Math.Comput.Sci. (2015) 9:327–344
DOI 10.1007/s11786-015-0240-2
Mathematics in Computer Science

Variable and Clause Elimination for LTL Satisfiability Checking

Martin Suda

Received: 26 August 2014 / Revised: 19 July 2015 / Accepted: 30 July 2015 /
Published online: 25 September 2015

Abstract We study preprocessing techniques for clause normal forms of LTL formulas. Applying the mechanism of labeled clauses enables us to reinterpret LTL satisfiability as a set of purely propositional problems and thus to transfer simplification ideas from SAT to LTL. We demonstrate this by adapting variable and clause elimination, a very effective preprocessing technique used by modern SAT solvers. Our experiments confirm that even in the temporal setting substantial reductions in formula size and subsequent decrease of solver runtime can be achieved.

Keywords Linear temporal logic · Satisfiability · Preprocessing

Mathematics Subject Classification 68T15 Theorem proving (deduction, resolution, etc.) · 03B35 Mechanization of proofs and logical operations · 03B44 Temporal logic

1 Introduction

Propositional linear temporal logic (LTL) is a modal logic with modalities referring to time [13]. Traditionally, it finds its use in formal verification of reactive systems where it serves as a specification language for expressing the system's desired behavior. The specifications are subsequently checked against a model of the system during the process of *model checking* [3]. More recently, the importance of LTL *satisfiability checking* is becoming recognized [14,16], where the task is to decide whether a given LTL formula has a model at all. This is, for instance, essential for assuring quality of formal specifications [12]. Satisfiability checking of LTL is a computationally difficult task, in fact a PSPACE-complete one [17], and thus techniques for improving solving methods are of practical importance.

One possibility for speeding up the checking lies in simplifying the input formula before the actual decision method is started. In the context of resolution-based methods for LTL satisfiability [8,18], on which we focus here, formulas are first translated into a clause normal form. Simplification then means reducing the number of clauses and variables while preserving satisfiability of the formula. Such a preprocessing step may have a significant positive impact on the subsequent running time.

In this paper we take inspiration from the SAT community where a technique called variable and clause elimination [5] has been shown to be particularly effective. It combines exhaustive application of the resolution rule over selected

M. Suda (✉)
Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany
e-mail: suda@mpi-inf.mpg.de

variables with subsumption and other reductions. Our main contribution lies in showing that variable and clause elimination can be adapted from SAT to the setting of LTL. This is quite non-trivial, because LTL normal forms consist of *temporal clauses*, which are bound to specific temporal contexts and so their interactions in inferences and reductions need to be carefully controlled.

A general method for reducing LTL satisfiability to the purely propositional setting has been introduced in [18]. There, the existence of a model of an LTL formula is shown to be equivalent to satisfiability of one of infinitely many potentially infinite standard clause sets. These are, however, finitely represented with the help of *labels*, which allows for an effective transfer of resolution-based reasoning techniques from propositional logic to LTL. In this paper, we extend the ideas of [18] to adapt variable and clause elimination. An additional label component is needed to justify elimination in its general form, but we prove it can be dispensed with after the elimination process.

Our exposition starts in Sect. 2, where we describe our version of clause normal form of LTL formulas, which we call temporal satisfiability task (TST). TSTs are a particular refinement of the separated normal form [7], which can be seen as concise representations of Büchi automata. This observation, which is of independent interest, represents another contribution of this paper. The mechanism of labeled clauses itself is introduced in Sect. 3 and utilized for variable and clause elimination in Sect. 4. Practical potential of our method is demonstrated in Sect. 5, where we describe the effect of the simplification on runtimes of two resolution-based LTL provers over an extensive set of benchmark problems. In Sect. 6 we follow the connection to Büchi automata to discuss related work, and we conclude in Sect. 7 by mentioning possibilities for future work.

2 Preliminaries

We assume the reader is familiar with propositional logic and the syntax and semantics of LTL.[1] LTL formulas are built over a given *signature* $\Sigma = \{p, q, r, \ldots\}$ of propositional variables using propositional connectives $\neg, \wedge, \vee, \ldots$, and temporal operators $\bigcirc, \Box, \Diamond, \mathsf{U}, \ldots$ Propositional clauses, denoted C, D, possibly with subscripts, are sets of literals understood as disjunctions. A propositional *valuation* is a mapping $W : \Sigma \to \{\mathbf{0}, \mathbf{1}\}$. We write $W \models C$ if a valuation W propositionally satisfies a clause C. An *interpretation* of an LTL formula is an infinite sequence of valuations $(W_i)_{i \in \mathbb{N}}$, in this context also referred to as *states*.

In order to talk about two neighboring states at once we introduce a disjoint copy of the basic signature $\Sigma' = \{p', q', r', \ldots\}$. Given a clause C over Σ, we write C' to denote its obvious counterpart over Σ'. For a valuation W over Σ let W' denote the valuation over Σ' that behaves on primed symbols in the same way as W does on non-primed ones. We therefore have $W \models C$ if and only if $W' \models C'$ for any such W and C. If W_1 and W_2 are two valuations over Σ, we let $[W_1, W_2]$ denote the joined valuation $W_1 \cup (W_2)' : \Sigma \cup \Sigma' \to \{\mathbf{0}, \mathbf{1}\}$. Such a valuation is needed to evaluate clauses over the joined signature $\Sigma \cup \Sigma'$.

Most resolution-based approaches to satisfiability checking first translate the input formula into a certain normal form. In the context of LTL, the separated normal form (SNF) developed by Fisher [7] has proven to be very useful. It is obtained from an LTL formula by applying transformations that (1) introduce new variables as names for complex subformulas, (2) remove temporal operators by expanding their fixpoint definitions, (3) apply classical rewrite operations to obtain a result which is clausal, i.e. represented by a top-level conjunction of certain temporal clauses, which are disjunctive in nature. The whole transformation preserves satisfiability of the input formula and it is ensured that the result does not grow in size by more than a linear factor [8].[2]

In this paper, we use a particular refinement of SNF which we call temporal satisfiability task (TST).[3] To obtain a TST, a general SNF is first normalized further by using the ideas of [4]. In particular, we transform certain temporal clauses called conditional eventuality clauses to unconditional ones and then reduce the potentially multiple (unconditional) eventuality clauses to just one eventuality clause.[4] Finally, to obtain a compact representation, we

[1] See Appendix A for a short overview.

[2] A streamlined version of the transformation can be found in Appendix B.

[3] Our previous work [18,20] uses the term “LTL-specification” for this concept.

[4] A recapitulation of these refinements has been moved to Appendix B.

explicitly sort the clauses into three categories, strip off the temporal operators $\Box$ and $\Diamond$, and write the clauses down as standard propositional ones using the priming notation. Even after these refinements the result is linearly bounded in size and equisatisfiable with respect to the original formula.

Definition 2.1 A *TST* is a quadruple $\mathcal{T} = (\Sigma, I, T, G)$ such that

- Σ is a finite propositional signature,
- I is a set of *initial* clauses C_i over the signature Σ,
- T is a set of *step* clauses $C_t \vee (D_t)'$ over the joined signature $\Sigma \cup \Sigma'$,
- G is a set of *goal* clauses C_g over the signature Σ.

The initial and step clauses are directly translated from SNF. The goal clauses *all together* express the single eventuality obtained in the previous step. This generalization (from a single goal clause) is for free and appears to make the definition conceptually cleaner. Intuitively, a TST stands for the LTL formula

$$\left(\bigwedge C_i\right) \wedge \Box \left(\bigwedge (C_t \vee \bigcirc D_t)\right) \wedge \Box\Diamond \left(\bigwedge C_g\right),$$

which directly translates to the following formal definition.

Definition 2.2 An interpretation $(W_i)_{i\in\mathbb{N}}$ is a *model* of $\mathcal{T} = (\Sigma, I, T, G)$ if

1. for every $C_i \in I$, $W_0 \models C_i$,
2. for every $i \in \mathbb{N}$ and every $C_t \vee (D_t)' \in T$, $[W_i, W_{i+1}] \models C_t \vee (D_t)'$, and
3. there are infinitely many indexes j such that for every $C_g \in G$, $W_j \models C_g$.

A TST $\mathcal{T}$ is *satisfiable* if it has a model.

Remark 2.3 We close this section with an interesting observation relating our approach to LTL satisfiability to explicit methods based on automata. It is well known (see e.g. [9]) that for any LTL formula φ there is a Büchi automaton $\mathcal{A}_\varphi$ recognizing models of φ, i.e. an automaton that accepts exactly those valuations $(W_i)_{i\in\mathbb{N}}$ that are models of φ. The size of such an automaton, i.e. the number of its states, is bounded by $2^{|\varphi|}$, where $|\varphi|$ denotes the size of the formula.

Now we can easily interpret a TST $\mathcal{T}$ as a *symbolic representation* of such an automaton. The states of the automaton are formed by the set $Q = 2^\Sigma$, i.e. the set of all valuations over Σ, its transition function $\delta = \{(W_1, W_2) \mid [W_1, W_2] \models \bigwedge (C_t \vee (D_t)')\}$ contains those pairs of valuations that satisfy the step clauses, and its initial and accepting sets are defined as $Q_I = \{W \mid W \models \bigwedge C_i\}$ and $Q_F = \{W \mid W \models \bigwedge C_g\}$, respectively. It is easy to check that the models of $\mathcal{T}$ are exactly the accepting runs of this automaton.

This way one can view the transformations from an LTL formula to SNF and further to a TST as an alternative way of obtaining a Büchi automaton for the formula. Interestingly, it is only the last step, when the automaton is made explicit, that incurs the inherent exponential blowup.

3 Mechanism of Labeled Clauses

The purpose of this section is to show that the task of LTL satisfiability can be reduced to a set of purely propositional SAT problems. This provides a means for transferring the well-known resolution-based reasoning techniques from the propositional level to that of LTL. In particular, it will in Sect. 4 allow us to transfer variable and clause elimination. The reduction from LTL that we present leaves us with infinitely many propositional problems over an infinite signature. Labels are then used to *finitely* represent and control clauses within these problems, abbreviating entire clause sets.

Assume we have a TST $\mathcal{T} = (\Sigma, I, T, G)$ and want to decide satisfiability of the formula it represents. It is a known fact that when considering satisfiability of LTL formulas attention can be restricted to *ultimately periodic* [17] interpretations. These start with a finite sequence of states and then repeat another finite sequence of states forever. This observation, which is one of the key ingredients of our approach, motivates the following definition.

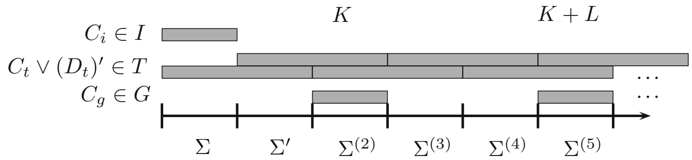

Fig. 1 Schematic presentation of the potentially infinite set of clauses that is satisfiable if and only if a TST $\mathcal{T} = (\Sigma, I, T, G)$ has a (K, L)-model with $K = 2$ and $L = 3$. The axis represents the infinite signature Σ^*, while the *gray bars* stand for individual copies of the initial, step, and goal clauses, respectively

Definition 3.1 Let $K \in \mathbb{N}$, and $L \in \mathbb{N}^+ = \mathbb{N}\setminus\{0\}$ be given. An interpretation $(W_i)_{i\in\mathbb{N}}$ is a (K, L)-*model* of $\mathcal{T} = (\Sigma, I, T, G)$ if

1. for every $C \in I$, $W_0 \models C$,
2. for every $i \in \mathbb{N}$ and every $C \in T$, $[W_i, W_{i+1}] \models C$,
3. for every $i \in \mathbb{N}$ and every $C \in G$, $W_{(K+i\cdot L)} \models C$.

We will call the pair (K, L) of natural numbers $K \in \mathbb{N}$ and $L \in \mathbb{N}^+$ a *rank* of a model.

Satisfiability within a (K, L)-model for *some* values of K and L corresponds to the original semantics except that the condition on the goal clauses to be satisfied in infinitely many states is now controlled and we require that these states form an arithmetic progression with K as the initial term and L the common difference. Please consult [19] for a detailed proof of why focusing only on (K, L)-models does not change the notion of satisfiability.

For a particular choice of K and L, the existence of a (K, L)-model can be stated as an infinite but purely propositional problem over the infinite signature $\Sigma^* = \bigcup_{i\in\mathbb{N}} \Sigma^{(i)}$. Here we extend the convention about priming and allow it to be applied more than once. Thus along with signatures Σ and Σ' we also have Σ'', Σ''', ... (also written $\Sigma^{(2)}$, $\Sigma^{(3)}$, ...), as other disjoint copies of the basic signature implicitly meant to represent states further in the future. Now the purely propositional problem simply restates the definition of a (K, L)-model in the form of clauses over Σ^*, making use of the natural bijection between propositional valuations over Σ^* and interpretations.[5] It consists of:

- the set of initial clauses $I = \{C^{(0)} \mid C \in I\}$,
- together with $\{C^{(i)} \mid C \in T, i \in \mathbb{N}\}$,
- and with $\{C^{(K+i\cdot L)} \mid C \in G, i \in \mathbb{N}\}$,

where the symbol $C^{(i)}$ means that each literal in C is being "moved i signatures forward". Thus, e.g., for a clause $C = p \vee q'$ over $\Sigma \cup \Sigma'$ we denote by $C^{(2)}$ the clause $p^{(2)} \vee q^{(3)}$ over $\Sigma^{(2)} \cup \Sigma^{(3)}$. See Fig. 1 for an illustration of the situation.

3.1 Introducing Labels

We have now reduced LTL satisfiability of a TST $\mathcal{T}$ to infinitely many (for every pair of K and L) infinite propositional problems over Σ^*. We proceed by assigning labels to the clauses of $\mathcal{T}$ such that a *labeled clause* represents up to infinitely many *standard clauses* over Σ^*. Then an inference, such as resolution, performed between labeled clauses corresponds to infinitely many inferences on the level of Σ^*. This is similar to the idea of "lifting" from first-order theorem proving where clauses with variables represent up to infinitely many ground instances. Here, however, we deal with the additional dimension of performing infinitely many reasoning tasks on the "ground level" in parallel, one for each rank (K, L).

[5] Given $W^* : \Sigma^* \to \{\mathbf{0}, \mathbf{1}\}$, the corresponding interpretation $(W_i)_{i\in\mathbb{N}} : \mathbb{N} \times \Sigma \to \{\mathbf{0}, \mathbf{1}\}$ is defined by the equation $W_i(p) = W^*(p^{(i)})$ for every $i \in \mathbb{N}$ and every $p \in \Sigma$.

Definition 3.2 A *label* is a triple $(b, k, l) \in \{*, 0\} \times (\{*\} \cup \mathbb{N}) \times \mathbb{N}$. A *labeled clause* $\mathcal{C}$ is a pair $(b, k, l) \,||\, C$ consisting of a label and a standard clause over Σ^*.[6]

The three label components stand for three independent conditions on the time indexes to which the clause relates. Intuitively, the first label component b relates the clause to the beginning of time and the second component relates the clause to the indexes of the form $K + i \cdot L$, where the goal should be satisfied. In both cases, $*$ stands for a "don't care" value, so if b or k equals $*$, the respective condition is trivially satisfied by any index. The third label component restricts the set of (K, L) models for which the clause is relevant by a certain divisibility condition in the values of L. It also has a default, "don't care" value corresponding to a vacuous restriction, namely $l = 0$. The precise statement of these conditions is postponed till Definition 3.4 below.

When a labeled clause set is constructed from a TST three particular label values are used:

Definition 3.3 Given a TST $\mathcal{T} = (\Sigma, I, T, G)$, the *initial labeled clause set* $N_{\mathcal{T}}$ for $\mathcal{T}$ is defined to contain

- labeled clauses of the form $(0, *, 0) \,||\, C$ for every $C \in I$,
- labeled clauses of the form $(*, *, 0) \,||\, C$ for every $C \in T$, and
- labeled clauses of the form $(*, 0, 0) \,||\, C$ for every $C \in G$.

Later on, new label values are computed from old ones using certain operations when labeled clauses interact in inferences, as will be detailed shortly. The full generality of labels reflects an entire "closure" of the above three initial values under these operations.

Semantics of labels and labeled clauses is given via a map to certain sets of time indexes.

Definition 3.4 Let a rank (K, L) be given. We define a set $R_{(K,L)}(b, k, l)$ of indexes *represented* by the label (b, k, l) as the set of all $t \in \mathbb{N}$ such that

1. $b \neq * \rightarrow t = 0$ and
2. $k \neq * \rightarrow \exists s \in \mathbb{N} \,.\, t + k = K + s \cdot L$ and
3. L divides l.

A standard clause of the form $C^{(t)}$ is said to be *represented by the labeled clause* $(b, k, l) \,||\, C$ *in* (K, L) if $t \in R_{(K,L)}(b, k, l)$.

Example Let us assume that a TST $\mathcal{T}$ contains a goal clause $(a \vee b) \in G$. In the initial labeled clause set $N_{\mathcal{T}}$ this goal clause becomes $(*, 0, 0) \,||\, a \vee b$. If we now, for example, fix $K = 2$ and $L = 3$ as in Fig. 1, our labeled clause will represent all the standard clauses $(a \vee b)^{(t)}$ with $t \in R_{(2,3)}(*, 0, 0) = \{2, 5, 8, \ldots\}$.

Notice that when computing $R_{(2,3)}(*, 0, 0)$, the condition on the first label component b is trivially satisfied because $b = *$. Also the condition on the third label component l is vacuously true, since any number $L \in \mathbb{N}^+$ divides 0. Thus we are looking for numbers t of the form $t = K + s \cdot L$ for some $s \in \mathbb{N}$. This gives us the set $R_{(2,3)}(*, 0, 0) = \{2, 5, 8, \ldots\}$.

The semantics of labels is chosen such that for any given rank (K, L) the standard clauses over Σ^* represented by the labeled clauses from the initial labeled clause set $N_{\mathcal{T}}$ form the previously described purely propositional problem that encodes the existence of a (K, L)-model of $\mathcal{T}$. That is the idea behind the following definition and lemma which capture soundness of the translation of a TST into an initial labeled clause set.

Definition 3.5 Let $N_{(K,L)} = \{C^{(t)} \mid (b, k, l) \,||\, C \in N \;\&\; t \in R_{(K,L)}(b, k, l)\}$ denote the set of standard clauses represented in (K, L) by the labeled clauses from N. A set of labeled clauses N is called (K, L)*-satisfiable* if there is a valuation $W^* : \Sigma^* \rightarrow \{0, 1\}$ which (propositionally) satisfies $N_{(K,L)}$. The set N is called *satisfiable* if it is (K, L)-satisfiable for some rank (K, L).

[6] The theory allows for the full generality of Σ^*. As discussed later, in practice we restrict our attention to clauses over $\Sigma \cup \Sigma'$.

Lemma 3.6 *Let $\mathcal{T}$ be a TST and $N_{\mathcal{T}}$ its initial labeled clause set. Then $\mathcal{T}$ is satisfiable if and only if $N_{\mathcal{T}}$ is. More precisely, for any rank (K, L) the initial labeled clause set $N_{\mathcal{T}}$ is (K, L)-satisfiable if and only if $\mathcal{T}$ has a (K, L)-model.*

To justify the semantics of the remaining values and how they relate to soundness, we will need to first describe the mentioned operations which act on labels during the resolution inference.

3.2 Label Merge

The ultimate goal of this section is to "lift" the classical resolution inference to labeled clauses. To achieve this goal we first need to describe how to combine the labels of the inference's premises to obtain the label of the inference's conclusion. This label is obtained using the merge operation.

Definition 3.7 The *merge* of labels (b_1, k_1, l_1) and (b_2, k_2, l_2) is a label (b, k, l) defined imperatively as follows:

- if $b_1 = *$ then $b \leftarrow b_2$ else if $b_2 = *$ then $b \leftarrow b_1$ else $b \leftarrow 0$,
- if $k_1 = *$ then $k \leftarrow k_2$ else if $k_2 = *$ then $k \leftarrow k_1$ else $k \leftarrow \min(k_1, k_2)$,
- if $k_1 = *$ or $k_2 = *$ then $l \leftarrow \gcd(l_1, l_2)$ else $l \leftarrow \gcd(l_1, l_2, |k_1 - k_2|)$,

where gcd stands for the greatest common divisor operation and $\gcd(0, 0) = 0$.

Example Merge of $(*, 2, 0)$ and $(*, 5, 0)$ is $(*, 2, 3)$; we compute the minimum of the k components, and the greatest common divisor of their difference and the original l components. Merge of $(*, 2, 3)$ and $(*, 2, 3)$ is $(*, 2, 3)$; merge is, in fact, idempotent. Merge of $(*, 2, 3)$ and $(*, *, 0)$ is $(*, 2, 3)$; merge has, in fact, a neutral element $(*, *, 0)$. Merge of $(*, 2, 3)$ and $(0, 1, 4)$ is $(0, 1, 1)$.

The merge operation is defined such that the way it acts on labels corresponds to the intersection operation on the represented sets of indexes.

Lemma 3.8 *Let (b, k, l) be the merge of labels (b_1, k_1, l_1) and (b_2, k_2, l_2). Then for any rank (K, L)*

$$R_{(K,L)}(b, k, l) = R_{(K,L)}(b_1, k_1, l_1) \cap R_{(K,L)}(b_2, k_2, l_2).$$

Proof The proof is straightforward from the definitions. We check by case analysis that

$(b_1 \neq * \Rightarrow t = 0)$ and $(b_2 \neq * \Rightarrow t = 0)$ is equivalent to $(b \neq * \Rightarrow t = 0)$,

that

$(k_1 \neq * \Rightarrow \exists s \in \mathbb{N} \,.\, t + k = K + s \cdot L)$ and $(k_2 \neq * \Rightarrow \exists s \in \mathbb{N} \,.\, t + k = K + s \cdot L)$

is equivalent to

$(k \neq * \Rightarrow \exists s \in \mathbb{N} \,.\, t + k = K + s \cdot L)$,

provided ($k_1 = *$ or $k_2 = *$ or $k_1, k_2 \in \mathbb{N}$ and L divides $|k_1 - k_2|$), and, finally, that

(L divides l_1) and (L divides l_2) is equivalent to (L divides $\gcd(l_1, l_2)$).

The only case that now needs additional explanation is when both $k_1, k_2 \in \mathbb{N}$ and $k_1 \neq k_2$. To explore this case, let us without loss of generality pick $k_1 < k_2$. We obtain for an index t represented by both labels that $t = K + s_1 \cdot L - k_1 = K + s_2 \cdot L - k_2$ for some $s_1, s_2 \in \mathbb{N}$, which implies $k_2 - k_1 = (s_2 - s_1) \cdot L$. Hence the extra divisibility condition which propagates to the third label component l (taking the absolute value of the difference $k_1 - k_2$ in Definition 3.7 is just cosmetics). Notice, on the other hand, that if L divides $k_2 - k_1$ then any index t of the form $t = K + s_1 \cdot L - k_1$ can be also expressed as $t = K + s_2 \cdot L - k_2$ (taking s_2 as $k_2 - k_1$ divided by L), but not necessarily vice versa, when $k_1 < k_2$. That is why the new value of the second label component k equals $\min(k_1, k_2)$. See also the accompanying Fig. 2 to follow the argument. □

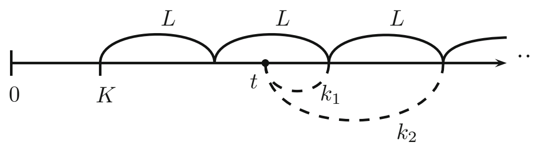

Fig. 2 A situation in which an index t is represented by two labels (b_1, k_1, l_1) and (b_2, k_2, l_2) such that $k_1, k_2 \in \mathbb{N}$ and $k_1 \neq k_2$. Such an overlap can happen if and only if L divides $|k_1 - k_2|$

3.3 Temporal Shift

The second operation on labels we need is temporal shift. Its intuitive role is to align one labeled clause with another such that the variable over which we subsequently plan to resolve the two clauses occurs in both of them under the same number of primes.

We define the temporal shift operation on labeled clauses by the following two equations

$$TS(\ (*, *, l) \,||\, C\ ) = (*, *, l) \,||\, (C)', \tag{3.1}$$

$$TS(\ (*, k, l) \,||\, C\ ) = (*, k+1, l) \,||\, (C)'. \tag{3.2}$$

Note that the operation is *undefined* for labeled clauses with the first component $b = 0$, because these only represent standard clauses fixed to the time index 0.

Soundness of time shift is the statement that all the standard clauses represented by the right hand side of (3.1) and (3.2) are also represented by the respective left hand sides in any rank (K, L).[7]

Lemma 3.9 *Let (K, L) be a rank and let $\mathcal{D} = TS(\mathcal{C})$ for some labeled clause $\mathcal{C} = (*, k, l) \,||\, C$, $k \in (\{*\} \cup \mathbb{N})$. Then any standard clause represented in (K, L) by $\mathcal{D}$ is represented in (K, L) by $\mathcal{C}$.*

Proof Let $(C')^{(t)}$ be a standard clause represented in (K, L) by $\mathcal{D} = (*, k', l) \,||\, (C)'$. This means that $t \in R_{(K,L)}(*, k', l)$. We either have $k = *$ and $k' = *$ or $k \in \mathbb{N}$ and $k' = k + 1$. In any case $t + 1 \in R_{(K,L)}(*, k, l)$, and thus $C^{(t+1)} = (C')^{(t)}$ is represented in (K, L) by $\mathcal{C} = (*, k, l) \,||\, C$. □

3.4 Labeled Resolution

We finally arrive to defining the labeled resolution operation. Because we in general allow arbitrarily many primes in labeled clauses, the operation involves iterated application of temporal shift.

Definition 3.10 Let $\mathcal{C}_1 = (b_1, k_1, l_1) \,||\, p^{(i)} \vee C_1$ and $\mathcal{C}_2 = (b_2, k_2, l_2) \,||\, \neg p^{(j)} \vee C_2$ be two labeled clauses. Their labeled resolvent over the variable p, denoted $\mathcal{C}_1 \otimes_p \mathcal{C}_2$, is computed as follows:

1. If $i = j$ then $\mathcal{C}_1 \otimes_p \mathcal{C}_2$ equals the labeled clause $(b, k, l) \,||\, C_1 \vee C_2$ where (b, k, l) is the merge of the labels (b_1, k_1, l_1) and (b_2, k_2, l_2).
2. If $i < j$ and $b_1 = *$ then $\mathcal{C}_1 \otimes_p \mathcal{C}_2$ equals $TS^{j-i}(\mathcal{C}_1) \otimes_p \mathcal{C}_2$, where TS^n stands for the n-fold application of the temporal shift operation. This reduces the computation to the previous case.
3. If $i > j$ and $b_2 = *$ then $\mathcal{C}_1 \otimes_p \mathcal{C}_2$ equals $\mathcal{C}_1 \otimes_p TS^{i-j}(\mathcal{C}_2)$; analogously to case 2.
4. If either $i < j$ and $b_1 = 0$ or $i > j$ and $b_2 = 0$ then the resolvent is undefined.

Strictly speaking, the resolvent $\mathcal{C}_1 \otimes_p \mathcal{C}_2$ does not depend just on the variable p, but more specifically on the occurrences of the literals $p^{(i)}$ and $\neg p^{(j)}$ in the respective clauses. We will, however, only use Definition 3.10 in situations where there is just one occurrence of a literal mentioning the variable p (possibly primed) in each of the two clauses, and therefore, confusion will not arise.

[7] The converse does not hold. For instance, the labeled clause $(*, *, 0) \,||\, p' = TS(\ (*, *, 0) \,||\, p\ )$ does not represent the standard clause $p^{(0)}$ in any rank (K, L), although the original clause $(*, *, 0) \,||\, p$ does.

Example Let two labeled clauses $(*, 0, 0) \,||\, \neg p \vee q$ and $(*, 0, 0) \,||\, r \vee p'$ be given. They cannot directly (as in case 1 above) generate a labeled resolvent, although in $(K, L) = (0, 1)$ there are (for every t) standard clauses $\neg p^{(t+1)} \vee q^{(t+1)}$ and $r^{(t)} \vee p^{(t+1)}$ represented, respectively, by the two labeled clauses, which resolve on $p^{(t+1)}$. The first labeled clause first needs to be shifted to $(*, 1, 0) \,||\, \neg p' \vee q'$ (case 2), and the clauses then resolve on p' and a labeled resolvent $(*, 0, 1) \,||\, r \vee q'$ is obtained.

Labeled resolution operation is sound in the following sense.

Lemma 3.11 *Let (K, L) be a rank. Any standard clause represented in (K, L) by the labeled resolvent $\mathcal{C}_1 \otimes_p \mathcal{C}_2$ of clauses $\mathcal{C}_1$ and $\mathcal{C}_2$ is a propositional resolvent over some "instance" $p^{(i)}$ of the variable p of clauses represented in (K, L) by $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively.*

Proof Follows from Lemmas 3.8 and 3.9. □

More interestingly, labeled resolution operation is also designed to be complete, i.e. to lift every possible propositional resolution from the "ground level".

Lemma 3.12 *Let $\mathcal{C}_1 = (b_1, k_1, l_1) \,||\, p^{(i)} \vee C_1$ and $\mathcal{C}_2 = (b_2, k_2, l_2) \,||\, \neg p^{(j)} \vee C_2$ be two labeled clauses and let (K, L) be a rank. If there are numbers $n, t_1, t_2 \in \mathbb{N}$ such that $n = i + t_1 = j + t_2$ and the clauses $p^{(n)} \vee (C_1)^{(t_1)}$ and $\neg p^{(n)} \vee (C_2)^{(t_2)}$ are represented in (K, L) by $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, then the labeled resolvent $\mathcal{C} = \mathcal{C}_1 \otimes_p \mathcal{C}_2$ is defined and the propositional resolvent $(C_1)^{(t_1)} \vee (C_2)^{(t_2)}$ over $p^{(n)}$ is represented in (K, L) by $\mathcal{C}$.*

Proof The labeled resolvent is only undefined when the corresponding temporal shift operation would be undefined. Let us (without loss of generality) focus on one of the symmetrical cases when this happens and assume that $i < j$ and $b_1 = 0$. Because $b_1 = 0$ we must have $t_1 = 0$, but then $i + t_1 = i < j \leq j + t_2$ for any $t_2 \in \mathbb{N}$, and so there can be no $n = i + t_1 = j + t_2$. In other words, whenever there are numbers $n, t_1, t_2 \in \mathbb{N}$ such that $n = i + t_1 = j + t_2$ and the clauses $p^{(n)} \vee (C_1)^{(t_1)}$ and $\neg p^{(n)} \vee (C_2)^{(t_2)}$ are represented in (K, L) by $\mathcal{C}_1$ and $\mathcal{C}_2$, respectively, then the labeled resolvent $\mathcal{C} = \mathcal{C}_1 \otimes_p \mathcal{C}_2$ is defined.

To prove the second part of the lemma, we focus on case 2 of Definition 3.10. The remaining cases are simpler (case 1) or analogous (case 3). We, therefore, assume that $i < j$ and $b_1 = *$. The clause $TS^{j-i}(\mathcal{C}_1)$ must be of the form $(b_1, k_1', l_1) \,||\, p^{(j)} \vee (C_1)^{(j-i)}$, where either $k_1' = k_1 = *$ or $k_1' = k_1 + (j - i)$. In both cases, by Definition 3.4

$$t_1 \in R_{(K,L)}(b_1, k_1, l_1) \text{ implies } t_2 = t_1 - (j - i) \in R_{(K,L)}(b_1, k_1', l_1).$$

Thus, by Lemma 3.8 $t_2 \in R_{(K,L)}(b, k, l)$, where (b, k, l) is the merge of labels (b_1, k_1', l_1) and (b_2, k_2, l_2), i.e. the label of the labeled resolvent $\mathcal{C}_1 \otimes_p \mathcal{C}_2 = \mathcal{C}$. Thus $\mathcal{C}$, which is of the form $(b, k, l) \,||\, (C_1)^{(j-i)} \vee C_2$, in (K, L) represents

$$((C_1)^{(j-i)} \vee C_2)^{(t_2)} = (C_1)^{(j-i)+t_2} \vee (C_2)^{(t_2)} = (C_1)^{(t_1)} \vee (C_2)^{(t_2)}.$$

□

Remark 3.13 The way we package the temporal shift operation with resolution is reminiscent to the use of most general unifiers in first-order theorem proving. Indeed, we can understand temporal shift as a form substitution on time indexes.

There is a straightforward encoding of LTL formulas and, in particular, of TSTs into first-order logic. The encoding models time by first-order variables understood to range over the natural numbers. On the syntax level, we have the constant 0 for the initial time point and the successor function s for modeling the single step from the current time point X to the next $s(X)$. Also, each signature symbol $p \in \Sigma$ is translated to a monadic predicate $p(X)$, parameterized by the time point where it is supposed to hold. Table 1 shows three labeled clauses translated this way.

In the light of this translation, the fact that temporal shift cannot be applied to initial clauses corresponds to non-unifiability of the constant 0 with any term starting with the successor function s: that is the reason why clauses

Table 1 Example labeled clauses translated to first-order logic

	Labeled clause	First-order clause
(1)	$(0, *, 0) \mid\mid \neg p \vee q$	$\neg p(0) \vee q(0)$
(2)	$(*, *, 0) \mid\mid p' \vee r'$	$\forall X.\ p(s(X)) \vee r(s(X))$
(3)	$(*, *, 0) \mid\mid \neg r \vee q$	$\forall Y.\ \neg r(Y) \vee q(Y)$

(1) and (2) in Table 1 cannot be unified and resolved on p. The successful use of temporal shift corresponds to a substitution: we can substitute $Y \mapsto s(X)$ to unify clauses (2) and (3) of Table 1 and resolve them. Lemma 3.12 effectively states that the substitutions corresponding to temporal shift operation employed in Definition 3.10 are, in fact, most general unifiers.

4 Elimination

By variable and clause elimination we understand the preprocessing technique described in [5] for simplifying propositional SAT problems. It consists of a combination of a controlled version of variable elimination and of the subsumption reduction[8] for removing clauses, as described below. These two are alternated in a saturation loop until no further immediate improvement is possible. In this section we show how the mechanism of labeled clauses can be used to adapt variable and clause elimination to the context of LTL.

Propositional variable elimination relies on *exhaustive* application of the resolution inference rule. Given (standard) clauses $C = p \vee C_0$ and $D = \neg p \vee D_0$, their standard resolvent $C \otimes D$ is $C_0 \vee D_0$. Now, given a propositional problem in CNF consisting of a set of clauses N and a variable p, one separates N into three disjoint subsets $N = N_p \cup N_{\neg p} \cup N_0$ of clauses. The first set, N_p, is a set of clauses containing the variable p positively, the clauses from $N_{\neg p}$ contain p negatively, and N_0 is a set of clauses without variable p. A new clause set $\overline{N}$ is obtained as $(N_p \otimes N_{\neg p}) \cup N_0$, where $N_p \otimes N_{\neg p} = \{C \otimes D \mid C \in N_p, D \in N_{\neg p}\}$. The set $\overline{N}$ no longer contains the variable p and is satisfiable if and only if N is.

The obtained set $\overline{N}$ may contain tautological clauses,[9] which are redundant and should be removed. Then the sizes of N and $\overline{N}$ are compared. In general, eliminating a single variable may incur a quadratic blowup. An elimination step is only considered an *improvement* and should be committed to when the size of $\overline{N}$ is not greater than that of N (possibly up to an additive constant). It is shown in [5] that improvement eliminations occur often in practice and that they can be used to simplify the input formula considerably.

4.1 Lifting Variable Elimination to Labeled Clauses

Let us now turn to eliminating variables from TSTs. We know that TSTs naturally correspond to sets of labeled clauses and these in turn represent propositional problems (albeit, in general, infinite ones) from which variables can be eliminated by the standard procedure described above. There is still a complication, however, because a single variable $p \in \Sigma$ from the TST corresponds to all its "instances" $p, p', p^{(2)}, \ldots$ on the "ground level" of the signature Σ^*. To be able to represent the result after elimination, all these instances need to be eliminated from the ground level uniformly, in one step. This seems to be a difficult task when the TST contains a clause that mentions the variable p in two different time contexts, like, for example, in $\neg p \vee q \vee p'$. In this case the individual eliminations cannot be done independently from each other and we rule the case out from further considerations.

[8] A standard clause C *subsumes* a clause D, if C's literals are a subset of D's literals. Subsumed clauses are redundant and can be discarded.

[9] A tautological clause contains both a variable and its negation.

Remark 4.1 There are some interesting subcases where eliminating such a variable would, in theory, be possible and would yield useful results. Consider the SNF containing p, $\Box(\neg p \vee p')$, $\Box(\neg p \vee r)$, from which p can be "semantically" eliminated and one obtains $\Box r$. On the other hand, eliminating p from the SNF containing p, $\Box(\neg p \vee \neg p')$, $\Box(p \vee p')$, $\Box(\neg p \vee a)$ should give us a formula whose models $(W_i)_{i \in \mathbb{N}}$ satisfy the condition ($i \bmod 2 = 0 \Rightarrow W_i \models a$), which is a property known [21] not to be expressible by an LTL formula over the single variable a.

Let us now, therefore, assume that we are given a set of labeled clauses N, perhaps an initial labeled clause set for a TST $\mathcal{T}$, and a variable $p \in \Sigma$ such that no clause in N contains more than one possibly primed occurrence of p. We separate N into $N_p \cup N_{\neg p} \cup N_0$, a subset containing p positively (possibly primed), a subset containing p negatively (possibly primed), and a subset not containing p at all. A new set of labeled clauses $\overline{N}$ is constructed as $(N_p \otimes_p N_{\neg p}) \cup N_0$, where $N_p \otimes_p N_{\neg p} = \{\mathcal{C}_1 \otimes_p \mathcal{C}_2 \mid \mathcal{C}_1 \in N_p, \mathcal{C}_2 \in N_{\neg p}\}$ stands for the set of all the labeled resolvents (Definition 3.10) over the variable p between labeled clauses from N_p and $N_{\neg p}$, respectively.

Example Let us assume that a set N contains the following labeled clauses

$$(0, *, 0) \,||\, p \vee q \vee r, \tag{4.1}$$

$$(0, *, 0) \,||\, \neg p \vee \neg r, \tag{4.2}$$

$$(*, *, 0) \,||\, r \vee \neg p', \tag{4.3}$$

$$(*, 0, 0) \,||\, \neg p \vee q, \tag{4.4}$$

and these are the only labeled clauses of N mentioning the variable p. Then eliminating p from N means removing the above labeled clauses and replacing them by all the possible labeled resolvents over p. Notice that, actually,

- the tautology (4.1) $\otimes_p$ (4.2) $= (0, *, 0) \,||\, q \vee r \vee \neg r$ can be immediately dropped,
- and (4.1) $\otimes_p$ (4.3) is undefined, because temporal shift does not apply to (4.1).

Thus we replace in N the above four clauses by the only nontrivial resolvent (4.1) $\otimes_p$ (4.4) $= (0, 0, 0) \,||\, q \vee r$.

We now show that variable elimination preserves satisfiability of the given clause set.

Theorem 4.2 *Let $N = N_p \cup N_{\neg p} \cup N_0$ and $\overline{N} = (N_p \otimes_p N_{\neg p}) \cup N_0$ be sets of labeled clauses as described above. Then N is (K, L)-satisfiable if and only if $\overline{N}$ is.*

Proof Let us first assume that N is (K, L)-satisfiable. This means there is a valuation $V : \Sigma^* \to \{0, 1\}$ such that $V \models N_{(K,L)}$. In order to show that $\overline{N}$ is (K, L)-satisfiable, we construct a new valuation $\overline{V} : (\overline{\Sigma})^* \to \{\mathbf{0}, \mathbf{1}\}$, where $\overline{\Sigma} = \Sigma \backslash \{p\}$ is the reduced basic signature. Similarly to the propositional case, we do this by simply forgetting the value of eliminated variable's instances:

$$\overline{V}(q^{(i)}) = V(q^{(i)})$$

for every $q \in \overline{\Sigma}$ and every $i \in \mathbb{N}$. Now we need to show that $\overline{V} \models \overline{N}_{(K,L)}$.

Because the variable p does not occur in the clauses of N_0, we have directly that $\overline{V} \models (N_0)_{(K,L)}$. Let us now take a clause $E \in (N_p \otimes_p N_{\neg p})_{(K,L)}$. By Lemma 3.11, E must be of the form $C \vee D$ for some $p^{(i)} \vee C \in (N_p)_{(K,L)}$ and $\neg p^{(i)} \vee D \in (N_{\neg p})_{(K,L)}$. Because these two clauses are true in V by assumption, their resolvent E is true in V by soundness of propositional resolution. Because the variable p does not occur in E, the clause is also true in $\overline{V}$.

To show the opposite direction let us assume that the set $\overline{N}$ is (K, L)-satisfiable, i.e., that there is a valuation $\overline{V} : (\overline{\Sigma})^* \to \{\mathbf{0}, \mathbf{1}\}$ such that $\overline{V} \models \overline{N}_{(K,L)}$. We extend $\overline{V}$ to a valuation V over Σ^* by defining for every $i \in \mathbb{N}$

$$V(p^{(i)}) = \mathbf{1} \text{ if and only if there is a clause } p^{(i)} \vee C \in (N_p)_{(K,L)} \text{ such that } \overline{V} \not\models C.$$

Note that the definition is correct, because by our restriction on variable elimination no instance of p occurs in the clause C above. Next we show that $V \models N_{(K,L)}$.

The fact that $V \models (N_0)_{(K,L)}$ follows again trivially from the assumption. Also, from the way we extended $\overline{V}$ to V, we have $V \models (N_p)_{(K,L)}$. Let us prove by contradiction that also $V \models (N_{\neg p})_{(K,L)}$. Assume there is a clause $\neg p^{(i)} \vee D \in (N_{\neg p})_{(K,L)}$ false in V. This means $V(p^{(i)}) = \mathbf{1}$ and, therefore, there is another clause $p^{(i)} \vee C \in (N_p)_{(K,L)}$ such that C is false in $\overline{V}$. We must have that the propositional resolvent $C \vee D$ is false in $\overline{V}$. But by Lemma 3.12, we have $C \vee D \in (N_p \otimes_p N_{\neg p})_{(K,L)}$. A contradiction. □

Apart from the previously explained restriction, there is another limitation on practical variable elimination. Consider a clause set containing the following two labeled clauses:

$(*, *, 0) \,||\, \neg x \vee p'$ and $(*, *, 0) \,||\, \neg p \vee y'$.

Eliminating p from the set would yield (possibly among other clauses) the labeled clause

$(*, *, 0) \,||\, \neg x \vee y''$.

This could be a useful simplification in some contexts, but notice that it got us outside SNF and TSTs, because y now occurs doubly primed.

Although we normally avoid eliminating variables like p above, there is, nevertheless, an advantage in knowing that such a step has a proper meaning and can be performed. If the problematic resolvent could be, for instance, shown redundant in the clause set (e.g. by subsumption), it would be sound to remove it and the desired syntactic simplicity of the clause set would be preserved.

4.2 Eliminating Labeled Clauses by Subsumption

Let us now turn to reductions and in particular to showing how to extend subsumption to work with labels. To lift propositional subsumption, we use the same idea as with resolution: any standard clause represented by the subsumed labeled clause must be subsumed by a standard clause represented by the subsuming labeled clause. This is achieved by the following:

Definition 4.3 A labeled clause $(b_1, k_1, l_1) \,||\, C$ *subsumes* a labeled clause $(b_2, k_2, l_2) \,||\, D$, if C subsumes D (i.e., $C \subset D$) and the merge of the labels (b_1, k_1, l_1) and (b_2, k_2, l_2) is equal to (b_2, k_2, l_2).

In analogy to resolution, the subsumption relation on labeled clauses can be made stronger if we allow the subsuming clause (but not the subsumed one) to be potentially shifted in time. For example, the clause $(*, *, 0) \,||\, q$ subsumes $(*, 1, 0) \,||\, p \vee q'$ in this sense. On the other hand, the clause $(*, *, 0) \,||\, q'$ cannot subsume $(*, *, 0) \,||\, p \vee q$, because there is a standard clause represented by the latter, namely $(p \vee q)^{(0)} = p \vee q$, which is not subsumed by any standard clause represented by the former.

The following theorem states soundness of labeled clause elimination by subsumption.

Theorem 4.4 *Let N and $\widetilde{N}$ be sets of labeled clauses, such that $\widetilde{N} \subseteq N$ and for every $\mathcal{D} \in N \backslash \widetilde{N}$ there exists $\mathcal{C} \in \widetilde{N}$ such that $\mathcal{C}$ subsumes $\mathcal{D}$. Then N is (K, L)-satisfiable if and only if $\widetilde{N}$ is.*

Proof As in the propositional case, a subsumed clause can be removed, because the constraint it imposes on the possible valuations is implied by that of the present subsuming clause. □

4.3 Elimination of "Exotic" Labels

We know that only the clauses labeled by $(0, *, 0)$, $(*, *, 0)$ and $(*, 0, 0)$, which are the labels used in the definition of the starting labeled clause set, directly correspond to initial, step and goal clauses of a TST, respectively. When clauses with other labels arise during elimination, the subsequent procedure for deciding satisfiability of the resulting set needs to know how to deal with them. Interestingly, according to the theorem below, we may drop several kinds of labeled clauses just after they are created without affecting satisfiability of the clause set.

We will need a simple lemma, which follows directly from the definitions.

Lemma 4.5 *Let (K, L) be an arbitrary rank, $i \in \mathbb{N}$, and $j \in \mathbb{N}^+$. Then $(K + i \cdot L, j \cdot L)$ is a rank and for any label (b, k, l) we have $R_{(K+i\cdot L, j\cdot L)}(b, k, l) \subseteq R_{(K,L)}(b, k, l)$.*

Theorem 4.6 *Let N be a finite set of labeled clauses and let N^- be a subset of N obtained be removing all the clauses with a label (b, k, l) such that either ($b = 0$ and $k \neq *$) or ($l \neq 0$). Then N^- is satisfiable if and only if N is.*

Proof One implication is trivial since $N^- \subseteq N$. For the other implication, we need an auxiliary definition. We say that a label (b, k, l) is *relevant* for a rank (K, L) if $R_{(K,L)}(b, k, l) \neq \emptyset$. Let us now assume that N^- is (K_0, L_0)-satisfiable, i.e. that there is a valuation V such that $V \models (N^-)_{(K_0,L_0)}$. We may choose K_1 of the form $K_0 + i \cdot L_0$ and L_1 of the form $j \cdot L_0$ large enough such that none of the removed clauses, i.e. none of the clauses from $N \backslash N^-$, is relevant for the rank (K_1, L_1). This is possible, because a clause with a label satisfying ($b = 0$ and $k \neq *$) is only relevant for a rank (K, L) if $k = K + s \cdot L$ for some $s \in \mathbb{N}$, and a clause with a label satisfying ($l \neq 0$) is only relevant for a rank (K, L) when L divides l. Moreover, there is only finitely many such clauses thanks to the assumption of the theorem. If we now write

$$N_{(K_1,L_1)} = (N \backslash N^-)_{(K_1,L_1)} \cup (N^-)_{(K_1,L_1)},$$

we can observe that $(N \backslash N^-)_{(K_1,L_1)} = \emptyset$ by the choice of (K_1, L_1) and $(N^-)_{(K_1,L_1)} \subseteq (N^-)_{(K_0,L_0)}$ by Lemma 4.5. Therefore, $V \models N_{(K_1,L_1)}$ and so N is (K_1, L_1)-satisfiable. □

Example Deriving an empty labeled clause during elimination does not immediately imply that the current clause set is unsatisfiable. For instance, the label of the empty clause $(*, 0, 2) \,||\, \bot$ is only relevant for (K, L) when L divides 2 and thus the current clause set may still be (K, L)-satisfiable for $L > 2$. Moreover, thanks to Lemma 4.5 we can always avoid dealing with the problematic ranks for which the empty clause is relevant by "typecasting" a potential model to a higher rank.

After filtering a clause set with the help of Theorem 4.6, it will only contain clauses with the familiar labels of the starting labeled clause set and possibly also clauses labeled by $(*, k, 0)$, $k \in \mathbb{N}$. These do not pose any further complications, because they arise naturally in our calculus LPSup [18] for LTL satisfiability.

4.4 The Elimination Procedure

We close this section by shortly discussing the overall variable and clause elimination procedure. As already mentioned, it is advantageous to alternate variable elimination attempts with exhaustive application of subsumption and possibly other reductions. That is because removing a subsumed clause may turn elimination of a particular variable into an improvement and, on the other hand, new clauses generated during elimination may be subject to subsumption. This holds true for the original SAT setting as it does with labels. A detailed description on how to efficiently organize this process can be found in [5].

5 Experimental Evaluation

For our evaluation of the effectiveness of variable and clause elimination in LTL, we extended the preprocessing capabilities of Minisat [6] version 2.2. We kept Minisat's main simplification loop, which efficiently combines variable elimination with subsumption and self-subsuming resolution, along with the fine-tuned heuristics for deciding which variables to eliminate and in what order. We emulated labels by extending respective clauses with extra *marking* literals[10] and, to ensure correctness, we disallowed elimination of variables that occur both primed and

[10] For example, any goal clause C is inserted as $C \vee g$, where g is a fresh variable designated for marking goal clauses.

non-primed in the input formula. Although this does not exploit the full potential of variable and clause elimination with labeled clauses as described in Sect. 4, we already obtained encouraging results with this setup.

For testing we used a set of LTL benchmarks collected by Schuppan and Darmawan [16]. The set consists of a total of 3723 problems from various sources (mostly previous papers on LTL satisfiability) and of various flavors (application, crafted, random), and represents the most comprehensive collection of LTL problems we are aware of. The testing proceeded in three stages. First, all the benchmarks were translated by our tool from the original format into TSTs. Then we applied the Minisat-based elimination tool and obtained a set of simplified TSTs. Finally, we ran two resolution-based LTL provers on both the original and simplified TSTs to measure the effect of simplification on prover runtime. We choose the LTL prover LS4 [20], most likely the strongest LTL solver[11] currently publicly available, and trp++ [10], a well established temporal resolution prover by Boris Konev. Having performed the experiments on two independent implementations should allow us to draw more general conclusions about the effects of variable and clause elimination.

The experiments were performed on our servers with 3.16 GHz Xeon CPU, 16 GB RAM, and Debian 6.0. All the tools along with intermediate files and experiment logs can be found at http://www.mpi-inf.mpg.de/~suda/vce.html.

We recorded for each problem the number of variables and clauses that we were able to eliminate during the second stage. We distinguished variables from the *original* problem and *auxiliary* variables that were introduced during the transformation in stage one. In total, 39 % of the variables (7 % original, 32 % auxiliary) and 32 % of the clauses were eliminated. The numbers vary greatly over individual subsets of the benchmarks. For example, the family `phltl` allowed for almost no simplification: only 3 % of the variables (just auxiliary), and 2 % of the clauses could be removed. On the other hand, 99 % of the variables (almost all of them original) and 98 % of the clauses were removed on the family `O1formula`. While the former extreme can be explained by a concise and already almost clausal structure of the original formulas from `phltl`, the latter follows from the fact that most of the variables in `O1formula` occur in just one polarity, i.e. are pure. Eliminating a pure variable amounts to removal of all the clauses in which the variable appears.[12]

The results of the third stage, in which we measured the effect of simplification on the performance of the two selected provers, are summarized in Table 2 and at the same time represented graphically in Fig. 3. We see that both LS4 and trp++ substantially benefit from the simplification, both in the number of solved instances and the overall runtime. On some subsets the effect is quite pronounced (see, e.g., LS4 on `alaska` or trp++ on `forobots`), while on others it is more modest. Only on the subset `trp` did the simplification result in less problems solved.[13] What the table does not show, however, is that even among the `trp` problems there were some only solved in the simplified form (16 such problems for LS4 and 9 for trp++). When judging the relative number of problems gained by each prover, it should be noted that many problems come from scalable families and are mostly trivial or too difficult to solve. This leaves the "gray zone" where improvement is possible relatively small.

To conclude, the results of our evaluation indicate that variable and clause elimination represents a useful preprocessing technique of TSTs. Simplifying a clause set not only removes redundancies introduced by a previous, potentially sub-optimal normal form transformation (when auxiliary variables get eliminated), but usually reduces the input even further. This ultimately decreases the time needed to solve the problem. Further improvements are expected from an independent implementation that will harness the full potential of the mechanism of labels.

---

[11] LS4 solves 3556 of the above benchmarks within the time limit of 60s; the best system reported by Schuppan and Darmawan [16], the bounded model checker of NuSMV 2.5, is able the solve 3368 of these benchmarks under the same conditions.

[12] If x is a pure variable (literal) then $N_{\neg x}$ is empty and so $N_x \otimes N_{\neg x}$ is empty as well.

[13] We currently do not have a better explanation for this phenomenon than that a seemingly innocuous change in the presentation of the problem caused by the elimination steers the provers to a different part of the search space. (These effects are common in theorem proving in general.) This may be partially supported by an observation that the "lost problems" are not shared by LS4 and trp++.

Table 2 Performance of the two provers on original (o) and simplified (s) problems, grouped by problem subset

Subset	Size		LS4		trp++	
			Solved	Time (s)	Solved	Time (s)
`acacia`	71	o	71	7.1	71	39.3
		s	71	7.1	71	11.3
`alaska`	140	o	121	6607.0	9	39,423.2
		s	139	882.0	12	38,717.5
`anzu`	111	o	93	5754.2	0	33,300.0
		s	94	5482.2	0	33,300.0
`forobots`	39	o	39	4.3	39	1198.8
		s	39	3.9	39	194.2
`rozier`	2320	o	2278	13,312.9	2063	96,293.7
		s	2278	13,270.7	2120	76,921.1
`schuppan`	72	o	41	9332.8	36	11,189.8
		s	41	9320.9	37	10,741.0
`trp`	970	o	940	12,327.5	364	189,045.2
		s	934	11,887.5	359	190,138.3
Total	3723	o	3583	47,345.8	2582	370,490.0
		s	3596	40,854.3	2638	350,023.4

Number of problems *solved* by each prover within the time limit 300 s and the overall *time* spent during the attempts are shown. Unsolved problems contribute 300.0 s, solved ones at least 0.1 s due to the measurement technique. The times spent on the actual simplification are not included; these were observed to be negligible for most of the problems, with maximum of 0.3 s for the largest instance

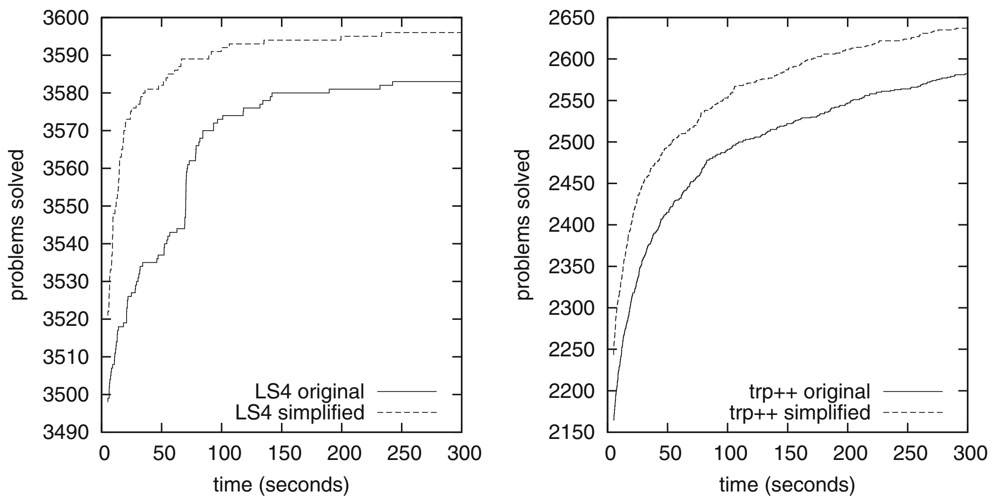

Fig. 3 Comparing the number of problems solved, simplified and original, within a given time limit. Although the value ranges for LS4 (on the *left*) and trp++ (on the *right*) differ, both figures demonstrate better performance on the simplified problems

6 Discussion

We are not aware of any related work directly focusing on simplifying clause normal forms for LTL. However, some interesting connections can be drawn with the help of Remark 2.3 of Sect. 2, which shows that a TST can be viewed as a symbolic representation of a Büchi automaton. For instance, in the classical paper [9], an automaton accepting the models of an LTL formula φ is constructed such that its states are identified with sets of φ's subformulas. A closer look reveals an immediate connection between these subformulas and the variables introduced to represent them in the SNF for φ. The above paper also suggests several improvements of the basic algorithm. For instance, it is advocated that subformulas of the form $\mu_1 \wedge \mu_2$ need not be stored, because the individual conjuncts μ_1 and μ_2 will be later added as well and they already imply the conjunction as a whole. We can restate this on the symbolic level as an observation that a variable introduced to represent a conjunctive subformula can always be eliminated, which is a claim easy to verify.

We believe this connection deserves further exploration, as one could possibly use it to bring some of the numerous techniques for optimizing explicit automata construction (see e.g. [14]) to the symbolic level. Note, however, that the main application of the explicit automata construction approach lies in model checking and so the resulting automaton is required to be *equivalent* to the original formula. On the other hand, our clausal symbolic approach is meant for satisfiability testing only and so more general *satisfiability preserving* transformations are allowed. An elimination of a variable from the original signature of the formula φ, or the "forgetting step" justified by Theorem 4.6, are examples of transformations that do not have a counterpart on the automata side.

While the explicit notion of a symbolic representation of a Büchi automaton via a clause normal form has received relatively little attention so far,[14] symbolic approaches to LTL model checking and satisfiability based on Binary Decision Diagrams are well known [2]. Again, it seems possible that some optimization techniques could be shared between the two approaches. For instance, different BDD encodings recently studied by Rozier and Vardi [15], could correspond to different ways of turning a formula into a TST.

7 Conclusion

We have shown that variable and clause elimination, a practically successful preprocessing technique for propositional SAT problems, can be adapted to the setting of linear temporal logic. For that purpose we have utilized the mechanism of labeled clauses, a method for interpreting an LTL formula as finitely represented infinite sets of standard propositional clauses. The ideas were implemented and tested on a comprehensive set of benchmarks with encouraging results. In particular, variable and clause elimination has been shown to significantly improve subsequent runtime of resolution-based provers LS4 and trp++.

We would like to stress here that labeled clauses provide a general method for transferring resolution-based reasoning from SAT to LTL. It is therefore plausible that other preprocessing techniques, like, for example, the blocked clause elimination [11], can be adapted along the same lines. Exploring this possibility will be one of the directions for future work.

Appendix A. LTL Preliminaries

The language of linear temporal logic (LTL) formulas is an extension of the propositional language with temporal operators. The most commonly used are Next $\bigcirc$, Always $\Box$, Eventually $\Diamond$, Until U, and Release R. Formally, let $\Sigma = \{p, q, \ldots\}$ be a (finite) signature of propositional variables, then the set of LTL formulas is defined inductively as follows:

- any $p \in \Sigma$ is a formula,
- if φ and ψ are formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \vee \psi$,
- if φ and ψ are formulas, then so are $\bigcirc\varphi$, $\Box\varphi$, $\Diamond\varphi$, $\varphi\mathsf{U}\psi$, and $\varphi\mathsf{R}\psi$.

A propositional valuation, or simply a *state*, is a mapping $W : \Sigma \to \{0, 1\}$. An *interpretation* for an LTL formula is an infinite sequence of states $\mathcal{W} = (W_i)_{i\in\mathbb{N}}$. The truth relation $\mathcal{W}, i \models \varphi$ between an interpretation $\mathcal{W}$, time index $i \in \mathbb{N}$, and a formula φ is defined recursively as follows:

[14] A correspondence between SNF and Büchi automata has been shown in [1]. The relevant theorem of the paper, however, does not establish an equivalence between models of the formula and accepting runs of the automaton. Its value for translating techniques between the symbolic and explicit approaches is, therefore, limited.

$\mathcal{W}, i \models p$ iff $W_i \models p$,
$\mathcal{W}, i \models \neg\varphi$ iff not $\mathcal{W}, i \models \varphi$,
$\mathcal{W}, i \models \varphi \wedge \psi$ iff $\mathcal{W}, i \models \varphi$ and $\mathcal{W}, i \models \psi$,
$\mathcal{W}, i \models \varphi \vee \psi$ iff $\mathcal{W}, i \models \varphi$ or $\mathcal{W}, i \models \psi$,
$\mathcal{W}, i \models \bigcirc\varphi$ iff $\mathcal{W}, i+1 \models \varphi$,
$\mathcal{W}, i \models \Box\varphi$ iff for every $j \geq i$, $\mathcal{W}, j \models \varphi$,
$\mathcal{W}, i \models \Diamond\varphi$ iff for some $j \geq i$, $\mathcal{W}, j \models \varphi$,
$\mathcal{W}, i \models \varphi \mathsf{U} \psi$ iff there is $j \geq i$ such that $\mathcal{W}, j \models \psi$ and $\mathcal{W}, k \models \varphi$ for every k, $i \leq k < j$,
$\mathcal{W}, i \models \varphi \mathsf{R} \psi$ iff for all $j \geq i$, $\mathcal{W}, j \models \psi$ or there is $j \geq i$ with $\mathcal{W}, j \models \varphi$ and for all k, $i \leq k \leq j$, $\mathcal{W}, k \models \psi$.

An interpretation $\mathcal{W}$ is a *model* of an LTL formula φ if $\mathcal{W}, 0 \models \varphi$. A formula φ is called *satisfiable* if it has a model, and is called *valid* if every interpretation is a model of φ.

Appendix B. Transforming LTL Formulas to SNF

Formulas in SNF are conjunctions of *temporal clauses*, each of them assuming one of the following forms:

- an *initial* clause: $\bigvee_j k_j$,
- a *step* clause: $\Box(\bigvee_j k_j \vee \bigvee_j \bigcirc l_j)$,
- an *eventuality* clause: $\Box(\bigvee_j k_j \vee \Diamond l)$,

where k_j, l_j, and l stand for standard literals, i.e. propositional variables or their negation.

The translation of an LTL formula φ into an equisatisfiable SNF starts by first turning φ into an equivalent formula that is in negation normal form (NNF), meaning the negation sign only occurs in front of propositional variables in the leaves of the formula tree. This can be achieved by a standard operation that "pushes negations downwards" with the help of De Morgan's rules and temporal equivalences like $\neg\bigcirc\varphi \equiv \bigcirc\neg\varphi$, $\neg\Box\varphi \equiv \Diamond\neg\varphi$, and $\neg(\varphi \mathsf{U} \psi) \equiv (\neg\varphi)\mathsf{R}(\neg\psi)$. Finally, multiple negations are absorbed with the help of the classical equivalence $\neg\neg\varphi \equiv \varphi$. In what follows we assume that φ is already in NNF.

The actual transformation is performed with the help of operator τ defined in Fig. 4, which recursively reduces any formula of the form $\Box(\neg x \vee \varphi)$, where x is a propositional atom, into the final SNF. During the process, new "fresh" variables are being introduced (we typeset them in bold) which serve two different purposes: They stand as names for subformulas (as in the case of the rules for, e.g., conjunction), and may also play a role of "trackers" that influence the value of other variables not just in the current state, but also in those to follow. This is how the semantics of, e.g., the Always operator $\Box$ is being encoded. The overall translation is triggered by the following rule

$$\varphi \quad \longrightarrow \quad \mathbf{i} \wedge \tau[\Box(\neg\mathbf{i} \vee \varphi)] \ ,$$

with a fresh variable $\mathbf{i}$ that represents the whole formula.

$$\begin{array}{rrcl}
1. & \tau[\Box(\neg x \vee l)] & \longrightarrow & \Box(\neg x \vee l), \text{ if } l \text{ is a literal}, \\
2. & \tau[\Box(\neg x \vee (\varphi \wedge \psi))] & \longrightarrow & \tau[\Box(\neg x \vee \varphi)] \wedge \tau[\Box(\neg x \vee \psi)], \\
3. & \tau[\Box(\neg x \vee (\varphi \vee \psi))] & \longrightarrow & \Box(\neg x \vee \mathbf{u} \vee \mathbf{v}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)] \wedge \tau[\Box(\neg \mathbf{v} \vee \psi)], \\
4. & \tau[\Box(\neg x \vee \bigcirc\varphi)] & \longrightarrow & \Box(\neg x \vee \bigcirc\mathbf{u}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)], \\
5. & \tau[\Box(\neg x \vee \Box\varphi)] & \longrightarrow & \Box(\neg x \vee \mathbf{u}) \wedge \Box(\neg \mathbf{u} \vee \bigcirc\mathbf{u}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)], \\
6. & \tau[\Box(\neg x \vee \Diamond\varphi)] & \longrightarrow & \Box(\neg x \vee \Diamond\mathbf{u}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)], \\
7. & \tau[\Box(\neg x \vee (\varphi \mathsf{U} \psi)] & \longrightarrow & \Box(\neg x \vee \Diamond\mathbf{v}) \wedge \\
 & & & \Box(\neg x \vee \mathbf{v} \vee \mathbf{w}) \wedge \Box(\neg \mathbf{w} \vee \mathbf{u}) \wedge \Box(\neg \mathbf{w} \vee \bigcirc\mathbf{v} \vee \bigcirc\mathbf{w}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)] \wedge \tau[\Box(\neg \mathbf{v} \vee \psi)], \\
8. & \tau[\Box(\neg x \vee (\varphi \mathsf{R} \psi)] & \longrightarrow & \Box(\neg x \vee \mathbf{w}) \wedge \Box(\neg \mathbf{w} \vee \mathbf{v}) \wedge \Box(\neg \mathbf{w} \vee \mathbf{u} \vee \bigcirc\mathbf{w}) \wedge \\
 & & & \tau[\Box(\neg \mathbf{u} \vee \varphi)] \wedge \tau[\Box(\neg \mathbf{v} \vee \psi)],
\end{array}$$

Fig. 4 The rules for SNF transformation. The freshly introduced variables are in *bold*

Example Here we work out an example from [8] to demonstrate the translation procedure. Assume we would like to prove the formula $(\Diamond p \wedge \Box(p \rightarrow \bigcirc p)) \rightarrow \Diamond\Box p$. In refutational theorem proving we proceed by negating the formula and trying to show the negation to be unsatisfiable. By taking the negation into NNF (and translating away the implication symbol) we obtain

$$(\Diamond p \wedge \Box(\neg p \vee \bigcirc p)) \wedge \Box\Diamond\neg p,$$

which is consequently translated into the following set of clauses:

i	By the initial rule.
$\Box(\neg i \vee \Diamond u_1)$	The first conjunct by rule 6,
$\Box(\neg u_1 \vee p)$	terminates by rule 1.
$\Box(\neg i \vee u_2)$	
$\Box(\neg u_2 \vee \bigcirc u_2)$	The second conjunct by rule 5,
$\Box(\neg u_2 \vee u_3 \vee v_3)$	inside which there is disjunction (rule 3),
$\Box(\neg u_3 \vee \neg p)$	the first argument is a literal (rule 1),
$\Box(\neg v_3 \vee \bigcirc u_4)$	the second goes by rule 4
$\Box(\neg u_4 \vee p)$	and terminates by rule 1.
$\Box(\neg i \vee u_5)$	
$\Box(\neg u_5 \vee \bigcirc u_5)$	The third conjunct by rule 5,
$\Box(\neg u_5 \vee \Diamond u_6)$	inside which we apply rule 6,
$\Box(\neg u_6 \vee \neg p)$	and terminate by rule 1.

Notice that transformation τ introduces more new variables than would be strictly necessary. For example, the variable u_6 just "connects" the last two clauses, which could be replaced by one equivalent eventuality clause $\Box(\neg u_5 \vee \Diamond\neg p)$. This is a price we pay here for the simple statement of the transformation rules in Fig. 4 (no side conditions). An actual implementation would strive to detect the literal case as soon as possible, and thus, e.g., introduction of u_6 would be avoided.

Appendix C. Transforming General SNF to a TST

The transformation of general SNF to TSTs focuses on eventuality clauses. It consists in two simplification steps:

1. turning the *conditional* eventuality clauses into *unconditional* ones (of the form $\Box\Diamond l$),
2. reducing *multiple* (unconditional) eventuality clauses from the SNF into *just one* eventuality clause.

We present our modification of the simplifications first introduced in [4] that performs both steps at once.

Assume that an SNF of a formula contains n (in general) conditional eventuality clauses

$$\Box(C_i \vee \Diamond l_i)$$

for $i = 1, \ldots, n$, where C_i is the conditional part, i.e. a disjunction of literals. We remove these, and replace them with a single unconditional eventuality clause

$$\Box\Diamond\mathbf{m} \tag{C.1}$$

together with the following five step clauses for every $i = 1, \ldots, n$:

$$\Box\ (C_i \vee l_i \vee \mathbf{t_i}), \tag{C.2}$$
$$\Box\ (\neg\mathbf{t_i} \vee \bigcirc l_i \vee \bigcirc\mathbf{t_i}), \tag{C.3}$$
$$\Box\ (\mathbf{s_i} \vee \neg\mathbf{t_i} \vee \bigcirc\neg\mathbf{s_i}), \tag{C.4}$$
$$\Box\ (\neg\mathbf{s_i} \vee \neg\mathbf{m}), \tag{C.5}$$
$$\Box\ (\mathbf{s_i} \vee \bigcirc\neg\mathbf{m}), \tag{C.6}$$

where again the bold variables are supposed to be new to the formula.

The idea behind the simplification is the following: If the condition $\neg C_i$ is satisfied in the current state and the respective eventuality l_i is not satisfied in the same state we start "tracking" the eventuality with the help of the new variable $\mathbf{t_i}$ (clause C.2). The tracking variable $\mathbf{t_i}$ is forced to stay true also in the future states unless the eventuality l_i is finally satisfied (clause C.3). Now let us look from the other side. The unconditional eventuality (clause C.1) will infinitely often ensure that all the variables $\mathbf{s_i}$ are false in one state (clause C.5) and were true in the previous state (clause C.6). Thus in the intervals between states where $\mathbf{m}$ holds, there will always be two consecutive states where $\mathbf{s_i}$ changes from false to true. But this cannot happen if we are tracking that particular eventuality at that time (clause C.4). To sum up, for each of the original eventualities we have a guarantee that in every interval between states where $\mathbf{m}$ holds the eventuality was either not triggered at all ($\neg C_i$ was false in the whole interval) or the eventuality was triggered and subsequently satisfied in that interval. Please consult [4] for a formal proof.

Example Our previous example contained two conditional eventuality clauses $\Box(\neg i \vee \Diamond u_1)$ and $\Box(\neg u_5 \vee \Diamond u_6)$. We may replace these by the following set of clauses to obtain an equisatisfiable problem with just one unconditional eventuality clause:

$$\begin{gathered}
\Box\Diamond m,\\
\Box(\neg i \vee u_1 \vee t_1),\\
\Box(\neg t_1 \vee \bigcirc u_1 \vee \bigcirc t_1),\\
\Box(s_1 \vee \neg t_1 \vee \bigcirc \neg s_1),\\
\Box(\neg s_1 \vee \neg m),\\
\Box(s_1 \vee \bigcirc \neg m),\\
\Box(\neg u_5 \vee u_6 \vee t_2),\\
\Box(\neg t_2 \vee \bigcirc u_6 \vee \bigcirc t_2),\\
\Box(s_2 \vee \neg t_2 \vee \bigcirc \neg s_2),\\
\Box(\neg s_2 \vee \neg m),\\
\Box(s_2 \vee \bigcirc \neg m).
\end{gathered}$$

Acknowledgments The author wants to thank the anonymous reviewers for their useful suggestions, which greatly helped to improve the quality of the final version of this paper. This work was partly supported by Microsoft Research through its PhD Scholarship Programme.

References

1. Bolotov, A., Fisher, M., Dixon, C.: On the relationship between ω-automata and temporal logic normal forms. J. Logic Comput. **12**(4), 561–581 (2002)
2. Clarke, E.M., Grumberg, O., Hamaguchi, K.: Another look at LTL model checking. Form. Methods Syst. Des. **10**(1), 47–71 (1997)
3. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press, Cambridge (2001)
4. Degtyarev, A., Fisher, M., Konev, B.: A simplified clausal resolution procedure for propositional linear-time temporal logic. In: TABLEAUX ’02. LNCS, vol. 2381, pp. 85–99. Springer, Berlin (2002)
5. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: SAT’05. LNCS, vol. 3569, pp. 61–75. Springer, Berlin (2005)
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: SAT’03. LNCS, vol. 2919, pp. 502–518. Springer, Berlin (2003)
7. Fisher, M.: A resolution method for temporal logic. In: IJCAI’91, pp. 99–104. Morgan Kaufmann Publishers Inc., Menlo Park (1991)
8. Fisher, M., Dixon, C., Peim, M.: Clausal temporal resolution. ACM Trans. Comput. Logic **2**, 12–56 (2001)
9. Gerth, R., Peled, D., Vardi, M., Wolper, P.: Simple on-the-fly automatic verification of linear temporal logic. In: In Protocol Specification Testing and Verification, pp. 3–18. Chapman & Hall, London (1995)
10. Hustadt, U., Konev, B.: Trp++ 2.0: a temporal resolution prover. In: CADE-19. LNCS, vol. 2741, pp. 274–278. Springer, Berlin (2003)
11. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: TACAS. LNCS, vol. 6015, pp. 129–144. Springer, Berlin (2010)
12. Pill, I., Semprini, S., Cavada, R., Roveri, M., Bloem, R., Cimatti, A.: Formal analysis of hardware requirements. In: DAC ’06, pp. 821–826. ACM, New York (2006)
13. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE, Washington (1977)
14. Rozier, K., Vardi, M.: LTL satisfiability checking. In: 14th International SPIN Workshop. LNCS, vol. 4595, pp. 149–167. Springer, Berlin (2007)
15. Rozier, K., Vardi, M.: A multi-encoding approach for LTL symbolic satisfiability checking. In: FM. LNCS, vol. 6664, pp. 417–431. Springer, Berlin (2011)
16. Schuppan, V., Darmawan, L.: Evaluating LTL satisfiability solvers. In: ATVA’11. LNCS, vol. 6996, pp. 397–413. Springer, Berlin (2011)
17. Sistla, A.P., Clarke, E.M.: The complexity of propositional linear temporal logics. J. ACM **32**, 733–749 (1985)
18. Suda, M., Weidenbach, C.: Labelled superposition for PLTL. In: LPAR-18. LNCS, vol. 7180, pp. 391–405. Springer, Berlin (2012)
19. Suda, M., Weidenbach, C.: Labelled superposition for PLTL. Research Report MPI-I-2012-RG1-001, Max-Planck-Institut für Informatik, Saarbrücken (2012)
20. Suda, M., Weidenbach, C.: A PLTL-prover based on labelled superposition with partial model guidance. In: IJCAR. LNCS, vol. 7364, pp. 537–543. Springer, Berlin (2012)
21. Wolper, P.: Temporal logic can be more expressive. Inf. Control **56**(1/2), 72–99 (1983)