u5}\}\}$ so $H = \{\neg x_2 \vee \neg x_3, \neg x_3 \vee \neg w_1, \neg x_2 \vee \neg w_2, \neg w_1 \vee \neg w_2\}$. The run of the loop body is omitted to save space, however it computes H being same as H', so ERes terminates and returns four universal clauses $\{\mathsf{u8}\colon \neg x_2 \vee \neg x_3, \mathsf{u9}\colon \neg x_3 \vee \neg w_1, \mathsf{u10}\colon \neg x_2 \vee \neg w_2, \mathsf{u11}\colon \neg w_1 \vee \neg w_2\}$. Only u10 and u11 are new, which are added to $\mathcal{U}$.

Finally as $\mathcal{U} \cup \mathcal{I}$ is unsatisfiable we can apply the termination rule and so PLTL$(\mathcal{I}, \mathcal{U}, \mathcal{S}, \mathcal{E})$ returns 'unsatisfiable'. □

Optimisations. As Example 2 demonstrates, different MUSes can contain the same right-hand sides of step clauses, which is not optimal. The search for merged clauses can be significantly sped up by *grouping* universal clauses together so that instead of looking for all minimal unsatisfiable subsets of $\{B \mid A \Rightarrow \bigcirc B \in \mathcal{S}\} \cup \mathcal{U}$, we look for all subsets $S \subseteq \{B \mid A \Rightarrow \bigcirc B \in \mathcal{S}\}$ such that $(S \cup \mathcal{U})$ is unsatisfiable and for every proper subset S' of S, $(S' \cup \mathcal{U})$ is satisfiable. In other words, all universal clauses are considered as one item. Not only is the number of MUSes with grouped universal clauses smaller than the number of all MUSes but also, crucially, MUS enumeration tools can efficiently take grouping into account [19].

We further exploit the disparity between the treatment of right-hand sides of step clauses and universal clauses by rewriting a given DNSF problem into a satisfiability equivalent problem having a smaller number of step clauses. If $\mathcal{S}$ contains two step clauses $A \Rightarrow \bigcirc B_1$ and $A \Rightarrow \bigcirc B_2$ with the same left-hand side, we first equivalently rewrite them into $A \Rightarrow \bigcirc(B_1 \wedge B_2)$ and then rename the conjunction $B_1 \wedge B_2$, to preserve the clausal form, to give a new step clause $A \Rightarrow \bigcirc X$ and two new universal clauses $\neg X \vee B_1$ and $\neg X \vee B_2$, where X is a fresh proposition. Similarly, if $\mathcal{S}$ contains two step clauses $A_1 \Rightarrow \bigcirc B$ and $A_2 \Rightarrow \bigcirc B$ with the same right-hand side, we equivalently rewrite them into $(A_1 \vee A_2) \Rightarrow \bigcirc B$ and then rename the disjunction in the left-hand side.

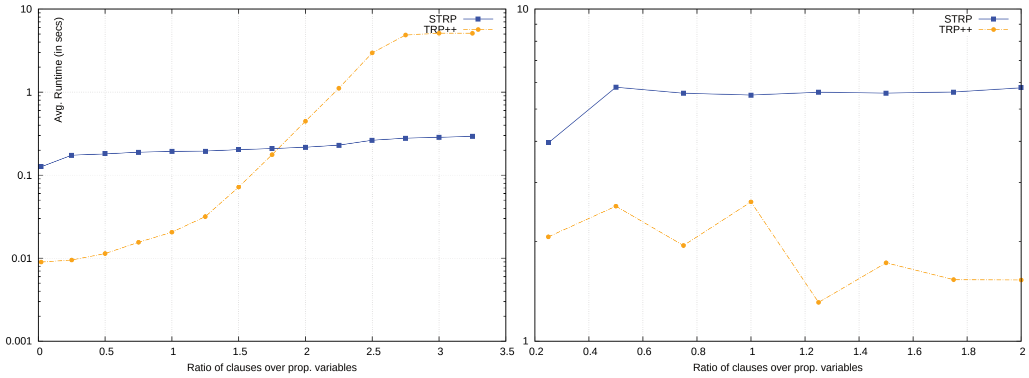

Fig. 4: Results for C^1_{ran} (left) and C^2_{ran} (right)

4 Experimental Evaluation

We have implemented the described approach in the STRP prover[1], which uses the CAMUS system [19] as the MUS enumeration tool. CAMUS works in two stages, the first extracts all minimal correction subsets (MCSes) from the input propositional clauses and the second extracts MUSes from the set of MCSes by extracting all minimal hitting sets (also know as minimal hypergraph transversals) from the set of all MCSes. Although both stages are not tractable [19], in our preliminary experiments, the second stage of CAMUS took much more time than the first stage. We put this down to the nature of our problems: SAT benchmarks typically are larger problems with a smaller number of MUSes [1], whereas our MUS problems are much smaller but typically contain a larger number of MUSes. From a small, randomly selected, sample of the benchmarks used in this work we have established a typical CNF problem size of 800–900 variables and 3500–4500 clauses (of which 120–130 were the right-hand sides of step clauses), from which about 1400 MUSes are typically extracted. We therefore replaced the second stage of CAMUS with other hypergraph transversal computation tools, MTminer [16] and shd [22]. Both tools proved to be two orders of magnitude faster on our problems; MTminer is slightly faster but it uses significantly more memory than shd.

Our experimental evaluation is focused purely on a comparison of the clausal-resolution based prover TRP++ [17], which has previously been shown to perform well in a number of studies [17, 25], and our new simplified resolution-based prover STRP. While a more comprehensive comparison featuring other proof methods similar to [25, 24] would provide interesting results it is beyond the scope of this current work. Notice however that we re-use some benchmark problems from previous studies [17, 25], which evaluated the performance of TRP++

[1] Available at `http://www.csc.liv.ac.uk/~rmw/STRP.html`

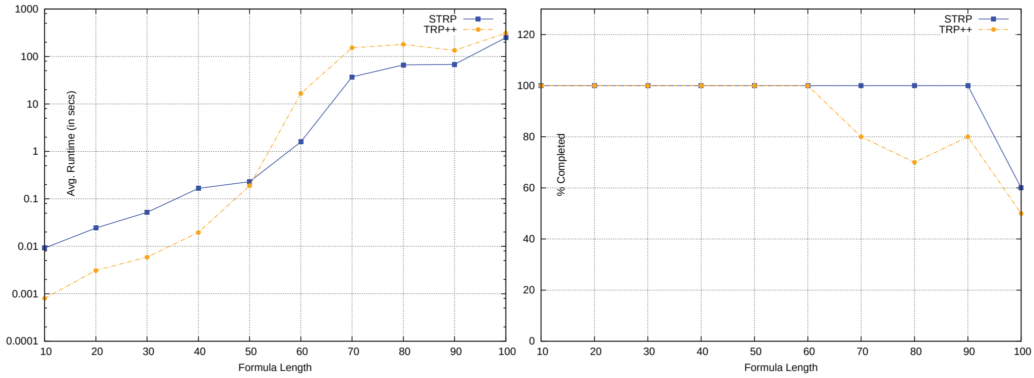

Fig. 5: Results for Rozier Benchmarks Runtimes (left) and % Tests completed (right)

against other systems, thus the performance of STRP compared with other systems can be derived from published results and our comparison. All experiments were conducted on PC with an Intel Core i5-2500K 3.30GHz CPU, with 16GB of RAM running Scientific Linux 6.3. As TRP++ and STRP both operate on inputs in DSNF, time taken to translate input formulae to DSNF has not been taken into account.

Random benchmarks. The first experiment involved two classes of semi-random benchmark formulae, C^1_{ran} and C^2_{ran} introduced in [18]. In previous experiments [17] TRP++ performed extremely fast (less than 0.1 on most problems), so we increased the size of the problems using the following parameters for the random formulae: $n = 48, k = 6$ and $p = 0.5$ where n is the number of propositional variables and k determines the number of distinct random variables chosen, with the polarity of each literal determined by the probability p. The results (Fig. 4) show STRP performing very consistently on both sets of problems irrespective of their size. A remarkable STRP performance on C^1_{ran} can be explained by the fact that all step clauses of C^1_{ran} problems are of the form $\mathbf{true} \Rightarrow \bigcirc(L_1 \vee ... \vee L_k)$, which are then all rewritten as a single step clause by the optimisation described above. In case of C^2_{ran}, the number of step clauses in the random formulae of different size remains fairly constant while the initial and eventuality parts increase in size and complexity. These results demonstrate the usefulness of the optimisations described above as well as suggest that the STRP performance mainly depends on the size of the step part of the input formula rather than on the size of other parts.

Another set of random benchmarks is sourced from [24]. This set of benchmarks has been first used to compare model checking approaches in [24] and as part of a more complete comparison of PLTL provers [25]. We used benchmarks with the following parameters $n = 5, p = 0.95$ and $l = 10a \dots 100$, where n is the

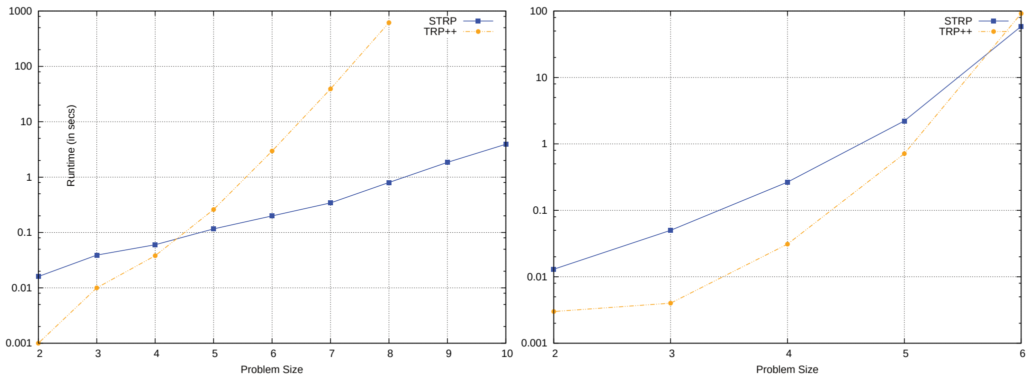

Fig. 6: O2 formulae (left) and phltl formulae (right)

number of variables, p is the probability of choosing temporal operator and l is the length of the formula. The method used to generate the random formulae does not allow one to directly link the length of the formula with the number of step clauses. For example there are some problems of length $n = 90$ with well over 200 step clauses whereas some problems of length $n = 100$ contain only 100 step clauses. This variation in the number of step clauses helps to account for the variable performance, particularly for STRP, as shown in Fig. 5.

The results also demonstrate that, while both system exhibit a similar dynamics with the growth of the formula size, on more complex problems ($l = 60$ and more), STRP is 5 to 10 times faster than TRP++. The lines converge due to timeouts. The plot of percentage of tests completed within the 1800s time limit (Fig. 5 right) shows that the number of tests TRP++ is able to complete within the time limit starts to drop after $l = 60$ whereas STRP only shows a decline only after $l = 90$.

Crafted benchmarks. Crafted benchmarks [25] are sets of PLTL formulae that have specifically been designed to trigger an exponential behaviour of PLTL solvers. Both TRP++ and STRP perform well on the O1 family and on the 'pattern' formulae [25] with TRP++ spending 37 seconds on the hardest Rformula1000, which contains 1998 sometime clauses and 3996 step clauses, and STRP spending 128 seconds. Problems of such large size can only be solved due the the fact that they are all trivially satisfiable or trivially unsatisfiable. For example, none of the 'pattern' formulae contain occurrences of negated propositions. Thus, simplified temporal resolution does not generate anything new from the input problem and clausal resolution generates very few (1002 in case of Rformula1000) new clauses. The extra time taken by STRP is due to I/O overhead passing information to and from the MUS extractor.

The families of O2 and phltl formulae are more challenging for both systems. STRP solves two more O2 problems within the $1,800$ second time limit; both

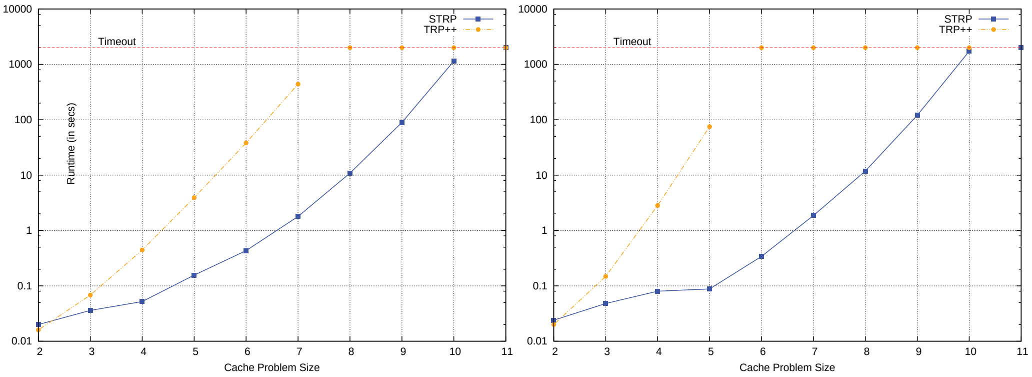

Fig. 7: Cache Coherence Results m(left) sm(right)

systems show a consistent behaviour on phltl benchmarks as shown in Fig. 6. Both systems timeout on problems of size larger than given in the graphs.

TLC Cache Coherence benchmarks. Temporal Logic with Cardinality Constraints (TLC) is an extension of PLTL with global constraints on temporal interpretations, which has been introduced in [10] to capture real-world problems. An example of a TLC constraint is $\{p, q, r\}^{=1}$, which requires that exactly one proposition from the set of $\{p, q, r\}$ is true at any moment of time. The expressive power of TLC is the same as PLTL as the constraints can be captured by temporal formulae. In our example, the constraint is captured by $\Box(p \lor q \lor r)$, $\Box(\neg p \lor \neg q)$, $\Box(\neg p \lor \neg r)$, $\Box(\neg q \lor \neg r)$. It has been argued that specialised tools are needed for practical reasoning in presence of cardinality constraints since PLTL formulae that capture them are too large and complex for existing provers [10].

PLTL representations of TLC formulae provide an interesting set of problems for our comparison as, due to the global nature of constraints, temporal formulae capturing such constraints contribute only the the universal part of DSNF. We use two families of TLC formulae introduced in [10] capturing verification conditions on a cache coherence protocol with n-processes, 'm': no two processes can simultaneously be in state m; and 'sm': it is not possible for one process to be in state s and another in state m. The problems feature an increasing number of processes; the number of transitions between the states is small but the set of constraints is large and complex. As a result, the PLTL representation of the original problem has a comparatively small number of step clauses but a very large number of universal clauses. As shown in Fig. 7, STRP outperforms TRP++ completing several more problems within the 1800s time limit.

Verification benchmarks. The Anzu verification benchmarks used in [3, 25] provide a good counter example to the Cache Coherence problems and are particularly difficult for STRP. The smallest example from this dataset (genbuf/spec1) takes STRP $416.266s$ whereas TRP++ only takes $0.236s$. These problems feature

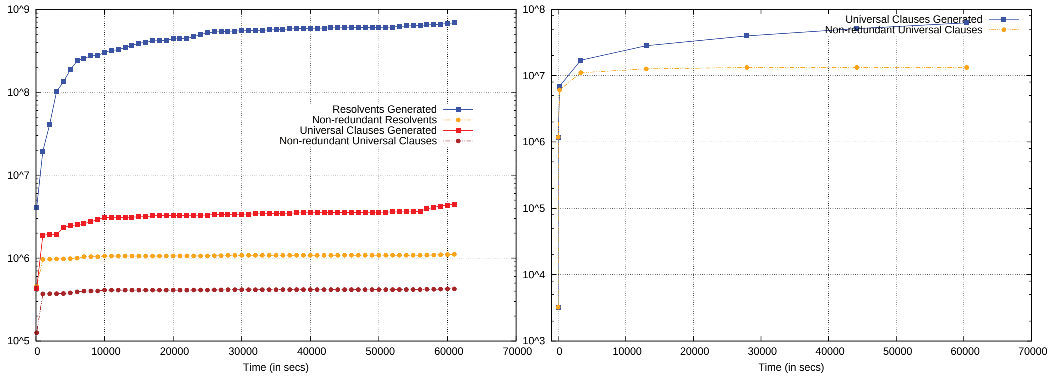

Fig. 8: TRP++ results (left) and STRP results (right)

a moderate number of step clauses (the smallest containing 36 step clauses); however, we did find an interesting characteristic on the small number of problems we were able to run. The benchmarks produce a very large number of MCSes (43 738 on average) which are then reduced to a very small number of MUSes (58 on average) this means both stages of the MUS enumeration process take significantly longer than on other datasets explored in this work.

Performance Degradation To evaluate how the performance of each solver changes over time, we let both systems run for 60 000s on a difficult random Rozier problem (P0.5N1L190). We captured for TRP++ the number of resolvents and universal clauses generated (both total and non-redundant) and for STRP the number of universal clauses generated (both total and non-redundant). For TRP++ this data was captured at regular intervals and for STRP we recorded a data point at each iteration of the main procedure.

The results (Fig. 8) show that TRP++ generates far fewer non-redundant universal clauses (125973) within the first 5 seconds of computation than STRP (1180608). Moreover, TRP++'s performance slows down noticeably at this point whereas STRP continues generating new clauses before stabilising at well over 13 million universal clauses. These numbers are not directly comparable as the systems utilise different calculi. In particular, clause-level redundancy elimination in TRP++ can be responsible for fewer non-redundant clauses being retained. However, in both calculi only universal and initial clauses contribute to the refutation of the given problem. STRP shows quite remarkable performance as it generates significantly more non-redundant universal clauses in a much shorter timeframe than TRP++.

Notice also that all formulae derived by STRP are added to the universal part, thus the search space remains constant throughout the run, which is not the case for TRP++.

5 Conclusions and future work

In this paper we have investigated a new approach to PLTL reasoning based on reductions to MUS enumeration. Despite the simplicity of the approach, our prototype implementation proved to perform very well on a significant number of benchmarks. Our new system performed especially well on problems having a relatively small number of step clauses but larger number of universal clauses.

Closest to our approach is bounded model checking [2], which can also establish satisfiability of PLTL formulae. However, in bounded model checking propositional formulae represent bounded-depth traces of a system and SAT solvers are used to check their realisability, while in our approach MUSes are used to facilitate resolutional proof search. Recent developments in bounded model checking include incremental inductive reasoning [4] and counting [5], which both can handle unbounded problems. The extraction of labelled superposition proofs from bounded system traces is explored in [26]. Reasoning procedures for modal logics with reductions to SAT have also been investigated in [13, 14].

There are a number of possible ways to improve the performance of STRP, which constitute future work. At the moment, we use an MUS enumerating procedure as a black box. Consecutive calls of AllMus can return already known MUSes, which are then discarded as redundant. One can reuse information from the previous runs of AllMus to avoid generation of redundant MUSes. This will require modifying the MUS extractor. On larger CNF instances the MUS enumeration procedure can take a significant amount of time to return the complete set of all MUSes. It may be possible to return MUSes as and when they are derived. This facility may be useful in the SRes on unsatisfiable problems as successful application of the termination rules may be possible without the need to generate all MUSes. Finally, it would be interesting to more thoroughly investigate the impact of optimisations reducing the size of the step part on the performance of our system.

References

1. Belov, A., I.Lynce, Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. 25(2), 97–116 (2012)
2. Biere, A.: Bounded model checking. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 457–481. IOS Press (2009)
3. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Automatic hardware synthesis from specifications: A case study. In: Proceedings of DATE'07. pp. 1–6. IEEE (2007)
4. Bradley, A.R.: SAT-based model checking without unrolling. In: Proceedings of VMCAI'11. LNCS, vol. 6538, pp. 70–87. Springer (2011)
5. Claessen, K., Sörensson, N.: A liveness checking algorithm that counts. In: Proceedings of FMCAD'12. pp. 52–59. IEEE (2012)
6. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (1999)
7. Degtyarev, A., Fisher, M., Konev, B.: A simplified clausal resolution procedure for propositional linear-time temporal logic. In: Proceedings of TABLEAUX'02. LNCS, vol. 2381, pp. 85–99. Springer (2002)

8. Degtyarev, A., Fisher, M., Konev, B.: Monodic temporal resolution. ACM Trans. Comput. Log. 7(1), 108–150 (2006)
9. Dixon, C.: Using Otter for temporal resolution. In: Advances in Temporal Logic. pp. 149–166. Kluwer (2000)
10. Dixon, C., Konev, B., Fisher, M., Nietiadi, S.: Deductive temporal reasoning with constraints. Journal of Applied Logic (2012)
11. Fisher, M., Dixon, C., Peim, M.: Clausal temporal resolution. ACM Transactions on Computational Logic 2(1), 12–56 (2001)
12. Fisman, D., Kupferman, O., Sheinvald-Faragy, S., Vardi, M.Y.: A framework for inherent vacuity. In: Haifa Verification Conference. LNCS, vol. 5394, pp. 7–22 (2008)
13. Giunchiglia, E., Tacchella, A., Giunchiglia, F.: SAT-based decision procedures for classical modal logics. J. Autom. Reasoning 28(2), 143–171 (2002)
14. Giunchiglia, F., Sebastiani, R.: A SAT-based decision procedure for ALC. In: Proceedings of KR'96. pp. 304–314. Morgan Kaufmann (1996)
15. Halpern, J.Y., van der Meyden, R., Vardi, M.Y.: Complete axiomatizations for reasoning about knowledge and time. SIAM J. Comput. 33(3), 674–703 (2004)
16. Hébert, C., Bretto, A., Crémilleux, B.: A data mining formalization to improve hypergraph minimal transversal computation. Fundamenta Informaticae 80(4), 415–433 (2007)
17. Hustadt, U., Konev, B.: TRP++2.0: A temporal resolution prover. In: Proceedings of CADE'03. LNCS, vol. 2741, pp. 274–278. Springer (2003)
18. Hustadt, U., Schmidt, R.A.: Scientific benchmarking with temporal logic decision procedures. In: Proceedings of KR'02. pp. 533–546. Morgan Kaufmann (2002)
19. Liffiton, M., Sakallah, K.: Algorithms for computing minimal unsatisfiable subsets of constraints. Journal of Automated Reasoning 40(1), 1–33 (2008)
20. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems: Specification. Springer (1992)
21. Marques-Silva, J.: Computing minimally unsatisfiable subformulas: State of the art and future directions. Multiple-Valued Logic and Soft Computing 19(1-3), 163–183 (2012)
22. Murakami, K., Uno, T.: Efficient algorithms for dualizing large-scale hypergraphs. CoRR abs/1102.3813 (2011)
23. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings POPL'89. pp. 179–190. ACM Press (1989)
24. Rozier, K., Vardi, M.: LTL satisfiability checking. International Journal on Software Tools for Technology Transfer 12(2), 123–137 (2010)
25. Schuppan, V., Darmawan, L.: Evaluating LTL satisfiability solvers. In: Proceedings of ATVA'11. LNCS, vol. 6996, pp. 397–413 (2011)
26. Suda, M., Weidenbach, C.: A PLTL-prover based on labelled superposition with partial model guidance. In: Proceedings of IJCAR'12. LNCS, vol. 7364, pp. 537–543. Springer (2012)
27. Tansel, A.U., Clifford, J., Gadia, S.K., Jajodia, S., Segev, A., Snodgrass, R.T. (eds.): Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings (1993)
28. Vardi, M., Wolper, P.: Automata-theoretic techniques for modal logics of programs. Journal of Computer and System Sciences 32(2), 183–219 (1986)
29. Williams, R., Konev, B.: Simplified temporal resolution using SAT solvers. In: Proceedings of ARW'12. pp. 9–10. The University of Manchester (2012)
30. Wolper, P.: The tableau method for temporal logic: An overview. Logique et Analyse 28, 119–152 (1985)