Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance

Viktor Schuppan
Viktor.Schuppan@gmx.de

LTL is frequently used to express specifications in many domains such as embedded systems or business processes. Witnesses can help to understand why an LTL specification is satisfiable, and a number of approaches exist to make understanding a witness easier. In the case of unsatisfiable specifications unsatisfiable cores (UCs), i.e., parts of an unsatisfiable formula that are themselves unsatisfiable, are a well established means for debugging. However, little work has been done to help understanding a UC of an unsatisfiable LTL formula. In this paper we suggest to enhance a UC of an unsatisfiable LTL formula with additional information about the time points at which the subformulas of the UC are relevant for unsatisfiability. For example, in $(\mathbf{G}p) \wedge (\mathbf{X}\neg p)$ the first occurrence of p is really only "relevant" for unsatisfiability at time point 1 (time starts at time point 0). We present a method to extract such information from the resolution graph of a temporal resolution proof of unsatisfiability of an LTL formula. We implement our method in TRP++, and we experimentally evaluate it. Source code of our tool is available.

1 Introduction

LTL and its relatives are important specification languages for reactive systems (e.g., [16]). Experience in verification (e.g., [7]) and in synthesis (e.g., [10]) has lead to specifications themselves becoming objects of analysis. Typically, a specification is expected to be satisfiable. If it turns out to be unsatisfiable, finding a reason for unsatisfiability can help with the ensuing debugging. Given the sizes of specifications of real world systems (e.g., [11]) automated support for determining a reason for unsatisfiability of a specification is crucial. Frequently, such reason for unsatisfiability is taken to be a part of the unsatisfiable specification that is by itself unsatisfiable; this is called an unsatisfiable core (UC) (e.g., [36, 5]).

Less simplistic ways to examine an LTL specification ϕ exist [29], and understanding their results also benefits from availability of UCs. First, one can ask whether a certain scenario ϕ', given as an LTL formula, is permitted by ϕ. That is the case iff $\phi \wedge \phi'$ is satisfiable. Second, one can check whether ϕ ensures a certain LTL property ϕ''. ϕ'' holds in ϕ iff $\phi \wedge \neg\phi''$ is unsatisfiable. In the first case, if the scenario turns out not to be permitted by the specification, a UC can help to understand which parts of the specification and the scenario are responsible for that. In the second case a UC can show which parts of the specification imply the property. Moreover, if there are parts of the property that are not part of the UC, then those parts of the property could be strengthened without invalidating the property in the specification; i.e., the property is vacuously satisfied (e.g., [7]).

Trying to help users to understand counterexamples in verification, which are essentially witnesses to a satisfiable formula, is a well established research topic (see, e.g., [6] for some references). In particular, it is common to add information to a counterexample on which parts of a counterexample are relevant at which points in time (e.g., [30, 6]). According to [6] such explanations are an integral part of every counterexample trace in IBM's verification platform RuleBase PE. Checks for vacuous specifications, which are closely related to UCs [36], are an important feature of industrial hardware verification tools (see, e.g., [7, 3]). In the academic world UCs are an important part of design methods for embedded

Preliminary Report. Final version to appear in:
QAPL 2013

systems (e.g., [29]) as well as for business processes (e.g., [4]). Despite this relevance of UCs efforts to provide additional information in the context of UCs or vacuity have remained isolated (e.g., [38]).

In this paper we suggest to enhance UCs for LTL with information on the time points at which its subformulas are relevant for unsatisfiability. As illustration we discuss the example from the abstract in more detail, shown in (1) again. When (1) is evaluated on some word π according to standard semantics of LTL (see Sec. 2), (1) and both of its conjuncts, $\mathbf{G}p$ and $\mathbf{X}\neg p$, are evaluated at time point 0 (time starts at time point 0), the operand of the $\mathbf{G}$ operator, p, is evaluated at all time points in $\mathbb{N}$, and the operand of the $\mathbf{X}$ operator, $\neg p$, as well as its operand, p, are evaluated at time point 1. We can include this information into (1) by writing the set of time points at which an operand is evaluated directly below the corresponding operator. Note that in this scheme there is no place for the set of time points at which (1) itself is evaluated; however, (1) (as any LTL formula) will always be evaluated only at time point 0, so this need not be spelled out explicitly. We then obtain (2). It is easy to see that (1) evaluates to 0 on any word π, i.e., it is unsatisfiable. The reason for this is that at time point 1 p would have to be 1 in the first conjunct $\mathbf{G}p$ and 0 in the second conjunct $\mathbf{X}\neg p$. Notice that for this to happen the operand of $\mathbf{G}p$, p, needs to be evaluated only at time point 1; it is immaterial at any other time point. We can include this information into (2) by replacing $\mathbb{N}$ below $\mathbf{G}$ with $\{1\}$, obtaining (3). (3) can be seen as a UC of (1).

$$(\mathbf{G}p) \wedge (\mathbf{X}\neg p) \qquad (1) \qquad\qquad (\underset{\mathbb{N}}{\mathbf{G}}p) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg} p) \qquad (2) \qquad\qquad (\underset{\{1\}}{\mathbf{G}}\, p) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg} p) \qquad (3)$$

In this paper we make the following contributions. 1. We suggest to enhance UCs for LTL with information on the time points at which the subformulas of a UC are relevant for unsatisfiability, leading to a more fine-grained notion of UCs for LTL than [36]. 2. We propose a method to obtain that information from a temporal resolution proof of unsatisfiability of an LTL formula. 3. We implement our method in TRP++, and we experimentally evaluate it. We are not aware of any other tool that performs extraction of UCs for propositional LTL at that level of granularity. We make the source code of our solver available. Conceptually, under the frequently legitimate assumption that a system description can be translated into an LTL formula, our results extend to vacuity for LTL.

Besides debugging as outlined above UCs are also used for avoiding the exploration of parts of a search space that can be known not to contain a solution for reasons "equivalent" to the reasons for previous failures (e.g., [12]). While our results might also benefit that application, we focus on debugging.

Temporal resolution [19] lends itself as a basis for enhancing UCs for LTL with information on temporal relevance for two reasons. First, the temporal resolution-based solver TRP++ [24, 25, 1] proved to be competitive in a recent evaluation of solvers for LTL satisfiability, in particular on unsatisfiable instances [37]. Second, a temporal resolution proof naturally induces a resolution graph [35], which provides a clean framework for extracting information from the proof.

In this paper we use the following notion of relevance. Assume an LTL formula ϕ and a temporal resolution proof of its unsatisfiability. Remove those parts of the proof that did not contribute to proving unsatisfiability. Consider what is left to be relevant for unsatisfiability; this includes not only which subformulas of ϕ are used but also the time points at which they are used. Clearly, this notion of relevance may not lead to results of minimal or minimum relevance: the fact that some part of ϕ is used at some time point in a specific proof of the unsatisfiability of ϕ does not mean that all such proofs will use that part of ϕ at that time point. For other notions of relevance see, e.g., [30, 6]. The reasons for using our notion of relevance are pragmatic. First, this notion of relevance can be computed with little overhead from a proof of unsatisfiability, which is assumed to be carried out anyway. Second, the result of our notion of relevance can serve as an already reduced input to some of the other notions.

Simmonds et al. [38] use SAT-based bounded model checking for vacuity detection. They indicate which time points are relevant for showing that a variable is non-vacuous. They only consider k-step

$(\pi,i) \models 1, \not\models 0$			$(\pi,i) \models p$	$\Leftrightarrow$	$p \in \pi[i]$
$(\pi,i) \models \neg\psi$	$\Leftrightarrow$	$(\pi,i) \not\models \psi$	$(\pi,i) \models \mathbf{X}\psi$	$\Leftrightarrow$	$(\pi,i+1) \models \psi$
$(\pi,i) \models \psi \vee \psi'$	$\Leftrightarrow$	$(\pi,i) \models \psi$ or $(\pi,i) \models \psi'$	$(\pi,i) \models \psi \wedge \psi'$	$\Leftrightarrow$	$(\pi,i) \models \psi$ and $(\pi,i) \models \psi'$
$(\pi,i) \models \psi\mathbf{U}\psi'$	$\Leftrightarrow$	$\exists i' \geq i\,.\,((\pi,i') \models \psi' \wedge \forall i \leq i'' < i'\,.\,(\pi,i'') \models \psi)$	$(\pi,i) \models \psi\mathbf{R}\psi'$	$\Leftrightarrow$	$\forall i' \geq i\,.\,((\pi,i') \models \psi' \vee \exists i \leq i'' < i'\,.\,(\pi,i'') \models \psi)$
$(\pi,i) \models \mathbf{F}\psi$	$\Leftrightarrow$	$\exists i' \geq i\,.\,(\pi,i') \models \psi$	$(\pi,i) \models \mathbf{G}\psi$	$\Leftrightarrow$	$\forall i' \geq i\,.\,(\pi,i') \models \psi$

Table 1: Semantics of LTL. π is a word in $(2^{AP})^{\omega}$, i is a time point in $\mathbb{N}$.

vacuity, i.e., taking into account bounded model checking runs up to a bound k, and leave the problem of removing the bound k open. [36] mentions indicating time points at which subformulas of a UC in LTL are relevant for unsatisfiability. The idea is not formalized. It first appears in the context of a UC extraction algorithm that is complete only for a strict subset of LTL. Later [36] proposes example (4) and conjectures that sets of time points can be obtained from a tableau method and that these sets are semilinear. Some work [30, 6] determines the time points at which propositions in witnesses of satisfiable LTL formulas are relevant for satisfiability.

This paper builds on a fair amount of previous work: we use temporal resolution as implemented in TRP++ [19, 24, 25, 1] and its extension to extract UCs [35]. To make this paper self-contained we provide a concise description of both. However, to allow sufficient room for the contributions of this paper we have to limit the amount of explanation for previous work. Temporal resolution has been developed since the early 1990s [18], and we refer to [19] for a general overview, to [15, 14, 13] for details on loop search, and to [24, 25, 1] for the implementation in TRP++. In [35] we provide some intuition on temporal resolution with a slant towards BDD-based symbolic model checking. Finally, we refer to Sec. 4 for an example of an execution of the temporal resolution algorithm and the corresponding extraction of UCs.

Section 2 starts with preliminaries. Temporal resolution and its clausal normal form SNF are introduced in Sec. 3. In Sec. 4 we restate the construction of a resolution graph and its use to obtain a UC from [35]. This is extended to compute time points at which subformulas are relevant for unsatisfiability in Sec. 5, 6. A number of examples are spread throughout the paper; in Sec. 7 we provide an example close to a real world situation. We discuss our implementation and experimental evaluation in Sec. 8. Section 9 concludes. Due to space constraints some proofs are sketched or omitted. For the full version of this paper [34] including proofs and for our implementation, examples, and log files see [2].

2 Preliminaries

Let $\mathbb{N}$ be the naturals, and let $I \subseteq \mathbb{N}$ be a set of naturals. I is *linear* iff there exist two naturals p (*period*) and o (*offset*) such that $I = p \cdot \mathbb{N} + o$. I is *semilinear* iff it is the union of finitely many linear sets. Let Σ be a finite alphabet, $\sigma \in \Sigma$ a letter in Σ, $L \subseteq \Sigma^*$ a language over Σ, and $w \in L$ a word in L. Define a function from words and letters to naturals $\Psi : \Sigma^* \times \Sigma \to \mathbb{N}, (w, \sigma) \mapsto m$ where m is the number of occurrences of σ in w. Ψ is called *Parikh mapping* and $\Psi(w,\sigma)$ is called the *Parikh image* of σ in w. The Parikh image of a set of words W is defined in the natural way: $\Psi(W,\sigma) = \{\Psi(w,\sigma) \mid w \in W\}$. Parikh's theorem [28] states that for every context-free language L, for every letter σ, the Parikh image $\Psi(L,\sigma)$ is semilinear.

We use a standard version of LTL, see, e.g., [17]. Let $\mathbb{B}$ be the set of Booleans, and let AP be a finite set of atomic propositions. The set of *LTL formulas* is constructed inductively as follows. The Boolean constants 0 (false), 1 (true) $\in \mathbb{B}$ and any atomic proposition $p \in AP$ are LTL formulas. If ψ, ψ' are LTL formulas, so are $\neg\psi$ (not), $\psi \vee \psi'$ (or), $\psi \wedge \psi'$ (and), $\mathbf{X}\psi$ (next time), $\psi\mathbf{U}\psi'$ (until), $\psi\mathbf{R}\psi'$ (releases), $\mathbf{F}\psi$ (finally), and $\mathbf{G}\psi$ (globally). We use $\psi \to \psi'$ (implies) as an abbreviation for $\neg\psi \vee \psi'$. For the semantics of LTL see Tab. 1. An occurrence of a subformula ψ of an LTL formula ϕ has *positive polarity* ($+$) if it appears under an even number of negations in ϕ and *negative polarity* ($-$) otherwise.

rule	premise 1	part.	premise 2	part.	conclusion	part.	p.1 – c	t.s. 1	p.2 – c	t.s. 2	vt. c
saturation											
init-ii	$(P \vee l)$	M	$(\neg l \vee Q)$	M	$(P \vee Q)$	M	✓	✗	✓	✗	✓
init-in	$(P \vee l)$	M	$(\mathbf{G}(\neg l \vee Q))$	M	$(P \vee Q)$	M	✓	✗	✓	✗	✓
step-nn	$(\mathbf{G}(P \vee l))$	M	$(\mathbf{G}(\neg l \vee Q))$	M	$(\mathbf{G}(P \vee Q))$	M	✓	✗	✓	✗	✓
step-nx	$(\mathbf{G}(P \vee l))$	M	$(\mathbf{G}((Q) \vee (\mathbf{X}(\neg l \vee R))))$	M	$(\mathbf{G}((Q) \vee (\mathbf{X}(P \vee R))))$	M	✓	✓	✓	✗	✓
step-xx	$(\mathbf{G}((P) \vee (\mathbf{X}(Q \vee l))))$	ML	$(\mathbf{G}((R) \vee (\mathbf{X}(\neg l \vee S))))$	ML	$(\mathbf{G}((P \vee R) \vee (\mathbf{X}(Q \vee S))))$	ML	✓	✗	✓	✗	✓
augmentation											
aug1	$(\mathbf{G}((P) \vee (\mathbf{F}(l))))$			M	$(\mathbf{G}(P \vee l \vee wl))$	M	✓	✗	—	—	✓
aug2	$(\mathbf{G}((P) \vee (\mathbf{F}(l))))$			M	$(\mathbf{G}((\neg wl) \vee (\mathbf{X}(l \vee wl))))$	M	✗	✗	—	—	✓
BFS loop search											
BFS-loop-it-init-x	$c \equiv (\mathbf{G}((P) \vee (\mathbf{X}(Q))))$ with $\lvert Q \rvert > 0$			M	c	L	✓	✗	—	—	✓
BFS-loop-it-init-n	$(\mathbf{G}(P))$			M	$(\mathbf{G}((0) \vee (\mathbf{X}(P))))$	L	✓	✓	—	—	✓
BFS-loop-it-init-c	$(\mathbf{G}(P))$	L'	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}((0) \vee (\mathbf{X}(P \vee l))))$	L	✗	✗	✗	✗	✓
BFS-loop-it-sub	$c \equiv (\mathbf{G}(P))$ with $c \rightarrow (\mathbf{G}(Q))$			L	$(\mathbf{G}((0) \vee (\mathbf{X}(Q \vee l))))$ generated by BFS-loop-it-init-c	L	✓	✓	—	—	✗
BFS-loop-conclusion1	$(\mathbf{G}(P))$	L	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}(P \vee Q \vee l))$	M	✓	✗	✓	✗	✓
BFS-loop-conclusion2	$(\mathbf{G}(P))$	L	$(\mathbf{G}((Q) \vee (\mathbf{F}(l))))$	M	$(\mathbf{G}((\neg wl) \vee (\mathbf{X}(P \vee l))))$	M	✓	✓	✗	✗	✓

Table 2: Production rules in TRP++. Let $P \equiv \bigvee_{i=1\ldots n} p_i$, $Q \equiv \bigvee_{i=1\ldots n'} q_i$, $R \equiv \bigvee_{i=1\ldots n''} r_i$, $S \equiv \bigvee_{i=1\ldots n'''} s_i$.

3 Temporal Resolution (TR) in TRP++

TR works on formulas in a clausal normal form called separated normal form (SNF) [19]. For any atomic proposition $p \in AP$ p and $\neg p$ are *literals*. Let $p_1, \ldots, p_n$, $q_1, \ldots, q_{n'}$, l with $0 \leq n, n'$ be literals such that $p_1, \ldots, p_n$ and $q_1, \ldots, q_{n'}$ are pairwise different. Then 1. $(p_1 \vee \ldots \vee p_n)$ is an *initial clause*; 2. $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{X}(q_1 \vee \ldots \vee q_{n'}))))$ is a *global clause*; and 3. $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l))))$ is an *eventuality clause*. l is called an *eventuality literal*. As usual an empty disjunction (resp. conjunction) stands for 0 (resp. 1). $()$ or $(\mathbf{G}())$, denoted $\Box$, stand for 0 or $\mathbf{G}(0)$ and are called *empty clause*. The set of all SNF clauses is denoted $\mathbb{C}$. Let $c_1, \ldots, c_n$ with $0 \leq n$ be SNF clauses. Then $\bigwedge_{1 \leq i \leq n} c_i$ is an LTL formula in *SNF*. Every LTL formula ϕ can be transformed into an equisatisfiable formula ϕ' in SNF [19].

We now describe TR [19] as implemented in TRP++ [24, 25, 1]. The production rules of TRP++ are shown in Tab. 2. The first column assigns a name to a production rule. The second and fourth columns list the premises. The sixth column gives the conclusion. Columns 3, 5, and 7 are described below. Columns 8–12 become relevant only in later sections.

Algorithm 1 provides a high level view of TR in TRP++ [25]. The algorithm takes a set of starting clauses C in SNF as input. It returns *unsat* if C is found to be unsatisfiable (by deriving $\Box$) and *sat* otherwise. Resolution between two initial or two global clauses or between an initial and a global clause is performed by a simple extension of propositional resolution. The corresponding production rules are listed under *saturation* in Tab. 2. Given a set of SNF clauses C we say that one *saturates* C if one applies these production rules to clauses in C until no new clauses are generated. Resolution between a set of initial and global clauses and an eventuality clause with eventuality literal l requires finding a set of global clauses that allows to infer conditions under which $\mathbf{XG}\neg l$ holds. Such a set of clauses is called a *loop* in $\neg l$. Loop search involves all production rules in Tab. 2 except $\boxed{\text{init-ii}}$, $\boxed{\text{init-in}}$, $\boxed{\text{step-nn}}$, and $\boxed{\text{step-nx}}$.

In line 1 Alg. 1 initializes M with the set of starting clauses and terminates iff one of these is the empty clause. Then, in line 2, it saturates M (terminating iff the empty clause is generated). In line 3 it *augments* M by applying production rule $\boxed{\text{aug1}}$ to each eventuality clause in M and $\boxed{\text{aug2}}$ once per eventuality literal in M, where wl is a fresh proposition. This is followed by another round of saturation in line 4. From now on Alg. 1 alternates between searching for a loop for some eventuality clause c (lines 9–18) and saturating M if loop search has generated new clauses (line 19). It terminates if either the empty clause was derived (line 19) or if no new clauses were generated (line 20).

Loop search for some eventuality clause c may take several *iterations* (lines 11–15). Each loop search iteration uses saturation restricted to $\boxed{\text{step-xx}}$ as a subroutine (line 12). Therefore, each loop search

Algorithm 1: LTL satisfiability checking via TR in TRP++.

```
    Input: A set of SNF clauses $C$.    Output: $Unsat$ if $C$ is unsatisfiable; $sat$ otherwise.
 1  $M \leftarrow C$; if $\Box \in M$ then return unsat;
 2  saturate($M$); if $\Box \in M$ then return unsat;
 3  augment($M$);
 4  saturate($M$); if $\Box \in M$ then return unsat;
 5  $M' \leftarrow \emptyset$;
 6  while $M' \neq M$ do
 7      $M' \leftarrow M$;
 8      for $c \in C$ . $c$ is an eventuality clause do
 9          $C' \leftarrow \{\Box\}$;
10          repeat
11              initialize-BFS-loop-search-iteration($M$, $c$, $C'$, $L$);
12              saturate-step-xx($L$);
13              $C' \leftarrow \{c' \in L \mid c'$ has empty $\mathbf{X}$ part$\}$;
14              $C'' \leftarrow \{(\mathbf{G}(Q)) \mid (\mathbf{G}((0) \vee (\mathbf{X}(Q \vee l)))) \in L$ generated by $\boxed{\text{BFS-loop-it-init-c}}\}$;
15              $found \leftarrow subsumes(C', C'')$;
16          until $found$ or $C' = \emptyset$ ;
17          if $found$ then
18              derive-BFS-loop-search-conclusions($c$, $C'$, $M$);
19              saturate($M$); if $\Box \in M$ then return unsat;
20  return sat;
```

iteration has its own set of clauses L in which it works. We call M and L *partitions*. Columns 3, 5, and 7 in Tab. 2 indicate whether a premise (resp. conclusion) of a production rule is taken from (resp. put into) the main partition (M), the loop partition of the current loop search iteration (L), the loop partition of the previous loop search iteration (L'), or either of M or L as long as premises and conclusion are in the same partition (ML). In line 11 partition L of a loop search iteration is initialized by applying production rule $\boxed{\text{BFS-loop-it-init-x}}$ once for each global clause with non-empty $\mathbf{X}$ part in M, rule $\boxed{\text{BFS-loop-it-init-n}}$ once for each global clause with empty $\mathbf{X}$ part in M, and rule $\boxed{\text{BFS-loop-it-init-c}}$ once for each global clause with empty $\mathbf{X}$ part in the partition of the previous loop search iteration L'. Notice that by construction at this point L contains only global clauses with non-empty $\mathbf{X}$ part. Then L is saturated using only rule $\boxed{\text{step-xx}}$ (line 12). A loop has been found iff each global clause with empty $\mathbf{X}$ part that was derived in the previous loop search iteration is subsumed by at least one global clause with empty $\mathbf{X}$ part that was derived in the current loop search iteration (lines 13–15). Subsumption between a pair of clauses corresponds to an instance of production rule $\boxed{\text{BFS-loop-it-sub}}$; note, though, that this rule does not produce a new clause but records a relation between two clauses to be used later for extraction of a UC. Loop search for c terminates, if either a loop has been found or no clauses with empty $\mathbf{X}$ part were derived (line 16). If a loop has been found, rules $\boxed{\text{BFS-loop-conclusion1}}$ and $\boxed{\text{BFS-loop-conclusion2}}$ are applied once to each global clause with empty $\mathbf{X}$ part that was derived in the current loop search iteration (line 18) to obtain the loop search conclusions for the main partition.

4 UC Extraction via TR

In this section we restate the main definitions from [35] that show how to construct UCs via TR. Then we present an example for extraction of a UC in SNF, which is extended with sets of time points in Sec. 6.

Given an unsatisfiable set of SNF clauses C we would first like to obtain a subset of C, C^{uc}, that is by itself unsatisfiable from an execution of Alg. 1. The general idea of the construction is unsurprising in that during the execution of Alg. 1 a resolution graph is built that records which clauses were used to generate other clauses (Def. 1). Then the resolution graph is traversed backwards from the empty clause to find the subset of C that was actually used to prove unsatisfiability (Def. 2). The main concern of Def. 1, 2, and their proof of correctness in Thm. 1 (see [35]) is therefore that certain parts of the TR proof do not need to be taken into account when determining C^{uc}.

Definition 1. *[35] A* resolution graph *G is a directed graph consisting of 1. a set of vertices V, 2. a set of directed edges $E \subseteq V \times V$, 3. a labeling of vertices with SNF clauses $L_V : V \to \mathbb{C}$, and 4. a partitioning $\mathcal{Q}^V$ of the set of vertices V into one main partition M^V and one partition L_i^V for each BFS loop search iteration: $\mathcal{Q}^V : V = M^V \uplus L_0^V \uplus \ldots \uplus L_n^V$.[1] Let C be a set of SNF clauses. During an execution of Alg. 1 with input C a resolution graph G is constructed as follows.*

In line 1 G is initialized: 1. V contains one vertex v per clause c in C: $V = \{v_c \mid c \in C\}$, 2. E is empty: $E = \emptyset$, 3. each vertex is labeled with the corresponding clause: $L_V : V \to C, L_V(v_c) = c$, and 4. the partitioning $\mathcal{Q}^V$ contains only the main partition M^V, which contains all vertices: $\mathcal{Q}^V : M^V = V$.

Whenever a new BFS loop search iteration is entered (line 11), a new partition L_i^V is created and added to $\mathcal{Q}^V$. For each application of a production rule from Tab. 2 that either generates a new clause in partition M or L or is the first application of rule $\boxed{\textit{BFS-loop-it-sub}}$ to clause c'' in C'' in line 15: 1. if column 12 (vt. c) *of Tab. 2 contains ✔, then a new vertex v is created for the conclusion c (which is a new clause), labeled with c, and put into partition M^V or L_i^V; 2. if column 8* (p.1 – c) *(resp. column 10* (p.2 – c)*) contains ✔, then an edge is created from the vertex labeled with premise 1 (resp. premise 2) in partition M^V or L_i^V to the vertex labeled with the conclusion in partition M^V or L_i^V.*

Definition 2. *[35] Let C be a set of SNF clauses to which Alg. 1 has been applied and shown unsatisfiability, let G be the resolution graph, and let $v_\Box$ be the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\Box$. Let G' be the smallest subgraph of G that contains $v_\Box$ and all vertices in G (and the corresponding edges) that are backward reachable from $v_\Box$. The* UC of C in SNF*, C^{uc}, is the subset of C such that there exists a vertex v in the subgraph G', labeled with $c \in C$, and contained in the main partition M^V of G: $C^{uc} = \{c \in C \mid \exists v \in V_{G'} \,.\, L_V(v) = c \wedge v \in M^V\}$.*

Theorem 1. *[35] Let C be a set of SNF clauses to which Alg. 1 has been applied and shown unsatisfiability, and let C^{uc} be the UC of C in SNF. Then C^{uc} is unsatisfiable.*

We now lift the extraction of UCs from SNF to LTL by restating the translation from LTL to SNF and the mapping from a UC in SNF back to LTL from [35].

Definition 3. *[35] Let ϕ be an LTL formula over atomic propositions AP, and let $X = \{x, x', \ldots\}$ be a set of fresh atomic propositions not in AP. Assign each occurrence of a subformula ψ in ϕ a Boolean value or a proposition according to col. 2 of Tab. 3, which is used to reference ψ in the SNF clauses for its superformula. Moreover, assign each occurrence of ψ a set of SNF clauses according to col. 3 or 4 of Tab. 3. Let $SNF_{aux}(\phi)$ be the set of all SNF clauses obtained from ϕ that way. Then the* SNF of ϕ *is defined as $SNF(\phi) \equiv x_\phi \wedge \bigwedge_{c \in SNF_{aux}(\phi)} c$.*

Definition 4. *[35] Let ϕ be an unsatisfiable LTL formula, let $SNF(\phi)$ be its SNF, and let C^{uc} be the UC of $SNF(\phi)$ in SNF. Then the* UC of ϕ in LTL*, ϕ^{uc}, is obtained as follows. For each positive (resp. negative) polarity occurrence of a proper subformula ψ of ϕ with proposition x_ψ according to Tab. 3, replace ψ in ϕ with* 1 *(resp.* 0*) iff C^{uc} contains no clause with an occurrence of proposition x_ψ that is marked $\boxed{\textit{blue boxed}}$ in Tab. 3. (We are sloppy in that we "replace" subformulas of replaced subformulas, while in effect they simply vanish.)*

Theorem 2. *[35] Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be the UC of ϕ in LTL. Then ϕ^{uc} is unsatisfiable.*

[1] $\uplus$ denotes disjoint union of sets.

Subf.	Prop.	SNF Clauses (positive polarity occurrences)	SNF Clauses (negative polarity occurrences)
$1/0/p$	$1/0/p$	—	—
$\neg\psi$	$x_{\neg\psi}$	$(\mathbf{G}(x_{\neg\psi} \rightarrow (\neg\boxed{x_\psi})))$	$(\mathbf{G}((\neg x_{\neg\psi}) \rightarrow \boxed{x_\psi}))$
$\psi \wedge \psi'$	$x_{\psi\wedge\psi'}$	$(\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow \boxed{x_\psi})), (\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow \boxed{x_{\psi'}}))$	$(\mathbf{G}((\neg x_{\psi\wedge\psi'}) \rightarrow ((\neg\boxed{x_\psi}) \vee (\neg\boxed{x_{\psi'}}))))$
$\psi \vee \psi'$	$x_{\psi\vee\psi'}$	$(\mathbf{G}(x_{\psi\vee\psi'} \rightarrow (\boxed{x_\psi} \vee \boxed{x_{\psi'}})))$	$(\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow (\neg\boxed{x_\psi}))), (\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow (\neg\boxed{x_{\psi'}})))$
$\mathbf{X}\psi$	$x_{\mathbf{X}\psi}$	$(\mathbf{G}(x_{\mathbf{X}\psi} \rightarrow (\mathbf{X}\boxed{x_\psi})))$	$(\mathbf{G}((\neg x_{\mathbf{X}\psi}) \rightarrow (\mathbf{X}\neg\boxed{x_\psi})))$
$\mathbf{G}\psi$	$x_{\mathbf{G}\psi}$	$(\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow (\mathbf{X}x_{\mathbf{G}\psi}))), (\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow \boxed{x_\psi}))$	$(\mathbf{G}((\neg x_{\mathbf{G}\psi}) \rightarrow (\mathbf{F}\neg\boxed{x_\psi})))$
$\mathbf{F}\psi$	$x_{\mathbf{F}\psi}$	$(\mathbf{G}(x_{\mathbf{F}\psi} \rightarrow (\mathbf{F}\boxed{x_\psi})))$	$(\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\mathbf{X}\neg x_{\mathbf{F}\psi}))), (\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\neg\boxed{x_\psi})))$
$\psi\mathbf{U}\psi'$	$x_{\psi\mathbf{U}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\boxed{x_{\psi'}} \vee \boxed{x_\psi}))), (\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\boxed{x_{\psi'}} \vee (\mathbf{X}x_{\psi\mathbf{U}\psi'})))),$ $(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (\mathbf{F}\boxed{x_{\psi'}})))$	$(\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow (\neg\boxed{x_{\psi'}}))), (\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow ((\neg\boxed{x_\psi}) \vee (\mathbf{X}\neg x_{\psi\mathbf{U}\psi'}))))$
$\psi\mathbf{R}\psi'$	$x_{\psi\mathbf{R}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow \boxed{x_{\psi'}})), (\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow (\boxed{x_\psi} \vee (\mathbf{X}x_{\psi\mathbf{R}\psi'}))))$	$(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg\boxed{x_{\psi'}}) \vee (\neg\boxed{x_\psi})))), (\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg\boxed{x_{\psi'}}) \vee (\mathbf{X}\neg x_{\psi\mathbf{R}\psi'})))),$ $(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow (\mathbf{F}\neg\boxed{x_{\psi'}})))$

Table 3: Translation from LTL to SNF.

In Fig. 1 we show an example of an execution of the TR algorithm with the corresponding resolution graph and UC extraction in SNF. The set of SNF clauses C to be solved contains a, $\mathbf{G}((\neg a) \vee \mathbf{X}b)$, $\mathbf{G}((\neg b) \vee \mathbf{X}a)$, $\mathbf{G}((\neg a) \vee \neg c)$, $\mathbf{G}((\neg c) \vee \mathbf{X}\neg a)$, and $\mathbf{G}(\mathbf{F}c)$. The first three clauses a, $\mathbf{G}((\neg a) \vee \mathbf{X}b)$, and $\mathbf{G}((\neg b) \vee \mathbf{X}a)$ force a to be 1 at even time points. This is contradicted by the last three clauses $\mathbf{G}((\neg a) \vee \neg c)$, $\mathbf{G}((\neg c) \vee \mathbf{X}\neg a)$, and $\mathbf{G}(\mathbf{F}c)$: they require that a eventually becomes 0 for two consecutive time points. Clearly, C is unsatisfiable. This example is based on the same idea as (4) in Sec. 5. However, the SNF obtained by our translation from LTL to SNF for (4) is larger than C, with the corresponding figure harder to fit on one page.

In Fig. 1 the TR algorithm proceeds from bottom to top. Clauses are connected with edges according to cols. 8 and 10 of Tab. 2 and labeled with the corresponding production rules, where “BFS-loop” is abbreviated to “loop”, “init” to “i”, and “conclusion” to “conc”. In the first row from the bottom (in the light red shaded rectangle) are the starting clauses from C. In the top right corner is the empty clause $\Box$ signalling unsatisfiability of C. Row 2 contains the clauses resulting from the first round of saturation (line 2 in Alg. 1) and from augmentation (line 3).[2] The second round of saturation (line 4) produces no new clauses. The dark green shaded rectangle is the partition for the first iteration of a loop search for a loop in $\neg c$. Row 3 contains the result of loop search initialization (line 11) and row 4 the clauses obtained by restricted saturation (line 12). As none of the clauses in row 4 subsumes $\Box$, this iteration terminates without having found a loop. The second loop search iteration is in the light green shaded rectangle. Again row 5 contains the result of loop search initialization and row 6 the clauses obtained by restricted saturation. This time the subsumption test is successful (lines 13–15), and row 7 shows the loop search conclusions (line 18). The last row finally contains the derivation of $\Box$ by saturation (line 19).

The clauses that are backward reachable from $\Box$ are shown in blue with blue, thick, dashed boxes. The corresponding edges are thick, blue or red, and dashed or dotted. The resulting UC comprises all clauses in C (note that this example shows the mechanism rather than the benefits of extracting UCs).

The distinction between blue, dashed and red, dotted edges as well as the sets of time points shown in black boxes are needed when sets of time points are added in Sec. 6. Please ignore those for now.

5 LTL with Sets of Time Points (LTLp)

In this section we propose a notation that allows to integrate more detailed information from a resolution proof of the unsatisfiability of some LTL formula ϕ into the UC ϕ^{uc}. The information we are interested

[2] While it may seem that some clauses are not considered for loop initialization or saturation, this is due to either subsumption of one clause by another (e.g., $\mathbf{G}((\neg wc) \vee \mathbf{X}(c \vee wc))$ by $\mathbf{G}(c \vee wc)$) or the fact that TRP++ uses *ordered* resolution (e.g., a with $\mathbf{G}((\neg a) \vee \neg c)$; [24]). Both are issues of completeness of TR and, therefore, not discussed in this paper.

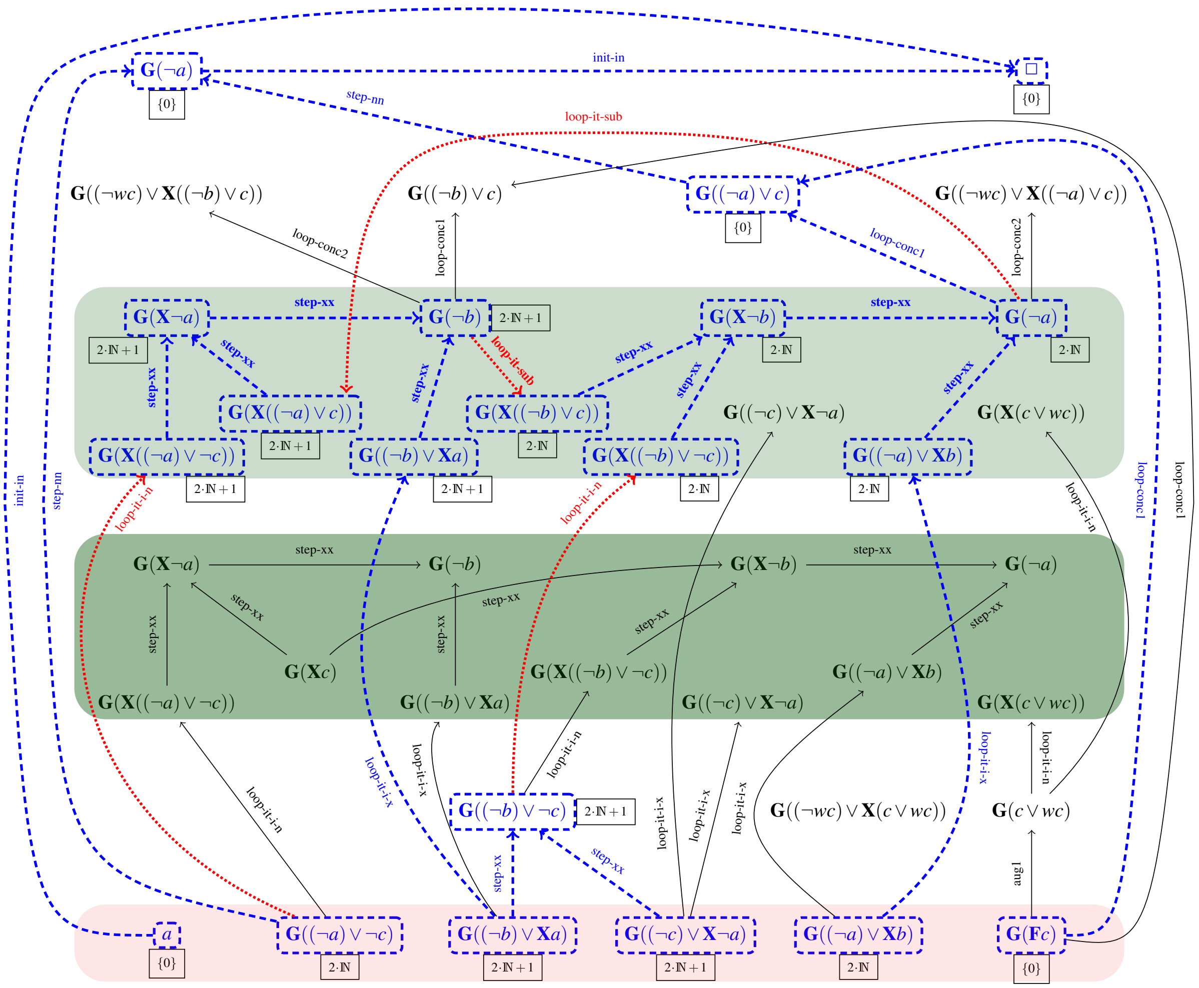

$\mathbf{G}(\neg a)$
$\{0\}$
init-in
$\square$
$\{0\}$
step-nn
loop-it-sub
$\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg b) \vee c))$
$\mathbf{G}((\neg b) \vee c)$
$\mathbf{G}((\neg a) \vee c)$
$\{0\}$
$\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg a) \vee c))$
loop-conc2
loop-conc1
loop-conc1
loop-conc2
$\mathbf{G}(\mathbf{X}\neg a)$
step-xx
$\mathbf{G}(\neg b)$
$2\cdot\mathbb{N}+1$
$\mathbf{G}(\mathbf{X}\neg b)$
step-xx
$\mathbf{G}(\neg a)$
$2\cdot\mathbb{N}+1$
$2\cdot\mathbb{N}$
$2\cdot\mathbb{N}$
step-xx
step-xx
loop-it-sub
step-xx
step-xx
step-xx
step-xx
$\mathbf{G}(\mathbf{X}((\neg a) \vee c))$
$\mathbf{G}(\mathbf{X}((\neg b) \vee c))$
$\mathbf{G}((\neg c) \vee \mathbf{X}\neg a)$
$\mathbf{G}(\mathbf{X}(c \vee wc))$
$2\cdot\mathbb{N}+1$
$2\cdot\mathbb{N}$
$\mathbf{G}(\mathbf{X}((\neg a) \vee \neg c))$
$\mathbf{G}((\neg b) \vee \mathbf{X}a)$
$\mathbf{G}(\mathbf{X}((\neg b) \vee \neg c))$
$\mathbf{G}((\neg a) \vee \mathbf{X}b)$
$2\cdot\mathbb{N}+1$
$2\cdot\mathbb{N}+1$
$2\cdot\mathbb{N}$
$2\cdot\mathbb{N}$
init-in
step-nn
loop-it-i-n
loop-it-i-n
loop-conc1
loop-conc1
loop-it-i-n
$\mathbf{G}(\mathbf{X}\neg a)$
step-xx
$\mathbf{G}(\neg b)$
$\mathbf{G}(\mathbf{X}\neg b)$
step-xx
$\mathbf{G}(\neg a)$
step-xx
step-xx
step-xx
step-xx
step-xx
step-xx
$\mathbf{G}(\mathbf{X}c)$
$\mathbf{G}(\mathbf{X}((\neg b) \vee \neg c))$
$\mathbf{G}((\neg a) \vee \mathbf{X}b)$
$\mathbf{G}(\mathbf{X}((\neg a) \vee \neg c))$
$\mathbf{G}((\neg b) \vee \mathbf{X}a)$
$\mathbf{G}((\neg c) \vee \mathbf{X}\neg a)$
$\mathbf{G}(\mathbf{X}(c \vee wc))$
loop-it-i-n
loop-it-i-x
loop-it-i-x
loop-it-i-n
$\mathbf{G}((\neg b) \vee \neg c)$
$2\cdot\mathbb{N}+1$
loop-it-i-x
loop-it-i-x
loop-it-i-x
$\mathbf{G}((\neg wc) \vee \mathbf{X}(c \vee wc))$
loop-it-i-x
loop-it-i-n
$\mathbf{G}(c \vee wc)$
step-xx
step-xx
aug1
a
$\{0\}$
$\mathbf{G}((\neg a) \vee \neg c)$
$2\cdot\mathbb{N}$
$\mathbf{G}((\neg b) \vee \mathbf{X}a)$
$2\cdot\mathbb{N}+1$
$\mathbf{G}((\neg c) \vee \mathbf{X}\neg a)$
$2\cdot\mathbb{N}+1$
$\mathbf{G}((\neg a) \vee \mathbf{X}b)$
$2\cdot\mathbb{N}$
$\mathbf{G}(\mathbf{F}c)$
$\{0\}$

Figure 1: Example of an execution of the TR algorithm with corresponding resolution graph and UC extraction in SNF with sets of time points.

in are the time points at which a part of an LTL formula is needed to prove unsatisfiability. Hence, we assign to each subformula a set of time points that indicates at which time points that subformula will be evaluated; at other time points the subformula is considered to be 1 or 0 depending on polarity. We wish to emphasize that it is not our goal to introduce a "new logic", but merely to suggest a notation with well defined semantics that allows to smoothly integrate such information.

Definition 5. *The set of* LTLp formulas *is constructed inductively as follows. The Boolean constants* 0 *(false),* 1 *(true)* $\in \mathbb{B}$ *and any atomic proposition* $p \in AP$ *are LTL*p *formulas. If* $I, I' \subseteq \mathbb{N}$ *are sets of time points and if* τ, τ' *are LTL*p *formulas, so are* $\underset{I}{\neg}\tau$ *(not),* $\tau \underset{I,I'}{\vee} \tau'$ *(or),* $\tau \underset{I,I'}{\wedge} \tau'$ *(and),* $\underset{I}{\mathbf{X}}\tau$ *(next time),* $\tau \underset{I,I'}{\mathbf{U}} \tau'$ *(until),* $\tau \underset{I,I'}{\mathbf{R}} \tau'$ *(releases),* $\underset{I}{\mathbf{F}}\tau$ *(finally), and* $\underset{I}{\mathbf{G}}\tau$ *(globally).* $\tau \underset{I,I'}{\rightarrow} \tau'$ *(implies) abbreviates* $\underset{I}{\neg}\tau \underset{I,I'}{\vee} \tau'$.

We now recursively define the semantics of an LTLp formula at time points $i \in \mathbb{N}$ of a word $\pi \in (2^{AP})^{\omega}$. Note that the semantics depends on the polarity of the occurrence of a subformula. The intuition for the semantics is that if a time point i is not contained in a set I, then the corresponding operand at that time point cannot be used to establish unsatisfiability.

formula		positive polarity	negative polarity
$(\pi,i) \models 1$ (resp. 0)	$\Leftrightarrow$	1 (resp. 0)	1 (resp. 0)
$(\pi,i) \models p$	$\Leftrightarrow$	$p \in \pi[i]$	$p \in \pi[i]$
$(\pi,i) \models \underset{I}{\neg}\tau$	$\Leftrightarrow$	$(i \notin I) \vee ((\pi,i) \not\models \tau)$	$(i \in I) \wedge ((\pi,i) \not\models \tau)$
$(\pi,i) \models \tau \underset{I,I'}{\vee} \tau'$	$\Leftrightarrow$	$((i \notin I) \vee ((\pi,i) \models \tau)) \vee ((i \notin I') \vee ((\pi,i) \models \tau'))$	$((i \in I) \wedge ((\pi,i) \models \tau)) \vee ((i \in I') \wedge ((\pi,i) \models \tau'))$
$(\pi,i) \models \tau \underset{I,I'}{\wedge} \tau'$	$\Leftrightarrow$	$((i \notin I) \vee ((\pi,i) \models \tau)) \wedge ((i \notin I') \vee ((\pi,i) \models \tau'))$	$((i \in I) \wedge ((\pi,i) \models \tau)) \wedge ((i \in I') \wedge ((\pi,i) \models \tau'))$
$(\pi,i) \models \underset{I}{\mathbf{X}}\tau$	$\Leftrightarrow$	$(i+1 \notin I) \vee ((\pi,i+1) \models \tau)$	$(i+1 \in I) \wedge ((\pi,i+1) \models \tau)$
$(\pi,i) \models \tau \underset{I,I'}{\mathbf{U}} \tau'$	$\Leftrightarrow$	$\exists i' \geq i \,.\, (((i' \notin I') \vee ((\pi,i') \models \tau')) \wedge (\forall i \leq i'' < i' \,.\, ((i'' \notin I) \vee ((\pi,i'') \models \tau))))$	$\exists i' \geq i \,.\, (((i' \in I') \wedge ((\pi,i') \models \tau')) \wedge (\forall i \leq i'' < i' \,.\, ((i'' \in I) \wedge ((\pi,i'') \models \tau))))$
$(\pi,i) \models \tau \underset{I,I'}{\mathbf{R}} \tau'$	$\Leftrightarrow$	$\forall i' \geq i \,.\, (((i' \notin I') \vee ((\pi,i') \models \tau')) \vee (\exists i \leq i'' < i' \,.\, ((i'' \notin I) \vee ((\pi,i'') \models \tau))))$	$\forall i' \geq i \,.\, (((i' \in I') \wedge ((\pi,i') \models \tau')) \vee (\exists i \leq i'' < i' \,.\, ((i'' \in I) \wedge ((\pi,i'') \models \tau))))$
$(\pi,i) \models \underset{I}{\mathbf{F}}\tau$	$\Leftrightarrow$	$\exists i' \geq i \,.\, ((i' \notin I) \vee ((\pi,i') \models \tau))$	$\exists i' \geq i \,.\, ((i' \in I) \wedge ((\pi,i') \models \tau))$
$(\pi,i) \models \underset{I}{\mathbf{G}}\tau$	$\Leftrightarrow$	$\forall i' \geq i \,.\, ((i' \notin I) \vee ((\pi,i') \models \tau))$	$\forall i' \geq i \,.\, ((i' \in I) \wedge ((\pi,i') \models \tau))$

Table 4: Semantics of LTLp. π is a word in $(2^{AP})^{\omega}$, i is a time point in $\mathbb{N}$.

Definition 6. *The semantics of LTL*p *is given in Tab. 4.* π satisfies *a formula* ϕ *iff the formula holds at the beginning of* π*:* $\pi \models \phi \Leftrightarrow (\pi,0) \models \phi$.

Our definition leaves the top level formula without a set of time points. This is justified, as the only useful value there is $\{0\}$; it is required for satisfaction of an LTLp formula in Def. 6. In Remark 1 we state some properties of LTLp.

Remark 1. *1. An LTL*p *formula* θ *s.t. all sets of time points are* $\mathbb{N}$ *is equivalent to the LTL formula that one obtains from* θ *by removing all sets of time points. 2. An LTL*p *formula* θ *with a positive (resp. negative) polarity subformula* τ*, where* τ *is neither a Boolean constant nor an atomic proposition, s.t. all sets of time points of the top level operator of* τ *are* $\emptyset$ *is equivalent to* θ *with* τ *replaced with* 1 *(resp.* 0*). 3. If* θ *and* χ *are two LTL*p *formulas s.t.* θ *and* χ *differ only in their sets of time points, and all sets of time points in* χ *are (possibly non-strict) supersets of those in* θ*, then* $\chi \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \theta$*. 4. LTL*p *with sets of time points restricted to semilinear sets is no more expressive than QLTL (for QLTL see, e.g., [17]).*

We now illustrate LTLp with an example (4), (5) that is somewhat more involved than (1)–(3) in Sec. 1. The example is still artificial to allow focusing on sets of time points. The first conjunct, p, and the second conjunct, $\mathbf{G}(p \rightarrow \mathbf{XX}p)$, force p to be 1 at even time points. The third conjunct, $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$, requires that eventually p is 0 at two consecutive time points. Clearly, the first two conjuncts contradict the third, i.e., (4) is unsatisfiable. We would now like to obtain small sets of time points that are still sufficient for (4) to be unsatisfiable. The three conjuncts p, $\mathbf{G}(p \rightarrow \mathbf{XX}p)$, and $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$ are evaluated only at time point 0. The operand of the second conjunct, $p \rightarrow \mathbf{XX}p$, needs to be evaluated only at even time points and, therefore, also both operands of the $\rightarrow$ operator. Consequently, it is sufficient to evaluate $\mathbf{X}p$ at odd time points and its operand, p, at even time points > 0. The last conjunct is more complicated. The operand of the $\mathbf{F}$ operator has to be evaluated at every time point; otherwise, $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$ would evaluate to 1. Now note that at each time point one of the two conjuncts of $(\neg p) \wedge \mathbf{X}\neg p$ must contradict a p induced by $p \wedge (\mathbf{G}(p \rightarrow \mathbf{XX}p))$. At time point 0 this can only be the first conjunct, $\neg p$. Hence, if the first conjunct, $\neg p$, is evaluated at even time points and the second conjunct, $\mathbf{X}\neg p$, is evaluated at odd time points, then unsatisfiability is preserved. The resulting LTLp formula is shown in (5). We call (5) a UC of (4) in LTL with sets of time points.

$$p \wedge (\mathbf{G}(p \rightarrow \mathbf{XX}p)) \wedge (\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)) \tag{4}$$

$$p \underset{\{0\},\{0\}}{\wedge} ((\underset{2\cdot\mathbb{N}}{\mathbf{G}} (p \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\rightarrow} \underset{2\cdot\mathbb{N}+1}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} p)) \underset{\{0\},\{0\}}{\wedge} (\underset{\mathbb{N}}{\mathbf{F}} ((\underset{2\cdot\mathbb{N}}{\neg} p) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}+1}{\wedge} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\neg} p))) \tag{5}$$

6 UC Extraction with Sets of Time Points

In this section we show how to enhance a UC in SNF and in LTL with the sets of time points at which its clauses or subformulas are used in its TR proof of unsatisfiability.

Let C be a set of SNF clauses to which Alg. 1 has been applied and shown unsatisfiability, let G be the resolution graph, let G' be the subgraph according to Def. 2 with corresponding UC in SNF C^{uc}, and let $v_\Box$ denote the vertex in the main partition that is L_V-labeled with $\Box$. We start in Def. 7 with labeling edges of G' with 1 if the source vertex is time-shifted one step into the future with respect to the target vertex (e.g., when a global clause with empty $\mathbf{X}$ part is used in $\boxed{\text{step-nx}}$) and all other edges with 0. Then, in Def. 8, we obtain a set of time points for each vertex in G' by assigning time point 0 to $v_\Box$ (i.e., the contradiction is assumed to happen at time point 0). Any other vertex v is assigned the set of the sums of the time steps that occur on any path from v to $v_\Box$ in G'.

Definition 7. *$L_{E'}$ is a labeling of the set of edges in G', E', with time steps in $\{0,1\}$ that maps an edge e to 1 if the corresponding column 9 (*t.s. 1*) or 11 (*t.s. 2*) in Tab. 2 contains a ✓ and to 0 otherwise.*

Definition 8. *Let the edges of G' be $L_{E'}$-labeled. $L'_{V'}$ is another labeling of the set of vertices in G', V', with sets of time points in $2^{\mathbb{N}}$ as follows. $v_\Box$ is $L'_{V'}$-labeled with $\{0\}$. Any other vertex v is $L'_{V'}$-labeled with a set of time points I that contains a time point i iff there exists a path π in G' from v to $v_\Box$ such that the sum of the $L_{E'}$-labels of π is i.*

We now continue the example in Fig. 1. Edges in the subgraph backward reachable from $v_\Box$ that involve a time step of 1 between source and target vertex according to cols. 9 and 11 of Tab. 2 are marked red, dotted. Backward reachable edges that involve no such time step are marked blue, dashed. In the backward reachable subgraph there are four edges that involve a time step of 1 between source and target vertex. Two of those originate from instances of $\boxed{\text{BFS-loop-it-init-n}}$: from $\mathbf{G}((\neg a) \vee \neg c)$ in row 1 to $\mathbf{G}(\mathbf{X}((\neg a) \vee \neg c))$ in row 5 and from $\mathbf{G}((\neg b) \vee \neg c)$ in row 2 to $\mathbf{G}(\mathbf{X}((\neg b) \vee \neg c))$ in row 5. Two others come from instances of $\boxed{\text{BFS-loop-it-sub}}$: from $\mathbf{G}(\neg b)$ to $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$ and from $\mathbf{G}(\neg a)$ to $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$, both from row 6 to row 5. Furthermore, there are two edges from instances of $\boxed{\text{BFS-loop-conclusion2}}$ that would be labeled with a time step of 1, if they were backward reachable from $v_\Box$: from $\mathbf{G}(\neg b)$ (row 6) to $\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg b) \vee c))$ (row 7) and from $\mathbf{G}(\neg a)$ (row 6) to $\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg a) \vee c))$ (row 7). Figure 1 contains no edges induced by an instance of $\boxed{\text{step-nx}}$. Notice how in each case the literals that are taken from the source vertex and put into the target vertex are in the $\mathbf{X}$ part of the target vertex while they are not in the $\mathbf{X}$ part of the source vertex; this is not the case for pairs of source and target vertex connected by an edge that is (or would be) labeled with time step 0.

Each clause c in the backward reachable subgraph is labeled with a set of time points (shown in a black box) obtained by counting the number of red, dotted edges that are traversed on any — possibly looping — path from v_c to $v_\Box$ according to Def. 8. For example, a in row 1 can only reach $\Box$ directly via a blue, dashed edge, leading to set of time points $\{0\}$ (which is the only one making sense for an initial clause; see Lemma 1). Similarly, $\mathbf{G}(\neg a)$ (row 8), $\mathbf{G}((\neg a) \vee c)$ (row 7), and $\mathbf{G}(\mathbf{F}c)$ (row 1) can only reach $\Box$ via sequences of blue, dashed edges, so they are also labeled with $\{0\}$. Only one of the clauses comprising the second loop search iteration (rows 5 and 6 in the light green shaded rectangle) can reach $\Box$ without passing through any other clause in rows 5 or 6, namely $\mathbf{G}(\neg a)$ (row 6) via a sequence of blue, dashed edges. I.e., its set of time points must contain $\{0\}$. However, $\mathbf{G}(\neg a)$ is also part of loop $\mathbf{G}(\neg a)$–$\mathbf{G}(\mathbf{X}((\neg a) \vee c))$–$\mathbf{G}(\mathbf{X}\neg a)$–$\mathbf{G}(\neg b)$–$\mathbf{G}(\mathbf{X}((\neg b) \vee c))$–$\mathbf{G}(\mathbf{X}\neg b)$–$\mathbf{G}(\neg a)$ that involves a time step of 1 between $\mathbf{G}(\neg a)$ and $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$ as well as between $\mathbf{G}(\neg b)$ and $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$. Hence, for each even i there exists a path such that $\mathbf{G}(\neg a)$ can reach $\Box$ on that path and that path contains i edges involving time steps of 1. Consequently, $\mathbf{G}(\neg a)$ is labeled with $2 \cdot \mathbb{N}$. The same holds for all vertices in rows 5 and 6 that are either on the loop between $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$ and $\mathbf{G}(\neg a)$ or backward reachable from those via blue, dashed edges: $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$, $\mathbf{G}(\mathbf{X}\neg b)$, $\mathbf{G}(\mathbf{X}((\neg b) \vee \neg c))$, and $\mathbf{G}((\neg a) \vee \mathbf{X} b)$. Analogously all vertices in rows 5 and 6 that are on the loop between $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$ and $\mathbf{G}(\neg b)$ or backward reachable from those via blue, dashed edges are labeled with $2 \cdot \mathbb{N} + 1$: $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$, $\mathbf{G}(\mathbf{X}\neg a)$, $\mathbf{G}(\neg b)$, $\mathbf{G}(\mathbf{X}((\neg a) \vee \neg c))$, and $\mathbf{G}((\neg b) \vee \mathbf{X} a)$. Finally, consider $\mathbf{G}((\neg a) \vee \neg c)$ in row 1. It reaches $\Box$ via $\mathbf{G}(\neg a)$ traversing no red, dotted edge, giving $\{0\}$. However, there is also the set of paths through the partition of the second loop search iteration, which uses $2 \cdot \mathbb{N} + 2$ red, dotted edges. Taking both contributions together we obtain $2 \cdot \mathbb{N}$ for this clause.

From now on we assume in this section that the edges and vertices of G' are labeled according to Def. 7 and 8. The following two lemmas are needed to prove correctness of UC extraction in SNF with sets of time points in Thm. 3. They can easily be proved from Def. 7, 8. Proposition 1 establishes that the sets of time points obtained in Def. 8 are semilinear (as suggested for tableaux in [36]). The construction in its proof will later be a fundamental step to actually compute the sets of time points.

Lemma 1. *Any vertex v in G' that is L_V-labeled with an initial clause is $L'_{V'}$-labeled with $\{0\}$.*

Lemma 2. *For each pair of vertices v, v' in G' such that there is an edge from v to v' in G', the labeling $L'_{V'}(v')$ is a (premise 1 of $\boxed{\textit{step-nx}}$, $\boxed{\textit{BFS-loop-it-init-n}}$, $\boxed{\textit{BFS-loop-it-sub}}$, $\boxed{\textit{BFS-loop-conclusion2}}$: time-shifted) subset of the labeling $L'_{V'}(v)$.*

Proposition 1. *For each vertex v in G' the labeling $L'_{V'}(v)$ is a semilinear set.*

Proof. For each vertex v turn the graph G' into a transition-labeled nondeterministic finite automaton (NFA) on finite words over $\{0,1\}$ as follows: 1. The set of states is the set of vertices of the graph G', V'. 2. The set of transitions is the set of *reversed* edges of the graph G'. 3. The labeling of the transitions is given by the $L_{E'}$-labeling of the corresponding edges. 4. The (only) initial state is $v_{\Box}$. 5. The (only) final state is v. Now it's clear from Def. 8 that the $L'_{V'}$-labeling of the vertex v is the Parikh image of the letter 1 of the regular language given by the automaton. The claim follows from Parikh's theorem [28]. □

We now define UCs in SNF with sets of time points. To simplify notation we first define what it means to assign a set of time points to an SNF clause (Def. 9). The definition of a UC in SNF with sets of time points is then immediate in Def. 10. Given the proof of Thm. 1 (see [35]) the proof of correctness in Thm. 3 (in App. B of [34]) can focus on why the construction remains correct with sets of time points. In Prop. 2 we state an upper bound on the complexity of extracting a UC in SNF with sets of time points.

Definition 9. *Let I be a set of time points. Let c be an SNF clause. Then c* with set of time points *I, $\underset{I}{c}$, is the following LTLp formula:*[3]

$$\underset{I}{c} = \begin{cases} ((\underset{I}{\neg})p_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} (\underset{I}{\neg})p_n) & \text{if } c = ((\neg)p_1 \vee \ldots \vee (\neg)p_n) \text{ is an initial clause; or} \\ (\underset{I}{\mathbf{G}}((\underset{I}{\neg})p_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} (\underset{I}{\neg})p_n \underset{I,I}{\vee} (\underset{I+1}{\mathbf{X}} (\underset{I+1}{\neg})q_1 \underset{I+1,I+1}{\vee} \ldots \underset{I+1,I+1}{\vee} (\underset{I+1}{\neg})q_{n'}))) & \\ \quad \text{if } c = (\mathbf{G}(((\neg)p_1 \vee \ldots \vee (\neg)p_n) \vee (\mathbf{X}((\neg)q_1 \vee \ldots \vee (\neg)q_{n'})))) \text{ is a global clause; or} & \\ (\underset{I}{\mathbf{G}}(((\underset{I}{\neg})p_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} (\underset{I}{\neg})p_n) \underset{I,I}{\vee} (\underset{[min(I),\infty)}{\mathbf{F}} ((\underset{[min(I),\infty)}{\neg})l)))) & \text{if } c = (\mathbf{G}(((\neg)p_1 \vee \ldots \vee (\neg)p_n) \vee (\mathbf{F}((\neg)l)))) \text{ is an eventuality clause.} \end{cases}$$

Definition 10. *Let $c_{1,1}, \ldots, c_{1,n_1}$ be the initial clauses in C^{uc}, $c_{2,1}, \ldots, c_{2,n_2}$ the global clauses in C^{uc}, and $c_{3,1}$, …, c_{3,n_3} the eventuality clauses in C^{uc}. Let $v_{m,m'}$ be the unique vertex in the main partition M of G' L_V-labeled with clause $c_{m,m'}$. Let $I_{m,m'}$ be the set of time points that vertex $v_{m,m'}$ is $L'_{V'}$-labeled with in G'. The* UC of C in SNF with sets of time points, *θ^{uc}, is given by*

$$\underset{I_{1,1}}{c_{1,1}} \underset{\{0\},\{0\}}{\wedge} \ldots \underset{\{0\},\{0\}}{\wedge} \underset{I_{1,n_1}}{c_{1,n_1}} \underset{\{0\},\{0\}}{\wedge} \underset{I_{2,1}}{c_{2,1}} \underset{\{0\},\{0\}}{\wedge} \ldots \underset{\{0\},\{0\}}{\wedge} \underset{I_{2,n_2}}{c_{2,n_2}} \underset{\{0\},\{0\}}{\wedge} \underset{I_{3,1}}{c_{3,1}} \underset{\{0\},\{0\}}{\wedge} \ldots \underset{\{0\},\{0\}}{\wedge} \underset{I_{3,n_3}}{c_{3,n_3}}.$$

[3] In this definition $(\neg)$ indicates a negation that may or may not be present.

Theorem 3. *Let θ^{uc} be the UC of C in SNF with sets of time points. Then θ^{uc} is unsatisfiable.*

Proposition 2. *Let θ^{uc} be the UC of C in SNF with sets of time points. Construction of θ^{uc} from G' can be performed in time $\mathcal{O}(|V'|^3 + |V'|^2 \cdot |C|)$.*

Proof. (Sketch) Construct an NFA from G' along the lines of the proof of Prop. 1. Turn the NFA into a unary NFA by regarding edges $L_{E'}$-labeled with 0 as ε-edges and making the NFA ε-free (e.g., [23]). Finally, use an algorithm by Gawrychowski [20] extended to handle all final states in parallel to compute sets of time points. □

We now apply Def. 10 to the example in Fig. 1 and obtain (6) as a UC in SNF with sets of time points. Notice, that all occurrences of a occur at even time points and how both occurrences of b interact at odd time points. Moreover, the last clause shows that only a single occurrence of c is required for unsatisfiability. Finally, the fourth clause has $\neg c$ at even time points, while the fifth clause becomes relevant at odd time points; thus all potential occurrences of c are covered. This concludes this example.

$$\begin{aligned} & a \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}}{\mathbf{G}}((\underset{2\cdot\mathbb{N}}{\neg} a) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\vee} \underset{2\cdot\mathbb{N}+1}{\mathbf{X}} b)) \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}+1}{\mathbf{G}}((\underset{2\cdot\mathbb{N}+1}{\neg} b) \underset{2\cdot\mathbb{N}+1,2\cdot\mathbb{N}+1}{\vee} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} a)) \\ \underset{\{0\},\{0\}}{\wedge} & (\underset{2\cdot\mathbb{N}}{\mathbf{G}}((\underset{2\cdot\mathbb{N}}{\neg} a) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\vee} \underset{2\cdot\mathbb{N}}{\neg} c)) \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}+1}{\mathbf{G}}((\underset{2\cdot\mathbb{N}+1}{\neg} c) \underset{2\cdot\mathbb{N}+1,2\cdot\mathbb{N}+1}{\vee} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\neg} a)) \underset{\{0\},\{0\}}{\wedge} (\underset{\{0\}}{\mathbf{G}}(\underset{\mathbb{N}}{\mathbf{F}} c)) \end{aligned} \tag{6}$$

Definition 11 adds sets of time points to a UC in LTL by transferring them from a UC in SNF with time points to a UC in LTL. The proof idea for Thm. 4 (in App. B of [34]) is similar to that of Thm. 2 (see [35]), but in addition we need to define a translation from the corresponding fragment of LTLp to SNF with sets of time points, which must be shown to be satisfiability- but not unsatisfiability-preserving.

Definition 11. *Let ϕ be an unsatisfiable LTL formula, let SNF(ϕ) be its SNF, let ϕ^{uc} be the UC of ϕ in LTL, and let θ^{uc} be the UC of SNF(ϕ) in SNF with sets of time points. Construct the* UC of ϕ in LTL with sets of time points, *θ'^{uc}, by assigning a set of time points I to each occurrence of a subformula ψ in ϕ^{uc} as follows. Let $I', I'', \ldots$ be the sets of time points of the occurrences of the proposition x_ψ in θ^{uc} that are marked blue boxed in Tab. 3. Then assign the occurrence of ψ in ϕ^{uc} the set of time points I that is the union of $I', I'', \ldots$.*

Theorem 4. *Let ϕ be an unsatisfiable LTL formula, and let θ'^{uc} be the UC of ϕ in LTL with sets of time points. Then θ'^{uc} is unsatisfiable.*

It's easy to see that no subformula in (1) or (4) can be replaced with 1 (for positive polarity occurrences) or 0 (for negative polarity occurrences) without making (1) or (4) satisfiable. I.e., (1) or (4) are the only UCs of themselves according to Def. 10 in [36] (and, hence, according to Def. 4 in [34]). The corresponding UCs in LTL with sets of time points in (3) and (5) show that UCs with sets of time points can be more fine-grained than UCs without.

7 Example

In this section we present an example that shows the utility of UCs with sets of time points for debugging that is closer to a real world situation. The UCs in this as well as in all other examples in this paper were obtained with our implementation, possibly except for minor rewriting.

The example (7) in Fig. 2 reuses the example of a lift specification from [35] (originally adapted from [22]) but extends it with sets of time points to show that understanding the presence of a problem becomes easier. The lift has two floors, indicated by f_0 and f_1. On each floor there is a button to call the

$$(\neg u) \wedge (f_0) \wedge (\neg b_0) \wedge (\neg b_1) \wedge (\neg up) \tag{7a}$$
$$\wedge\, (\mathbf{G}((u \rightarrow \neg \mathbf{X}u) \wedge ((\neg \mathbf{X}u) \rightarrow u))) \tag{7b}$$
$$\wedge\, (\mathbf{G}(f_0 \rightarrow \neg f_1)) \tag{7c}$$
$$\wedge\, (\mathbf{G}((f_0 \rightarrow \mathbf{X}(f_0 \vee f_1)) \wedge (f_1 \rightarrow \mathbf{X}(f_0 \vee f_1)))) \tag{7d}$$
$$\wedge\, (\mathbf{G}(u \rightarrow ((f_0 \rightarrow \mathbf{X}f_0) \wedge ((\mathbf{X}f_0) \rightarrow f_0) \wedge (f_1 \rightarrow \mathbf{X}f_1) \wedge ((\mathbf{X}f_1) \rightarrow f_1)))) \tag{7e}$$
$$\wedge \begin{array}{l}(\mathbf{G}(((\neg u) \rightarrow \\ \qquad ((b_0 \rightarrow \mathbf{X}b_0) \wedge ((\mathbf{X}b_0) \rightarrow b_0) \wedge (b_1 \rightarrow \mathbf{X}b_1) \wedge ((\mathbf{X}b_1) \rightarrow b_1)))))\end{array} \tag{7f}$$
$$\wedge\, (\mathbf{G}(((b_0 \wedge \neg f_0) \rightarrow \mathbf{X}b_0) \wedge ((b_1 \wedge \neg f_1) \rightarrow \mathbf{X}b_1))) \tag{7g}$$
$$\wedge\, (\mathbf{G}((f_0 \wedge \mathbf{X}f_0) \rightarrow ((up \rightarrow \mathbf{X}up) \wedge ((\mathbf{X}up) \rightarrow up)))) \tag{7h}$$
$$\wedge\, (\mathbf{G}((f_1 \wedge \mathbf{X}f_1) \rightarrow ((up \rightarrow \mathbf{X}up) \wedge ((\mathbf{X}up) \rightarrow up)))) \tag{7i}$$
$$\wedge\, (\mathbf{G}(((f_0 \wedge \mathbf{X}f_1) \rightarrow up) \wedge ((f_1 \wedge \mathbf{X}f_0) \rightarrow \neg up))) \tag{7j}$$
$$\wedge\, (\mathbf{G}((sb \rightarrow (b_0 \vee b_1)) \wedge ((b_0 \vee b_1) \rightarrow sb))) \tag{7k}$$
$$\wedge\, (\mathbf{G}(((f_0 \wedge \neg sb) \rightarrow (f_0 \mathbf{U}(sb \mathbf{R}((\mathbf{F}f_0) \wedge (\neg up))))))) \tag{7l}$$
$$\wedge\, (\mathbf{G}(((f_1 \wedge \neg sb) \rightarrow (f_1 \mathbf{U}(sb \mathbf{R}((\mathbf{F}f_0) \wedge (\neg up))))))) \tag{7m}$$
$$\wedge\, (\mathbf{G}((b_0 \rightarrow \mathbf{F}f_0) \wedge (b_1 \rightarrow \mathbf{F}f_1))) \tag{7n}$$

Figure 2: A lift specification.

lift (b_0, b_1). sb is 1 if some button is pressed. If the lift moves up, then up must be 1; if it moves down, then up must be 0. u switches turns between actions by users of the lift (u is 1) and actions by the lift (u is 0). For a more detailed explanation we refer to [22].

We first assume that an engineer is interested in seeing whether it is possible that b_1 is continuously pressed (8). As the UC (9) shows this is impossible as b_1 must be 0 at time point 0. Notice that (9) indicates that the argument of the **G** operator is only needed at time point 0 (trivial to see in this case).

$$\mathbf{G}b_1 \tag{8}$$

$$(\underset{\{0\}}{\neg}\, b_1) \underset{\{0\},\{0\}}{\wedge} \underset{\{0\}}{\mathbf{G}}\, b_1 \tag{9}$$

Now the engineer modifies her query such that b_1 is pressed only from time point 1 on (10). That is impossible, too; as the UC in (11) shows also this time the press of b_1 is required only at one time point.

$$\mathbf{XG}b_1 \tag{10}$$

$$(\underset{\{0\}}{\neg}\, u) \underset{\{0\},\{0\}}{\wedge} ((\underset{\{0\}}{\neg}\, b_1) \underset{\{0\},\{0\}}{\wedge} ((\underset{\{0\}}{\mathbf{G}} ((\underset{\{0\}}{\neg}\, u) \underset{\{0\},\{0\}}{\rightarrow} ((\underset{\{1\}}{\mathbf{X}}\, b_1) \underset{\{0\},\{0\}}{\rightarrow} b_1))) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{X}}\, \underset{\{1\}}{\mathbf{G}}\, b_1))) \tag{11}$$

The engineer now tries to have b_1 pressed only from time point 2 on and also obtains a UC that needs b_1 pressed only at a single time point (not shown). She becomes suspicious and checks whether b_1 can be pressed at all. She now sees that b_1 cannot be pressed at any time point and, therefore, this specification of a lift must contain a bug. This example illustrates the benefits of UCs with sets of time points, as (9) and (11) make it clear that b_1 being 1 is only needed at a single time point for unsatisfiability.

For an example showing disjuncts of an invariant holding at different time points see App. C of [34].

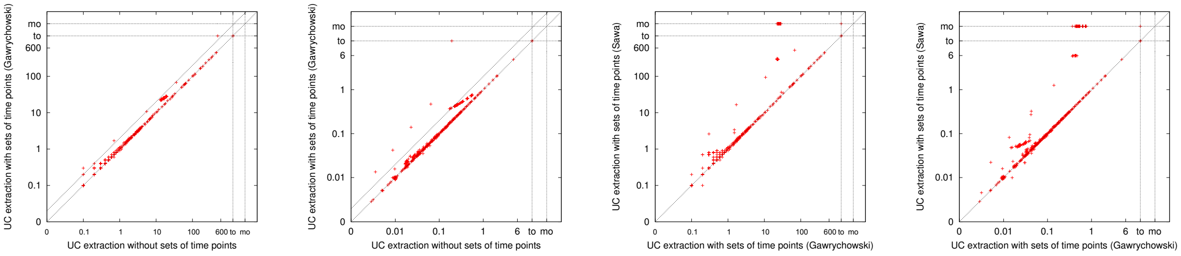
8 Experimental Evaluation

We use the version of TRP++ extended with extraction of UCs from [35] as the basis for our implementation. We implemented extraction of sets of time points along the lines of the proofs of Prop. 1, 2. To make an NFA ε-free we use a standard algorithm that performs DFS from each state to find the sets of states that are reachable via a sequence of ε-edges, inserts 1-edges between pairs of vertices v, v' such that v can reach v' by reading $\varepsilon^* 1 \varepsilon^*$, and removes ε-edges (e.g., [23]). To compute Parikh images for unary NFAs we implemented an algorithm by Gawrychowski [20] and one by Sawa [32, 33]. Both assume a single set of final states leading to a single Parikh image. We, however, have one final state for each SNF clause in the UC in SNF, each of which we need to assign a separate Parikh image. We adapted Gawrychowski's algorithm to our setting by computing the Parikh images for different final states in a single run of the algorithm. Similarly, we optimized Sawa's algorithm by computing parts that are common for different final states only once and by heuristically accelerating some of its steps.

Our examples are based on [37]. In categories **crafted** and **random** and in family **forobots** we considered all unsatisfiable instances from [37]. The version of **alaska_lift** used here contains a small bug fix: in [39, 37] the subformula $\mathbf{X}u$ was erroneously written as literal Xu. Combining 2 variants

category	family	source	# solved UC w/o s.o.t.p.	# solved UC w/ s.o.t.p. (Gawrychowski)	# solved UC w/ s.o.t.p. (Sawa)	|largest solved|
application	**alaska_lift**	[22, 39]	71	71	71	4605
	anzu_genbuf	[10]	16	16	16	2676
	forobots	[8]	25	25	25	635
crafted	**schuppan_O1formula**	[37]	21	21	21	1606
	schuppan_O2formula	[37]	8	7	7	91
	schuppan_phltl	[37]	4	4	4	125
random	**rozier_random**	[31]	66	66	66	157
	trp	[26]	397	397	345	1421

Table 5: Overview of benchmark families.



(a) run time [sec] (b) memory [GB] (c) run time [sec] (d) memory [GB]

Figure 3: Overhead of UC extraction with sets of time points: (a) and (b) show run time and memory for UC extraction with sets of time points using Gawrychowski's algorithm (y-axis) versus UC extraction without sets of time points (x-axis). (c) and (d) compare run time and memory for Gawrychowski's algorithm (x-axis) and Sawa's algorithm (y-axis) for UC extraction with sets of time points. The off-center diagonal in (a) and (b) shows where $y = 2x$.

of **alaska_lift** with 3 different scenarios we obtain 6 subfamilies of **alaska_lift**. For **anzu_genbuf** we invented 3 scenarios to obtain 3 subfamilies. For all benchmark families that consist of a sequence of instances of increasing difficulty we stopped after two instances that could not be solved due to time or memory out. Some instances were simplified to 0 during the translation from LTL to SNF; these instances were discarded. In Tab. 5 we give an overview of the benchmark families. Columns 1–3 give the category, name, and the source of the family. Columns 4–6 list the numbers of instances that were solved by our implementation with UC extraction without sets of time points, with UC extraction with sets of time points using Gawrychowski's algorithm, and with UC extraction with sets of time points using Sawa's algorithm. Column 7 indicates the size (number of nodes in the syntax tree) of the largest instance solved with UC extraction without sets of time points.

The experiments were performed on a laptop with Intel Core i7 M 620 processor at 2 GHz running Ubuntu 10.04. Run time and memory usage were measured with `run` [9]. The time and memory limits were 600 seconds and 6 GB.

In Fig. 3 (a) and (b) we show the overhead that is incurred by extracting UCs with sets of time points. An analysis by category (for plots see App. D of [34]) shows that the overhead for instances of the **application** category is at most 50 % for UC extraction with sets of time points using Gawrychowski's algorithm over UC extraction without sets of time points. Figure 3 (c) and (d) compare using Gawrychowski's and Sawa's algorithm for computing sets of time points. In Tab. 6 we show which sets of time points occur in the UCs of which benchmark families.

Our data show that extraction of UCs with sets of time points is possible with quite acceptable overhead in run time and memory usage (Fig. 3 (a), (b)). In particular, out of the 702 instances we considered with UC extraction without sets of time points, a UC was obtained for 608. With sets of time

category	family	$\{0\}$	$\{0,1\}$	$\{0,2\}$	$\{0,\mathbb{N}+2\}$	$\{1\}$	$\{1,2\}$	$\{1,2,3\}$	$\{1,3\}$	$\{2\}$	$\{2,3\}$	$\{3\}$	$\mathbb{N}$	$\mathbb{N}+1$	$\mathbb{N}+2$	$\mathbb{N}+3$	$\mathbb{N}+4$	$\mathbb{N}+5$	$\mathbb{N}+6$	$\mathbb{N}+7$	$\mathbb{N}+8$	$\mathbb{N}+9$	$\mathbb{N}+10$	$4\cdot\mathbb{N}$	$4\cdot\mathbb{N}+1$	$\{4\cdot\mathbb{N}+1,$ $4\cdot\mathbb{N}+2\}$	$\{4\cdot\mathbb{N}+1,$ $4\cdot\mathbb{N}+2,$ $4\cdot\mathbb{N}+3\}$	$4\cdot\mathbb{N}+2$	$\{4\cdot\mathbb{N}+2,$ $4\cdot\mathbb{N}+3\}$	$\{4\cdot\mathbb{N}+2,$ $4\cdot\mathbb{N}+3,$ $4\cdot\mathbb{N}+4\}$	$4\cdot\mathbb{N}+3$	$\{4\cdot\mathbb{N}+3,$ $4\cdot\mathbb{N}+4\}$	$4\cdot\mathbb{N}+4$	$4\cdot\mathbb{N}+5$	$\{5\cdot\mathbb{N}+0\},$ $\ldots,$ $\{5\cdot\mathbb{N}+5\}$	$\{12\cdot\mathbb{N}+0\},$ $\ldots,$ $\{12\cdot\mathbb{N}+12\}$
application	**alaska_lift**	✓	✓			✓	✓			✓			✓	✓																						
	anzu_genbuf	✓				✓							✓	✓	✓	✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
	forobots	✓	✓			✓							✓	✓	✓	✓																				
crafted	**schuppan_O1formula**	✓				✓																														
	schuppan_O2formula	✓											✓	✓																						
	schuppan_phltl	✓											✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓													
random	**rozier_random**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓																		
	trp	✓				✓							✓	✓	✓	✓																			✓	✓

Table 6: Occurrences of sets of time points in UCs: A ✓ in a field indicates that a subformula in a UC of that benchmark family is assigned that set of time points.

points enabled only once instance less is solved.

Sets of time points often provide helpful information. For some subfamilies of the **anzu_genbuf** and **trp** families they show that some subformulas are required only every 4th, 5th, or 12th time point. For an instance of the **forobots** family they make it clear that only the first two time points are relevant, although some of the subformulas involved are **G** subformulas. For the **schuppan_phltl** family (a temporal version of the pigeon hole problem; n pigeon holes are turned into a single pigeon hole over n time points) they indicate how the conditions of mutual exclusivity for the hole are invoked one after the other.

Gawrychowski's algorithm [20] has better worst case complexity than Sawa's algorithm [32]. We also found it easier to understand and implement. On our benchmarks Gawrychowski's algorithm tends to perform better than Sawa's algorithm (Fig. 3 (c) and (d)), especially when the NFAs become larger.

9 Conclusions

In this paper we showed how to obtain information on the time points at which subformulas of a UC for LTL are required for unsatisfiability, providing useful information in many cases and leading to a more fine-grained notion of UC than in [36]. We demonstrated with an implementation in TRP++ that UCs with sets of time points can be extracted efficiently. Potential future work includes extending the computation of sets of time points to tableau-based UC extraction for LTL such as [21] and exploring whether computation of sets of time points is feasible for BDD-based algorithms via, e.g., [27]. Another question is how to apply the idea of LTL with sets of time points to unrealizable cores (e.g., [36]). Finally, it would be interesting to see whether/how minimal or minimum sets of time points can be obtained, where $\leq$ is set inclusion (rather than syntactic expression size).

Acknowledgements I thank B. Konev and M. Ludwig for making TRP++ and TSPASS including their LTL translators available. I also thank A. Cimatti for bringing up the subject of temporal resolution and for pointing out that the resolution graph can be seen as a regular language acceptor. Initial parts of the work were performed while working under a grant by the Provincia Autonoma di Trento (project EMTELOS).

References

[1] Available at http://www.csc.liv.ac.uk/~konev/software/trp++/.

[2] Available at http://www.schuppan.de/viktor/qapl13/.

[3] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer & M. Vardi (2003): *Enhanced Vacuity Detection in Linear Temporal Logic*. In W. Hunt Jr. & F. Somenzi, editors: *CAV*, *LNCS* 2725, Springer, pp. 368–380, doi:10.1007/978-3-540-45069-6_35.

[4] A. Awad, R. Goré, Z. Hou, J. Thomson & M. Weidlich (2012): *An iterative approach to synthesize business process templates from compliance rules*. *Inf. Syst.* 37(8), pp. 714–736, doi:10.1016/j.is.2012.05.001.

[5] R. Bakker, F. Dikker, F. Tempelman & P. Wognum (1993): *Diagnosing and Solving Over-Determined Constraint Satisfaction Problems*. In: *IJCAI*, pp. 276–281. Available at http://ijcai.org/Past%20Proceedings/IJCAI-93-VOL1/PDF/039.pdf.

[6] I. Beer, S. Ben-David, H. Chockler, A. Orni & R. Trefler (2009): *Explaining Counterexamples Using Causality*. In A. Bouajjani & O. Maler, editors: *CAV*, *LNCS* 5643, Springer, pp. 94–108, doi:10.1007/978-3-642-02658-4_11.

[7] I. Beer, S. Ben-David, C. Eisner & Y. Rodeh (2001): *Efficient Detection of Vacuity in Temporal Model Checking*. *FMSD* 18(2), pp. 141–163, doi:10.1023/A:1008779610539.

[8] A. Behdenna, C. Dixon & M. Fisher (2009): *Deductive Verification of Simple Foraging Robotic Behaviours*. *Int. J. of Intelligent Comput. and Cybernetics* 2(4), pp. 604–643, doi:10.1108/17563780911005818.

[9] A. Biere & T. Jussila: *Benchmark Tool Run*. Available at http://fmv.jku.at/run/.

[10] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli & M. Weiglhofer (2007): *Specify, Compile, Run: Hardware from PSL*. In S. Glesner, J. Knoop & R. Drechsler, editors: *COCV*, *ENTCS* 190(4), Elsevier, pp. 3–16, doi:10.1016/j.entcs.2007.09.004.

[11] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta & B. Vittorini (2010): *Formalization and validation of a subset of the European Train Control System*. In J. Kramer, J. Bishop, P. Devanbu & S. Uchitel, editors: *ICSE (2)*, ACM, pp. 109–118, doi:10.1145/1810295.1810312.

[12] E. Clarke, M. Talupur, H. Veith & D. Wang (2003): *SAT Based Predicate Abstraction for Hardware Verification*. In E. Giunchiglia & A. Tacchella, editors: *SAT*, *LNCS* 2919, Springer, pp. 78–92, doi:10.1007/978-3-540-24605-3_7.

[13] C. Dixon (1995): *Strategies for Temporal Resolution*. Ph.D. thesis, Department of Computer Science, University of Manchester. Available at ftp://ftp.cs.man.ac.uk/pub/TR/UMCS-95-12-1.ps.Z.

[14] C. Dixon (1997): *Using Otter for Temporal Resolution*. In H. Barringer, M. Fisher, D. Gabbay & G. Gough, editors: *ICTL*, Applied Logic Series, Kluwer, pp. 149–166.

[15] C. Dixon (1998): *Temporal Resolution Using a Breadth-First Search Algorithm*. *Ann. Math. Artif. Intell.* 22(1-2), pp. 87–115, doi:10.1023/A:1018942108420.

[16] C. Eisner & D. Fisman (2006): *A Practical Introduction to PSL*. Springer, doi:10.1007/978-0-387-36123-9.

[17] E. Emerson (1990): *Temporal and Modal Logic*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, Elsevier and MIT Press, pp. 995–1072.

[18] M. Fisher (1991): *A Resolution Method for Temporal Logic*. In: *IJCAI*, pp. 99–104. Available at http://ijcai.org/Past%20Proceedings/IJCAI-91-VOL1/PDF/017.pdf.

[19] M. Fisher, C. Dixon & M. Peim (2001): *Clausal temporal resolution*. *ACM Trans. Comput. Log.* 2(1), pp. 12–56, doi:10.1145/371282.371311.

[20] P. Gawrychowski (2011): *Chrobak Normal Form Revisited, with Applications*. In B. Bouchou-Markhoff, P. Caron, J. Champarnaud & D. Maurel, editors: *CIAA*, *LNCS* 6807, Springer, pp. 142–153, doi:10.1007/978-3-642-22256-6_14.

[21] F. Hantry & M. Hacid (2011): *Handling Conflicts in Depth-First Search for LTL Tableau to Debug Compliance Based Languages*. In E. Pimentel & V. Valero, editors: *FLACOS*, *EPTCS* 68, pp. 39–53, doi:10.4204/EPTCS.68.5.

[22] A. Harding (2005): *Symbolic Strategy Synthesis For Games With LTL Winning Conditions*. Ph.D. thesis, University of Birmingham.

[23] J. Hopcroft & J. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

[24] U. Hustadt & B. Konev (2003): *TRP++ 2.0: A Temporal Resolution Prover*. In F. Baader, editor: *CADE*, *LNCS* 2741, Springer, pp. 274–278, doi:10.1007/978-3-540-45085-6_21.

[25] U. Hustadt & B. Konev (2004): *TRP++: A temporal resolution prover*. In M. Baaz, J. Makowsky & A. Voronkov, editors: *Collegium Logicum*, 8, Kurt Gödel Society, pp. 65–79.

[26] U. Hustadt & R. A. Schmidt (2002): *Scientific Benchmarking with Temporal Logic Decision Procedures*. In D. Fensel, F. Giunchiglia, D. McGuinness & M. Williams, editors: *KR*, Morgan Kaufmann, pp. 533–546.

[27] T. Jussila, C. Sinz & A. Biere (2006): *Extended Resolution Proofs for Symbolic SAT Solving with Quantification*. In A. Biere & C. Gomes, editors: *SAT*, *LNCS* 4121, Springer, pp. 54–60, doi:10.1007/11814948_8.

[28] R. Parikh (1966): *On Context-Free Languages*. *J. ACM* 13(4), pp. 570–581, doi:10.1145/321356.321364.

[29] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem & A. Cimatti (2006): *Formal analysis of hardware requirements*. In E. Sentovich, editor: *DAC*, ACM, pp. 821–826, doi:10.1145/1146909.1147119.

[30] K. Ravi & F. Somenzi (2004): *Minimal Assignments for Bounded Model Checking*. In K. Jensen & A. Podelski, editors: *TACAS*, *LNCS* 2988, Springer, pp. 31–45, doi:10.1007/978-3-540-24730-2_3.

[31] K. Rozier & M. Vardi (2010): *LTL satisfiability checking*. *STTT* 12(2), pp. 123–137, doi:10.1007/s10009-010-0140-3.

[32] Z. Sawa: *Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata*. Accepted for publication. Available at `http://www.cs.vsb.cz/sawa/papers/fi2011.pdf`.

[33] Z. Sawa (2010): *Efficient Construction of Semilinear Representations of Languages Accepted by Unary NFA*. In A. Kucera & I. Potapov, editors: *RP*, *LNCS* 6227, Springer, pp. 176–182, doi:10.1007/978-3-642-15349-5_12.

[34] V. Schuppan (2012): *Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance (full version)*. Available at `http://www.schuppan.de/viktor/qapl13/VSchuppan-QAPL-2013-full.pdf`.

[35] V. Schuppan (2012): *Extracting Unsatisfiable Cores for LTL via Temporal Resolution*. Submitted. Available at `arXiv:1212.3884v1 [cs.LO]`.

[36] V. Schuppan (2012): *Towards a notion of unsatisfiable and unrealizable cores for LTL*. *Sci. Comput. Program.* 77(7-8), pp. 908–939, doi:10.1016/j.scico.2010.11.004.

[37] V. Schuppan & L. Darmawan (2011): *Evaluating LTL Satisfiability Solvers*. In T. Bultan & P. Hsiung, editors: *ATVA*, *LNCS* 6996, Springer, pp. 397–413, doi:10.1007/978-3-642-24372-1_28.

[38] J. Simmonds, J. Davies, A. Gurfinkel & M. Chechik (2010): *Exploiting resolution proofs to speed up LTL vacuity detection for BMC*. *STTT* 12(5), pp. 319–335, doi:10.1007/s10009-009-0134-1.

[39] M. De Wulf, L. Doyen, N. Maquet & J. Raskin (2008): *Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking*. In C. Ramakrishnan & J. Rehof, editors: *TACAS*, *LNCS* 4963, Springer, pp. 63–77, doi:10.1007/978-3-540-78800-3_6.