Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance (full version; r730, June 11, 2013)

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

Motivation LTL (e.g., [Pnu77,Eme90]) and its relatives are important specification languages for reactive systems (e.g., [EF06]) and for business processes (e.g., [PA06]). Experience in verification (e.g., [Bee+01,Kup06]) and in synthesis (e.g., [Blo+07]) 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., [Chi+10]) 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 (e.g., [Sch12b,Bak+93,CD91]); this is called an unsatisfiable core (UC) (e.g., [Sch12b,GN03,ZM03b,Hoo99]).

Less simplistic ways to examine an LTL specification ϕ exist [Pil+06], 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., [Bee+01,KV03,Arm+03,GC04,Fis+08, Kup06]).

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., [Bee+09] for some references). In

Full version of a paper with the same title in:
L. Bortolussi and H. Wiklicky (Eds.): QAPL 2013
EPTCS 117, 2013, pp. 49–65, doi:10.4204/EPTCS.117.4

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., [RS04,Bee+09]). According to [Bee+09] 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 [Sch12b], are an important feature of industrial hardware verification tools (see, e.g., [Bee+01,Arm+03]). In the academic world UCs are an important part of design methods for embedded systems (e.g., [Pil+06]) as well as for business processes (e.g., [Awa+12]). Despite this relevance of UCs efforts to provide additional information in the context of UCs or vacuity have remained isolated (e.g., [Sim+10]).

Example 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.

$$(\mathbf{G}p) \wedge (\mathbf{X}\neg p) \tag{1}$$

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).

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

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. (2) can be enhanced with this information by replacing $\mathbb{N}$ below $\mathbf{G}$ with $\{1\}$, obtaining (3). (3) can be seen as a UC of (1).

$$(\underset{\{1\}}{\mathbf{G}}\, p) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg} p) \tag{3}$$

Approach We obtain the desired information as follows. We use a temporal resolution-based [Fis91, FDP01] solver to determine unsatisfiability of a given LTL formula. During the execution of the solver we construct a resolution graph [Sch12a]. In the resolution graph we distinguish between edges where a premise is time-shifted 1 step into the future with respect to the conclusion and others where this is not the case. An example for an inference that involves such a time step is concluding $\mathbf{G}(p)$ from premises $\mathbf{G}(p \vee \mathbf{X}\neg q)$ and $\mathbf{G}(q)$: the second premise $\mathbf{G}(q)$ is implicitly converted to $\mathbf{G}(\mathbf{X}q)$ to perform the inference. I.e., the resulting edge from the second premise $\mathbf{G}(q)$ to the conclusion $\mathbf{G}(p)$ involves a time step of 1, while this is not the case for the edge from the first premise $\mathbf{G}(p \vee \mathbf{X}\neg q)$ to the conclusion. To determine the sets of time points at which a clause c in an unsatisfiable set of SNF clauses (temporal resolution works on a clausal normal form called SNF [Fis91,FN92,FDP01]) is relevant for unsatisfiability we fix time point 0 as the time point at which the contradiction $\square$, which concludes the proof, is happening. Then we count the number of time-shifts that occur on paths from c to $\square$ in the resolution

graph: c is relevant for unsatisfiability at time point i iff there exists a path from c to $\Box$ in the resolution graph that traverses exactly i edges involving a time step of 1. Note that the resolution graph may contain loops. Therefore, to actually count the number of time-shifts on all possible paths from some clause c to the contradiction $\Box$, we regard the reversed resolution graph as a nondeterministic finite automaton over the language $\{0,1\}$ with initial state $\Box$ and final state c. The desired information is then simply the Parikh image (e.g., [Sal73]) of the letter 1 in the language given by the automaton. Parikh's theorem [Par66] implies that the resulting sets of time points are semilinear. To compute the Parikh images we use algorithms by Gawrychowski [Gaw11] or Sawa [Saw13]. For $|C|$ input SNF clauses and a resolution graph with $|V'|$ vertices backward reachable from $\Box$ the information can be computed in time $\mathcal{O}(|V'|^3 + |V'|^2 \cdot |C|)$. Our experimental evaluation shows that the resulting overhead compared to extraction of UCs without sets of time points is acceptable in practice.

Contributions In this paper we make the following contributions. (i) 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 [Sch12b]. (ii) We propose a method to obtain that information from a temporal resolution proof of unsatisfiability of an LTL formula. (iii) 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.

(Not) Just Debugging 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., [Cla+03,Cim+07]). While our results might also benefit that application, we focus on debugging.

Temporal Resolution as a Basis Temporal resolution [Fis91,FDP01] 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++` [HK03,HK04,trp++] proved to be competitive in a recent evaluation of solvers for LTL satisfiability, in particular on unsatisfiable instances (see pp. 51–55 of the full version of [SD11]). Second, a temporal resolution proof naturally induces a resolution graph [Sch12a], which provides a clean framework for extracting information from the proof. Note, that while the BDD-based solver `NuSMV` [Cim+02] also performed well on unsatisfiable instances in [SD11], the BDD layer makes extraction of information from the proof more involved. On the other hand, the tableau-based solvers `LWB` [Heu+95] and `pltl` [pltl] provide access to a proof of unsatisfiability comparable to temporal resolution, yet tended to perform not as good on unsatisfiable instances in [SD11].

Notion of Relevance 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., [RS04, Bee+09]. 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 (e.g., [Awa+12,ZM03a,CD91]).

Related Work Simmonds et al. [Sim+10] use SAT-based bounded model checking (e.g., [Bie+99, Bie09]) for vacuity detection. They indicate which time points are relevant for showing that a variable is non-vacuous. They only consider k-step 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. [Sch12b] 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 [Sch12b] 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 [RS04,Bee+09] determines the time points at which propositions in witnesses of satisfiable LTL formulas are relevant for satisfiability. For more general related work on UC extraction see [Sch12a,Sch12b].

Structure of the Paper This paper builds on a fair amount of previous work: we use temporal resolution as implemented in TRP++ [FDP01,HK03,HK04,trp++] and its extension to extract UCs [Sch12a]. 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 [Fis91], and we refer to [FDP01] for a general overview, to [Dix98,Dix97,Dix96,Dix95] for details on loop search, and to [HK03,HK04,trp++] for the implementation in TRP++. In [Sch12a] we provide some intuition on temporal resolution with a slant towards BDD-based symbolic model checking (e.g., [Bur+92,CGP01]). 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 [Sch12a]. 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 in the main part; these can be found in the appendices. For our implementation, examples, and log files see [www].

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 [Par66] states that for every context-free language L, for every letter σ, the Parikh image $\Psi(L, \sigma)$ is semilinear. See also [Sal73].

We use a standard version of LTL, see, e.g., [Eme90]. 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.

$(\pi,i) \models 1, \not\models 0$			$(\pi,i) \models p$	$\Leftrightarrow$	$p \in \pi[i]$
$(\pi,i) \models \neg\psi$	$\Leftrightarrow$	$(\pi,i) \not\models \psi$	$(\pi,i) \models \mathbf{X}\psi$	$\Leftrightarrow$	$(\pi,i+1) \models \psi$
$(\pi,i) \models \psi \vee \psi'$	$\Leftrightarrow$	$(\pi,i) \models \psi$ 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 $\mathbb{N}$. π satisfies ϕ iff $(\pi,0) \models \phi$.

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 $P \equiv \bigvee_{i=1\ldots n} p_i$, $Q \equiv \bigvee_{i=1\ldots n'} q_i$, $R \equiv \bigvee_{i=1\ldots n''} r_i$, $S \equiv \bigvee_{i=1\ldots n'''} s_i$.

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 $\psi \rightarrow \psi'$ (implies) as an abbreviation for $\neg\psi \vee \psi'$, $\psi \leftrightarrow \psi'$ (equivalent) for $(\psi \rightarrow \psi') \wedge (\psi' \rightarrow \psi)$, and $\psi\mathbf{W}\psi$ (weak until) for $(\psi\mathbf{U}\psi') \vee \mathbf{G}\psi$. 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.

3 Temporal Resolution (TR) in TRP++

TR works on formulas in a clausal normal form called separated normal form (SNF) [Fis91,FN92, FDP01]. 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 (i) $(p_1 \vee \ldots \vee p_n)$ is an *initial clause*; (ii) $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{X}(q_1 \vee \ldots \vee q_{n'}))))$ is a *global clause*; and (iii) $(\mathbf{G}((p_1 \vee \ldots \vee p_n) \vee (\mathbf{F}(l))))$ 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 [Fis91,FN92,FDP01].

We now describe TR [Fis91,FDP01] as implemented in TRP++ [HK03,HK04,trp++]. 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++ [HK04]. 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 (e.g., [Rob65,FM09,BG01]). The corre-

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 ← C; if □ ∈ M then return unsat;
2  saturate(M); if □ ∈ M then return unsat;
3  augment(M);
4  saturate(M); if □ ∈ M then return unsat;
5  M' ← ∅;
6  while M' ≠ M do
7      M' ← M;
8      for c ∈ C . c is an eventuality clause do
9          C' ← {□};
10         repeat
11             initialize-BFS-loop-search-iteration(M, c, C', L);
12             saturate-step-xx(L);
13             C' ← {c' ∈ L | c' has empty X part};
14             C'' ← {(G(Q)) | (G((0) ∨ (X(Q ∨ l)))) ∈ L generated by [BFS-loop-it-init-c]};
15             found ← subsumes(C', C'');
16         until found or C' = ∅ ;
17         if found then
18             derive-BFS-loop-search-conclusions(c, C', M);
19             saturate(M); if □ ∈ M then return unsat;
20 return sat;
```

sponding 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 [init-ii], [init-in], [step-nn], and [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 [aug1] to each eventuality clause in M and [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 [step-xx] as a subroutine (line 12). Therefore, each loop search 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 [BFS-loop-it-init-x] once for each global clause with non-empty $\mathbf{X}$ part in M, rule [BFS-loop-it-init-n] once for each global clause with empty $\mathbf{X}$ part in M, and rule [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 [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 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 **X** part were derived (line 16). If a loop has been found, rules BFS-loop-conclusion1 and BFS-loop-conclusion2 are applied once to each global clause with empty **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 [Sch12a] that show how to construct UCs via TR. In Sec. 6 we extend this construction to include information on when parts of a UC are relevant for unsatisfiability. Then we present an example for extraction of a UC in SNF, which is extended with sets of time points in Sec. 6.

Extraction of UCs in SNF 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 [Sch12a]) is therefore that certain parts of the TR proof do not need to be taken into account when determining C^{uc}.

Definition 1 (Resolution Graph)**.** *[Sch12a] 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 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 (UC in SNF)**.** *[Sch12a] 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 $v_\square$ be the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\square$. Let G' be the smallest subgraph of G that contains $v_\square$ and all vertices in G (and the corresponding edges) that are backward reachable from $v_\square$. 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: $C^{uc} = \{c \in C \mid \exists v \in V_{G'} \,.\, L_V(v) = c \wedge v \in M^V\}$.*

[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.

Theorem 1 (Unsatisfiability of UC in SNF)**.** *[Sch12a] 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.*

From LTL to SNF and Back 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 [Sch12a].

Definition 3 (Translation from LTL to SNF)**.** *[Sch12a] 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 (Mapping a UC in SNF to a UC in LTL)**.** *[Sch12a] 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 blue boxed in Tab. 3. (We are sloppy in that we "replace" subformulas of replaced subformulas, while in effect they simply vanish.)*

Theorem 2 (Unsatisfiability of UC in LTL)**.** *[Sch12a] Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be the UC of ϕ in LTL. Then ϕ^{uc} is unsatisfiable.*

Example 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 $\square$ signaling 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 $\square$, 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 $\square$ by saturation (line 19).

The clauses that are backward reachable from $\square$ 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 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. Note that this can be seen as an extension of a notion of UC in [Sch12b,KV03], where subformulas are replaced with 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 (LTLp Syntax)**.** *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.

Definition 6 (LTLp Semantics)**.** *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.

[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 $\mathbf{G}((\neg a) \vee \neg c)$; [HK03,BG01]). Both are issues of completeness of TR and, therefore, not discussed in this paper.

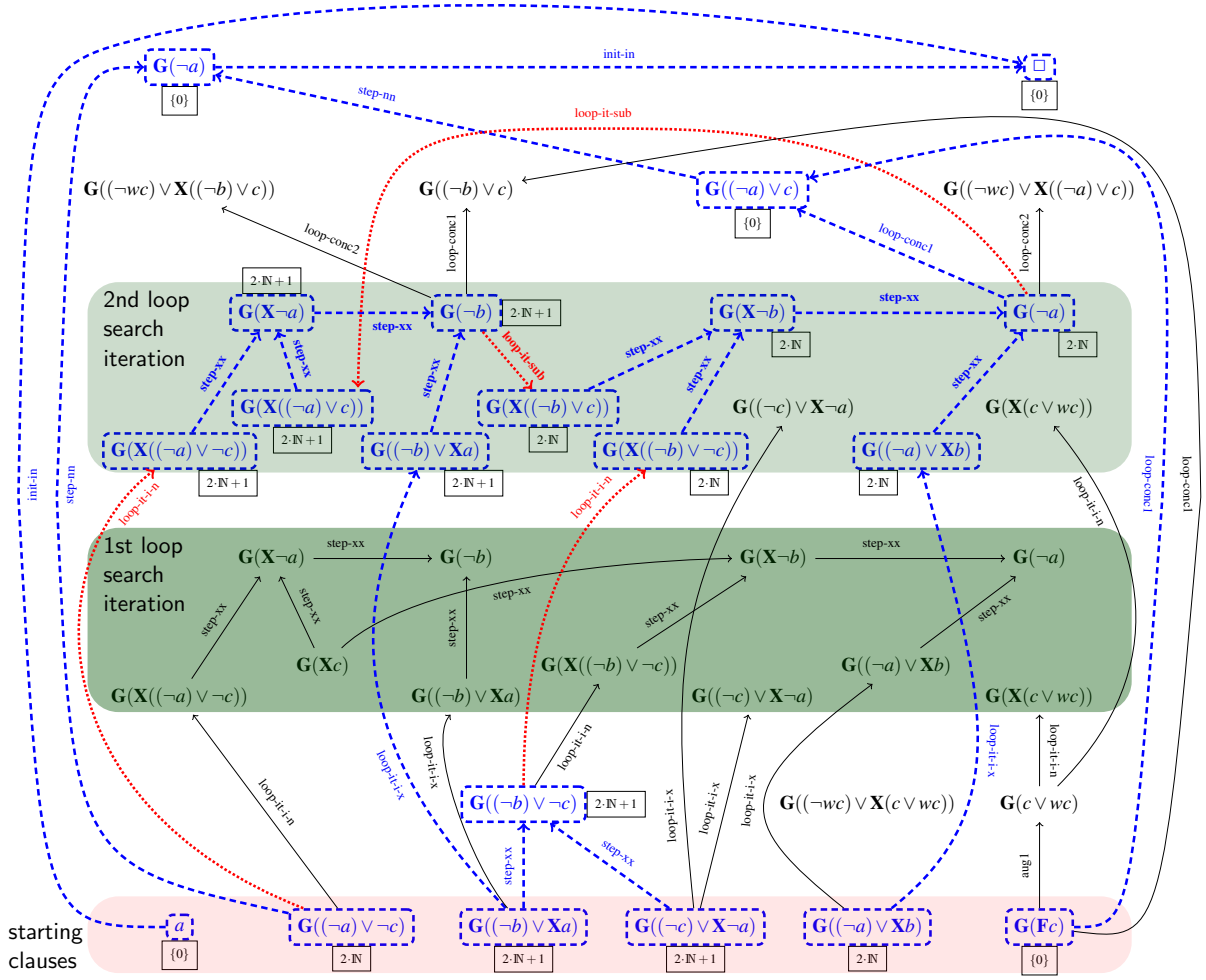


Figure 1: Example of an execution of the TR algorithm with corresponding resolution graph and UC extraction in SNF with sets of time points.

Remark 1 (Properties of LTLp)**.** *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., [Eme90]).*

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.

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

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,

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 $\mathbb{N}$.

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.

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

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.

UCs in SNF with Sets of Time Points 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 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 (Labeling Edges with Time Steps)**.** *$L_{E'}$ 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 (Labeling Vertices with Sets of Time Points)**.** *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 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 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 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 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 the 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 [Sch12b]). The construction in its proof will later be a fundamental step to actually compute the sets of time points.

Lemma 1 (Sets of Time Points for Vertices Labeled with Initial Clauses are $\{0\}$)**.** *Any vertex v in G' that is L_V-labeled with an initial clause is $L'_{V'}$-labeled with $\{0\}$.*

Lemma 2 (Labeling of Target Vertex is (Possibly Time-Shifted) Subset of Labeling of Source Vertex)**.** *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* step-nx*,* BFS-loop-it-init-n*,* BFS-loop-it-sub*,* BFS-loop-conclusion2*: time-shifted) subset of the labeling $L'_{V'}(v)$.*

Proposition 1 (Sets of Time Points are Semilinear Sets)**.** *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: (i) The set of states is the set of vertices of the graph G', V'. (ii) The set of transitions is the set of *reversed* edges of the graph G'. (iii) The labeling of the transitions is given by the $L_{E'}$-labeling of the corresponding edges. (iv) The (only) initial state is $v_\Box$. (v) 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 [Par66]. □

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 [Sch12a]) the proof of correctness in Thm. 3 (in App. B) 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 (SNF Clauses with Sets of Time Points)**.** *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 LTL*p *formula:*[3]

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

Definition 10 (UC in SNF with Sets of Time Points)**.** *Let $c_{1,1}, \ldots, c_{1,n_1}$ be the initial clauses in C^{uc}, $c_{2,1}$, …, 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*

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

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

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

Proposition 2 (Complexity of UC Extraction with Sets of Time Points)**.** *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., [HU79]). Finally, use an algorithm by Gawrychowski [Gaw11] 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.

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

UCs in LTL with Sets of Time Points 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) is similar to that of Thm. 2 (see [Sch12a]), 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 (Mapping a UC in SNF with Sets of Time Points to a UC in LTL with Sets of Time Points)**.** *Let ϕ be an unsatisfiable LTL formula, let $SNF(\phi)$ be its SNF, let ϕ^{uc} be the UC of ϕ in LTL, and let θ^{uc} be the UC of $SNF(\phi)$ 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 (Unsatisfiability of UC in LTL with Sets of Time Points)**.** *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 [Sch12b] (and, hence, according to Def. 4). 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 [Sch12a] (originally adapted from [Har05]) but extends it with sets of time points to show that understanding the presence of a problem

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

Figure 2: A lift specification.

becomes easier. The lift has two floors, indicated by f_0 and f_1. On each floor there is a button to call the 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 [Har05].

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.

$$\mathbf{X}\mathbf{G} b_1 \tag{10}$$

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

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 and for an example from the business process domain see App. C.

8 Experimental Evaluation

Our implementation, examples, and log files are available from [www].

category	family	source	# solved			|largest solved|
			UC w/o s.o.t.p.	UC w/ s.o.t.p. (Gawrychowski)	UC w/ s.o.t.p. (Sawa)	
application	**alaska_lift**	[Har05,Wul+08]	72	73	73	4605
	anzu_genbuf	[Blo+07]	16	16	16	1924
	forobots	[BDF09]	25	25	25	635
crafted	**schuppan_O1formula**	[SD11]	27	27	27	4006
	schuppan_O2formula	[SD11]	8	7	7	91
	schuppan_phltl	[SD11]	4	4	4	125
random	**rozier_formulas**	[RV10]	62	62	62	157
	trp	[HS02]	397	397	397	1421

Table 5: Overview of benchmark families.

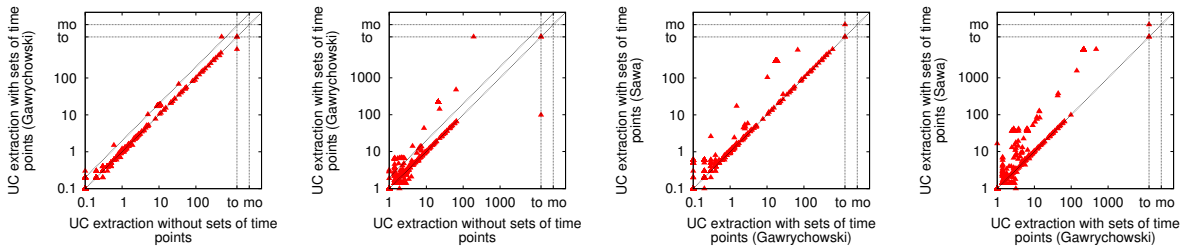
Implementation We use the version of TRP++ extended with extraction of UCs from [Sch12a] as the basis for our implementation. For data structures we used C++ Standard Library containers (e.g., [SL95, Jos12]), for graph operations the Boost Graph Library [bgl,SLL02].

Algorithms for Extracting Sets of Time Points 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., [HU79]). To compute Parikh images for unary NFAs we implemented an algorithm by Gawrychowski [Gaw11] and one by Sawa [Saw13]. 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.

Benchmarks Our examples are based on [SD11]. In categories **crafted** and **random** and in family **forobots** we considered all unsatisfiable instances from [SD11]. The version of **alaska_lift** used here contains a small bug fix: in [Wul+08,SD11] the subformula $\mathbf{X}u$ was erroneously written as literal Xu. Combining 2 variants 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.

Setup The experiments were performed on a laptop with Intel Core i7 M 620 processor at 2 GHz running Ubuntu 12.04. Run time and memory usage were measured with `run` [run]. The time and memory limits were 600 seconds and 6 GB.

Results In Fig. 3 (a) and (b) we show the overhead that is incurred by extracting UCs with 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.



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

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 Sawa's algorithm (y-axis) and Gawrychowski's algorithm (x-axis) for UC extraction with sets of time points. The off-center diagonal in (a) and (b) shows where $y = 2x$.

category	family	{0}	{0,1}	{0,1,2}	{0,2}	{1}	{1,2}	{1,2,3}	{1,2,3,4}	{1,3}	{1,4}	{2}	{2,3}	{2,3,4}	{3}	{3,4}	{4}	IN	IN+1	IN+2	IN+3	IN+4	IN+5	IN+6	IN+7, ..., IN+10	{4·IN+0}, ..., {4·IN+5}	{4·IN+1, 4·IN+2}	{4·IN+1, 4·IN+2, 4·IN+3}	{4·IN+2, 4·IN+3}	{4·IN+2, 4·IN+3, 4·IN+4}	{4·IN+3, 4·IN+4}	{5·IN+0}, ..., {5·IN+5}	{12·IN+0}, ..., {12·IN+12}
application	**alaska_lift**	✔	✔			✔	✔					✔						✔	✔														
	anzu_genbuf	✔				✔												✔	✔	✔	✔	✔	✔	✔		✔	✔	✔	✔	✔	✔		
	forobots	✔	✔			✔												✔	✔	✔	✔	✔	✔										
crafted	**schuppan_O1formula**	✔				✔																											
	schuppan_O2formula	✔																✔	✔														
	schuppan_phltl	✔																✔	✔	✔	✔	✔	✔	✔	✔								
random	**rozier_formulas**	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔										
	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.

Discussion 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 698 instances we considered with UC extraction without sets of time points, a UC was obtained for 611. With sets of time points enabled one instance more[4] and one instance less are solved. An analysis by category (for plots see App. D) shows that the run time (resp., memory) overhead for almost all instances of the **application** category is at most 50 % (resp., 100 %) for UC extraction with sets of time points using Gawrychowski's algorithm over UC extraction without sets of time points.

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, even though some of the subformulas involved are **G** subformulas. For the **schuppan_phltl** family (a temporal version of the pigeon hole problem (e.g., [Bie+09]); 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 [Gaw11] has better worst case complexity than Sawa's algorithm [Saw13]. 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

[4]For this instance the run time with sets of time points is just below the time limit.

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 [Sch12b]. 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 [HH11] and exploring whether computation of sets of time points is feasible for BDD-based algorithms via, e.g., [SB06,JSB06]. Other questions are how to apply the idea of sets of time points to unrealizable cores for LTL (e.g., [Sch12b]) or to branching time temporal logics. One could also investigate obtaining sets of time points by solving a system of constraints over sets of time points based on Lemmas 1, 2 rather than the approach based on Parikh images explored here. 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

[Arm+03] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. "Enhanced Vacuity Detection in Linear Temporal Logic". In: *CAV*. Ed. by W. Hunt Jr. and F. Somenzi. Vol. 2725. Lecture Notes in Computer Science. Springer, 2003, pp. 368–380. ISBN: 3-540-40524-0. DOI: 10.1007/978-3-540-45069-6_35 (cit. on pp. 1, 2).

[Awa+12] A. Awad, R. Goré, Z. Hou, J. Thomson, and M. Weidlich. "An iterative approach to synthesize business process templates from compliance rules". In: *Inf. Syst.* 37.8 (2012), pp. 714–736. DOI: 10.1016/j.is.2012.05.001 (cit. on pp. 2, 4).

[Bak+93] R. Bakker, F. Dikker, F. Tempelman, and P. Wognum. "Diagnosing and Solving Over-Determined Constraint Satisfaction Problems". In: *IJCAI*. 1993, pp. 276–281. URL: http://ijcai.org/Past%20Proceedings/IJCAI-93-VOL1/PDF/039.pdf (cit. on p. 1).

[BDF09] A. Behdenna, C. Dixon, and M. Fisher. "Deductive Verification of Simple Foraging Robotic Behaviours". In: *International Journal of Intelligent Computing and Cybernetics* 2.4 (2009), pp. 604–643. DOI: 10.1108/17563780911005818 (cit. on p. 16).

[Bee+01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". In: *Formal Methods in System Design* 18.2 (2001), pp. 141–163. DOI: 10.1023/A:1008779610539 (cit. on pp. 1, 2).

[Bee+09] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. "Explaining Counterexamples Using Causality". In: *CAV*. Ed. by A. Bouajjani and O. Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 94–108. ISBN: 978-3-642-02657-7. DOI: 10.1007/978-3-642-02658-4_11 (cit. on pp. 1–4).

[BG01] L. Bachmair and H. Ganzinger. "Resolution Theorem Proving". In: *Handbook of Automated Reasoning*. Ed. by J. Robinson and A. Voronkov. Elsevier and MIT Press, 2001, pp. 19–99. ISBN: 0-444-50813-9, 0-262-18223-8 (cit. on pp. 5, 9).

[bgl] URL: http://www.boost.org/doc/libs/release/libs/graph/ (cit. on p. 16).

[Bie+09] A. Biere, M. Heule, H. van Maaren, and T. Walsh, eds. *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009. ISBN: 978-1-58603-929-5 (cit. on p. 17).

[Bie+99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In: *TACAS*. Ed. by R. Cleaveland. Vol. 1579. Lecture Notes in Computer Science. Springer, 1999, pp. 193–207. ISBN: 3-540-65703-7. DOI: 10.1007/3-540-49059-0_14 (cit. on p. 4).

[Bie09] A. Biere. "Bounded Model Checking". In: [Bie+09], pp. 457–481. DOI: 10.3233/978-1-58603-929-5-457 (cit. on p. 4).

[Blo+07] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. "Specify, Compile, Run: Hardware from PSL". In: *COCV*. Ed. by S. Glesner, J. Knoop, and R. Drechsler. Vol. 190(4). ENTCS. Elsevier, 2007, pp. 3–16. DOI: 10.1016/j.entcs.2007.09.004 (cit. on pp. 1, 16, 32).

[Bur+92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. "Symbolic Model Checking: 10^{20} States and Beyond". In: *Inf. Comput.* 98.2 (1992), pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A (cit. on p. 4).

[CD91] J. Chinneck and E. Dravnieks. "Locating Minimal Infeasible Constraint Sets in Linear Programs". In: *ORSA Journal on Computing* 3.2 (1991), pp. 157–168. DOI: 10.1287/ijoc.3.2.157 (cit. on pp. 1, 4).

[CGP01] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001, pp. I–XIV, 1–314. ISBN: 978-0-262-03270-4 (cit. on p. 4).

[Chi+10] A. Chiappini, A. Cimatti, L. Macchi, O. Rebollo, M. Roveri, A. Susi, S. Tonetta, and B. Vittorini. "Formalization and validation of a subset of the European Train Control System". In: *ICSE (2)*. Ed. by J. Kramer, J. Bishop, P. Devanbu, and S. Uchitel. ACM, 2010, pp. 109–118. ISBN: 978-1-60558-719-6. DOI: 10.1145/1810295.1810312 (cit. on p. 1).

[Cim+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV 2: An OpenSource Tool for Symbolic Model Checking". In: *CAV*. Ed. by E. Brinksma and K. Larsen. Vol. 2404. Lecture Notes in Computer Science. Springer, 2002, pp. 359–364. ISBN: 3-540-43997-8. DOI: 10.1007/3-540-45657-0_29 (cit. on p. 3).

[Cim+07] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. "Boolean Abstraction for Temporal Logic Satisfiability". In: *CAV*. Ed. by W. Damm and H. Hermanns. Vol. 4590. Lecture Notes in Computer Science. Springer, 2007, pp. 532–546. ISBN: 3-540-22342-8. DOI: 10.1007/978-3-540-73368-3_53 (cit. on p. 3).

[Cla+03] E. Clarke, M. Talupur, H. Veith, and D. Wang. "SAT Based Predicate Abstraction for Hardware Verification". In: *SAT*. Ed. by E. Giunchiglia and A. Tacchella. Vol. 2919. Lecture Notes in Computer Science. Springer, 2003, pp. 78–92. ISBN: 3-540-20851-8. DOI: 10.1007/978-3-540-24605-3_7 (cit. on p. 3).

[Dbl] *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany*. IEEE Computer Society, 2003. ISBN: 0-7695-1870-2.

[Dix95] C. Dixon. "Strategies for Temporal Resolution". PhD thesis. Department of Computer Science, University of Manchester, 1995. URL: ftp://ftp.cs.man.ac.uk/pub/TR/UMCS-95-12-1.ps.Z (cit. on p. 4).

[Dix96] C. Dixon. "Search Strategies for Resolution in Temporal Logics". In: *CADE*. Ed. by M. McRobbie and J. Slaney. Vol. 1104. Lecture Notes in Computer Science. Springer, 1996, pp. 673–687. ISBN: 3-540-61511-3. DOI: 10.1007/3-540-61511-3_121 (cit. on p. 4).

[Dix97] C. Dixon. "Using Otter for Temporal Resolution". In: *ICTL*. Ed. by H. Barringer, M. Fisher, D. Gabbay, and G. Gough. Applied Logic Series. Kluwer, 1997, pp. 149–166. ISBN: 0-7923-6149-0 (cit. on p. 4).

[Dix98] C. Dixon. "Temporal Resolution Using a Breadth-First Search Algorithm". In: *Ann. Math. Artif. Intell.* 22.1-2 (1998), pp. 87–115. DOI: 10.1023/A:1018942108420 (cit. on p. 4).

[EF06] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006. DOI: 10.1007/978-0-387-36123-9 (cit. on p. 1).

[Eme90] E. Emerson. "Temporal and Modal Logic". In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. Ed. by J. van Leeuwen. Elsevier and MIT Press, 1990, pp. 995–1072. ISBN: 0-444-88074-7, 0-262-22039-3 (cit. on pp. 1, 4, 10, 23, 24).

[FDP01] M. Fisher, C. Dixon, and M. Peim. "Clausal temporal resolution". In: *ACM Trans. Comput. Log.* 2.1 (2001), pp. 12–56. DOI: 10.1145/371282.371311 (cit. on pp. 2–5).

[Fis+08] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. "A Framework for Inherent Vacuity". In: *Haifa Verification Conference*. Ed. by H. Chockler and A. Hu. Vol. 5394. Lecture Notes in Computer Science. Springer, 2008, pp. 7–22. ISBN: 978-3-642-01701-8. DOI: 10.1007/978-3-642-01702-5_7 (cit. on p. 1).

[Fis91] M. Fisher. "A Resolution Method for Temporal Logic". In: *IJCAI*. 1991, pp. 99–104. URL: http://ijcai.org/Past%20Proceedings/IJCAI-91-VOL1/PDF/017.pdf (cit. on pp. 2–5).

[FM09] J. Franco and J. Martin. "A History of Satisfiability". In: [Bie+09], pp. 3–74. DOI: 10.3233/978-1-58603-929-5-3 (cit. on p. 5).

[FN92] M. Fisher and P. Noël. *Transformation and Synthesis in* METATEM. *Part I: Propositional* METATEM. Tech. rep. UMCS-92-2-1. University of Manchester, Department of Computer Science, 1992. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.4998 (cit. on pp. 2, 5).

[Gaw11] P. Gawrychowski. "Chrobak Normal Form Revisited, with Applications". In: *CIAA*. Ed. by B. Bouchou-Markhoff, P. Caron, J. Champarnaud, and D. Maurel. Vol. 6807. Lecture Notes in Computer Science. Springer, 2011, pp. 142–153. ISBN: 978-3-642-22255-9. DOI: 10.1007/978-3-642-22256-6_14 (cit. on pp. 3, 14, 16, 17, 27, 28).

[GC04] A. Gurfinkel and M. Chechik. "How Vacuous Is Vacuous?" In: [JP04], pp. 451–466. DOI: 10.1007/978-3-540-24730-2_34 (cit. on p. 1).

[GN03] E. Goldberg and Y. Novikov. "Verification of Proofs of Unsatisfiability for CNF Formulas". In: [Dbl], pp. 10886–10891. DOI: 10.1109/DATE.2003.10008 (cit. on p. 1).

[Har05] A. Harding. "Symbolic Strategy Synthesis For Games With LTL Winning Conditions". PhD thesis. University of Birmingham, 2005 (cit. on pp. 14–16).

[Heu+95] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. "Propositional Logics on the Computer". In: *TABLEAUX*. Ed. by P. Baumgartner, R. Hähnle, and J. Posegga. Vol. 918. Lecture Notes in Computer Science. Springer, 1995, pp. 310–323. ISBN: 3-540-59338-1. DOI: 10.1007/3-540-59338-1_44 (cit. on p. 3).

[HH11] F. Hantry and M. Hacid. "Handling Conflicts in Depth-First Search for LTL Tableau to Debug Compliance Based Languages". In: *FLACOS*. Ed. by E. Pimentel and V. Valero. Vol. 68. EPTCS. 2011, pp. 39–53. DOI: 10.4204/EPTCS.68.5 (cit. on p. 18).

[HHT11] F. Hantry, M. Hacid, and R. Thion. "Detection of Conflicting Compliance Rules". In: *EDOCW*. IEEE Computer Society, 2011, pp. 419–428. ISBN: 978-1-4577-0869-5. DOI: 10.1109/EDOCW.2011.57 (cit. on p. 32).

[HK03] U. Hustadt and B. Konev. "TRP++ 2.0: A Temporal Resolution Prover". In: *CADE*. Ed. by F. Baader. Vol. 2741. Lecture Notes in Computer Science. Springer, 2003, pp. 274–278. ISBN: 3-540-40559-3. DOI: 10.1007/978-3-540-45085-6_21 (cit. on pp. 3–5, 9).

[HK04] U. Hustadt and B. Konev. "TRP++: A temporal resolution prover". In: *Collegium Logicum*. Ed. by M. Baaz, J. Makowsky, and A. Voronkov. Vol. 8. Kurt Gödel Society, 2004, pp. 65–79 (cit. on pp. 3–5).

[Hoo99] H. Hoos. *Heavy-Tailed Behaviour in Randomised Systematic Search Algorithms for SAT?* Tech. rep. TR-99-16. University of British Columbia, Department of Computer Science, 1999. URL: ftp://ftp.cs.ubc.ca/local/techreports/1999/TR-99-16.pdf (cit. on p. 1).

[HS02] U. Hustadt and R. A. Schmidt. "Scientific Benchmarking with Temporal Logic Decision Procedures". In: *KR*. Ed. by D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams. Morgan Kaufmann, 2002, pp. 533–546. ISBN: 1-55860-554-1 (cit. on p. 16).

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. ISBN: 0-201-02988-X (cit. on pp. 14, 16, 27).

[Jos12] N. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Second Edition. Addison-Wesley, 2012. ISBN: 978-0-321-62321-8 (cit. on p. 16).

[JP04] K. Jensen and A. Podelski, eds. *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*. Vol. 2988. Lecture Notes in Computer Science. Springer, 2004. ISBN: 3-540-21299-X.

[JSB06] T. Jussila, C. Sinz, and A. Biere. "Extended Resolution Proofs for Symbolic SAT Solving with Quantification". In: *SAT*. Ed. by A. Biere and C. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 54–60. ISBN: 3-540-37206-7. DOI: 10.1007/11814948_8 (cit. on p. 18).

[Kup06] O. Kupferman. "Sanity Checks in Formal Verification". In: *CONCUR*. Ed. by C. Baier and H. Hermanns. Vol. 4137. Lecture Notes in Computer Science. Springer, 2006, pp. 37–51. ISBN: 3-540-37376-4. DOI: 10.1007/11817949_3 (cit. on p. 1).

[KV03] O. Kupferman and M. Vardi. "Vacuity detection in temporal model checking". In: *STTT* 4.2 (2003), pp. 224–233. DOI: 10.1007/s100090100062 (cit. on pp. 1, 9).

[PA06] M. Pesic and W. van der Aalst. "A Declarative Approach for Flexible Business Processes Management". In: *Business Process Management Workshops*. Ed. by J. Eder and S. Dustdar. Vol. 4103. Lecture Notes in Computer Science. Springer, 2006, pp. 169–180. ISBN: 3-540-38444-8. DOI: 10.1007/11837862_18 (cit. on p. 1).

[Par66] R. Parikh. "On Context-Free Languages". In: *J. ACM* 13.4 (1966), pp. 570–581. DOI: 10.1145/321356.321364 (cit. on pp. 3, 4, 13).

[Pil+06] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. "Formal analysis of hardware requirements". In: *DAC*. Ed. by E. Sentovich. ACM, 2006, pp. 821–826. ISBN: 1-59593-381-6. DOI: 10.1145/1146909.1147119 (cit. on pp. 1, 2).

[pltl] URL: http://users.cecs.anu.edu.au/~rpg/PLTLProvers/ (cit. on p. 3).

[Pnu77] A. Pnueli. "The Temporal Logic of Programs". In: *FOCS*. IEEE, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32 (cit. on p. 1).

[Rob65] J. Robinson. "A Machine-Oriented Logic Based on the Resolution Principle". In: *J. ACM* 12.1 (1965), pp. 23–41. DOI: 10.1145/321250.321253 (cit. on p. 5).

[RS04] K. Ravi and F. Somenzi. "Minimal Assignments for Bounded Model Checking". In: [JP04], pp. 31–45. DOI: 10.1007/978-3-540-24730-2_3 (cit. on pp. 2–4).

[run] A. Biere and T. Jussila. *Benchmark Tool Run*. URL: http://fmv.jku.at/run/ (cit. on p. 16).

[RV10] K. Rozier and M. Vardi. "LTL satisfiability checking". In: *STTT* 12.2 (2010), pp. 123–137. DOI: 10.1007/s10009-010-0140-3 (cit. on p. 16).

[Sal73] A. Salomaa. *Formal Languages*. Academic Press, 1973 (cit. on pp. 3, 4).

[Saw13] Z. Sawa. "Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata". In: *Fundam. Inform.* 123.1 (2013), pp. 97–106. DOI: 10.3233/FI-2013-802 (cit. on pp. 3, 16, 17).

[SB06] C. Sinz and A. Biere. "Extended Resolution Proofs for Conjoining BDDs". In: *CSR*. Ed. by D. Grigoriev, J. Harrison, and E. Hirsch. Vol. 3967. Lecture Notes in Computer Science. Springer, 2006, pp. 600–611. ISBN: 3-540-34166-8. DOI: 10.1007/11753728_60 (cit. on p. 18).

[Sch12a] V. Schuppan. *Extracting Unsatisfiable Cores for LTL via Temporal Resolution*. Available at arXiv:1212.3884v1 [cs.LO]. 2012 (cit. on pp. 2–4, 7, 8, 13, 14, 16, 29).

[Sch12b] V. Schuppan. "Towards a notion of unsatisfiable and unrealizable cores for LTL". In: *Sci. Comput. Program.* 77.7-8 (2012), pp. 908–939. DOI: 10.1016/j.scico.2010.11.004 (cit. on pp. 1–4, 9, 12, 14, 18, 33).

[SD11] V. Schuppan and L. Darmawan. "Evaluating LTL Satisfiability Solvers". In: *ATVA*. Ed. by T. Bultan and P. Hsiung. Vol. 6996. Lecture Notes in Computer Science. Springer, 2011, pp. 397–413. ISBN: 978-3-642-24371-4. DOI: 10.1007/978-3-642-24372-1_28 (cit. on pp. 3, 16).

[Sim+10] J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. "Exploiting resolution proofs to speed up LTL vacuity detection for BMC". In: *STTT* 12.5 (2010), pp. 319–335. DOI: 10.1007/s10009-009-0134-1 (cit. on pp. 2, 4).

[SL95] A. Stepanov and M. Lee. *The Standard Template Library*. Tech. rep. 95-11(R.1). HP Laboratories, Nov. 1995. URL: http://www.stepanovpapers.com/STL/DOC.PDF (cit. on p. 16).

[SLL02] J. Siek, L. Lee, and A. Lumsdaine. *The Boost Graph Library - User Guide and Reference Manual*. C++ in-depth series. Pearson / Prentice Hall, 2002, pp. I–XXIV, 1–321. ISBN: 978-0-201-72914-6 (cit. on p. 16).

[Tar72] R. Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM J. Comput.* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010 (cit. on p. 28).

[trp++] URL: http://www.csc.liv.ac.uk/~konev/software/trp++/ (cit. on pp. 3–5).

[Wul+08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. "Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking". In: *TACAS*. Ed. by C. Ramakrishnan and J. Rehof. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 63–77. ISBN: 978-3-540-78799-0. DOI: 10.1007/978-3-540-78800-3_6 (cit. on p. 16).

[www] URL: http://www.schuppan.de/viktor/qapl13/ (cit. on pp. 4, 15).

[ZM03a] L. Zhang and S. Malik. *Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formula*. Presented at *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003*. 2003 (cit. on p. 4).

[ZM03b] L. Zhang and S. Malik. "Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications". In: [Dbl], pp. 10880–10885. DOI: 10.1109/DATE.2003.10014 (cit. on p. 1).

A Proofs: 5 LTL with Sets of Time Points (LTL*p*)

Let *LTLp2LTL* denote the function that takes an LTL*p* formula θ and returns an LTL formula ϕ by removing all sets of time points.

Lemma 3 (LTL*p* with all sets of time points $\mathbb{N}$ is LTL)**.** *Let θ be an LTL*p *formula such that all sets of time points are $\mathbb{N}$, and let $\phi \equiv LTL\text{p}2LTL(\theta)$. Then θ and ϕ are equivalent.*

Proof. By induction on θ. Base cases: Boolean constants and atomic propositions do not have sets of time points. Inductive cases: Each test for non-inclusion (i.e., $i \notin I$) in Tab. 4 is a left operand of a disjunction that, with $i \notin \mathbb{N}$ being 0, evaluates to its right operand. The latter evaluates to its LTL equivalent by inductive assumption. The case for inclusion is analogous. □

We take Lemma 3 as justification to use LTL operators (i.e., operators without sets of time points) to abbreviate LTL*p* operators with $\mathbb{N}$ as sets of time points.

Lemma 4 (An LTL*p* operator with sets of time points $\emptyset$ is equivalent to 1/0)**.** *Let θ be an LTL*p *formula with a positive (resp. negative) polarity subformula τ that is neither a Boolean constant nor an atomic proposition and with the sets of time points of the top level operator of τ being $\emptyset$. Then θ and θ such that τ is replaced with* 1 *(resp.* 0*) are equivalent.*

Proof. Directly from Def. 6: if τ has positive polarity, then the tests for non-inclusion (such as $i \notin I$) in Tab. 4 are left operands of disjunctions and evaluate to 1; the case for negative polarity is analogous. □

Lemma 5 (Enlarging sets of time points strengthens positive and weakens negative polarity subformulas)**.** *Let θ be an LTL*p *formula, let χ be such a modification of θ such that $LTL\text{p}2LTL(\theta) = LTL\text{p}2LTL(\chi)$ and all sets of time points in χ are (possibly non-strict) supersets of those in θ. Then $\chi \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \theta$.*

Proof. We show by induction on θ that for each subformula τ in θ with corresponding subformula σ in χ of positive (resp. negative) polarity $\sigma \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \tau$ (resp. $\tau \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \sigma$). Base cases: Boolean constants and atomic propositions do not have sets of time points. Inductive cases: For any LTL*p* operator except $\neg$ the operands τ' (, τ'') with associated sets of time points I' (, I'') have the same polarity as τ. The result follows by inductive assumption and increasing (resp. decreasing) monotonicity of Def. 6 in the operands and decreasing monotonicity in the sets of time points. For $\neg$ it is sufficient to note that it is monotonically decreasing (resp. increasing) in its operand, monotonically decreasing in its set of time points, and τ has opposite polarity of $\underset{I'}{\neg}\tau'$. □

Lemma 6. *LTL*p *with sets of time points restricted to semilinear sets is at most as expressive as QLTL (for QLTL see, e.g., [Eme90]).*

Proof. Let θ be an LTL*p* formula with sets of time points restricted to semilinear sets. I.e., each set of time points I occurring in θ can be written as $\bigcup_{1 \leq i \leq n} p_i \cdot \mathbb{N} + o_i$ for some $n \in \mathbb{N}, p_1, \ldots, p_n, o_1, \ldots, o_n \in \mathbb{N}$. For $m \in \mathbb{N}$ let $\mathbf{X}^m \psi$ abbreviate $\underbrace{\mathbf{X} \ldots \mathbf{X}}_{m} \psi$. Construct a QLTL formula as follows:

1. For each set of time points I introduce n fresh Boolean propositions q_{I_i} and a fresh Boolean proposition q_I.
2. Let $\circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$ and $\circ_2 \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}$. Replace each
   (a) positive polarity occurrence of $\underset{I}{\circ_1}\tau$ in θ with $\circ_1(q_I \rightarrow \tau)$,

(b) negative polarity occurrence of $\circ_1 \underset{I}{} \tau$ in θ with $\circ_1(q_I \wedge \tau)$,

(c) positive polarity occurrence of $\tau \circ_2 \underset{I,I'}{} \tau'$ in θ with $(q_I \rightarrow \tau) \circ_2 (q_{I'} \rightarrow \tau')$, and

(d) negative polarity occurrence of $\tau \circ_2 \underset{I,I'}{} \tau'$ in θ with $(q_I \wedge \tau) \circ_2 (q_{I'} \wedge \tau')$.

3. For each set of time points I replace θ with $\exists q_I \,.\, ((\mathbf{G}(q_I \leftrightarrow \bigvee_{1 \leq i \leq n} q_{I_i})) \wedge (\theta))$.

4. For each set of time points I, for each fresh proposition q_{I_i}, replace θ with

$$\exists q_{I_i} \,.\, ((\bigwedge_{0 \leq i' < o_i} \mathbf{X}^{i'} \neg q_{I_i}) \wedge (\mathbf{X}^{o_i}(q_{I_i} \wedge \mathbf{G}(q_{I_i} \rightarrow ((\bigwedge_{1 \leq i' < p_i} \mathbf{X}^{i'} \neg q_{I_i}) \wedge (\mathbf{X}^{p_i} q_{I_i})))))) \wedge (\theta))).$$

It's not hard to see that the resulting QLTL formula has the same set of satisfying assignments as θ. This concludes the proof. □

With Lemma 3–6 we have proved all parts of Remark 1.

Remark 1 (Properties of LTL*p*)**.** *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., [Eme90]).*

B Proofs: 6 UC Extraction with Sets of Time Points

Lemma 1 (Sets of Time Points for Vertices Labeled with Initial Clauses are $\{0\}$)**.** *Any vertex v in G' that is L_V-labeled with an initial clause is $L'_{V'}$-labeled with* $\{0\}$.

Proof. By Tab. 2 the only production rules that have initial clauses as premises are the initial resolution rules init-ii and init-in. Both rules have (as the only ones) an initial clause as conclusion. We denote with $v_\square$ the unique vertex in the main partition L_V-labeled with the empty clause □. Hence, either G' contains no vertex L_V-labeled with an initial clause, in which case the claim is vacuously true. Otherwise, the empty clause that L_V-labels $v_\square$ is an initial clause and $v_\square$ and all vertices L_V-labeled with initial clauses are connected via edges $L_{E'}$-labeled 0. The claim now follows with Def. 8 by induction on the distance of a vertex L_V-labeled with an initial clause from $v_\square$. □

Lemma 2 (Labeling of Target Vertex is (Possibly Time-Shifted) Subset of Labeling of Source Vertex)**.** *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 step-nx, BFS-loop-it-init-n, BFS-loop-it-sub, BFS-loop-conclusion2: time-shifted) subset of the labeling $L'_{V'}(v)$.*

Proof. Directly by Def. 8. □

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

Proof. In this proof we focus on reasoning why the production rules of TR continue to lead to correct inferences or to propagate information correctly in the presence of sets of time points. We assume that the clauses in the L_V-labeling of G' are assigned the sets of time points in the $L'_{V'}$-labeling of G' according to Def. 9. I.e., for any vertex v in the subgraph G', if $c = L_V(v)$ and $I = L'_{V'}(v)$, then we identify c and $\underset{I}{c}$.

We proceed as follows. We first show that inferences based on initial and step resolution rules [init-ii], [init-in], [step-nn], [step-nx], and [step-xx] also hold when taking sets of time points into account. Next we show that productions [BFS-loop-it-init-x], [BFS-loop-it-init-n], [BFS-loop-it-init-c], and [BFS-loop-it-sub] required as part of BFS loop search also apply with sets of time points, and then we show what a BFS loop search with sets of time points actually proves. Finally, we use the latter to show validity of the remaining rules [aug1], [aug2], [BFS-loop-conclusion1], and [BFS-loop-conclusion2].

For initial and step resolution notice, that these inference rules can be seen as essentially conjunctions of propositional inferences at a single (initial resolution) or all (step resolution) time points. For example, step resolution [step-nn] between $(\mathbf{G}(p_1 \vee \ldots \vee p_n))$ and $(\mathbf{G}(q_1 \vee \ldots \vee q_{n'}))$ can be seen as propositional resolution between $(p_1 \vee \ldots \vee p_n)$ and $(q_1 \vee \ldots \vee q_{n'})$ applied at all time points $i \in \mathbb{N}$. Hence, for [init-ii] and [init-in], as long as time point 0 is contained in the sets of time points of the premises, the conclusion can be inferred at time point 0. Similarly, for [step-nn] and [step-xx], if I is the set of time points of premise 1 and I' is the set of time points of premise 2, then the conclusion holds at time points $I \cap I'$. Taking time-shift into account, for [step-nx], if I is the set of time points of premise 1 and I' is the set of time points of premise 2, then the conclusion holds at time points $((I-1) \cap \mathbb{N}) \cap I'$. Using Lemmas 1 and 2, we can conclude that all instances of initial and step resolution in G' are correct inferences also when taking sets of time points into account.

The role of production rules [BFS-loop-it-init-x] and [BFS-loop-it-init-n] is propagation of information from the main partition to the loop partition. By Lemma 2 it is clear that information is propagated correctly also with sets of time points. Production rule [BFS-loop-it-init-c] does not lead to the creation of edges in G, hence, there is nothing to prove. Production rule [BFS-loop-it-sub] continues to record a correct relation between two clauses by a similar argument as for the rules used in saturation.

Let $c = (\mathbf{G}(p_1 \vee \ldots \vee p_n))$ be a clause contained in C' (see line 13 in Alg. 1) of a successful BFS loop search iteration. Let v_c be the corresponding vertex in the subgraph G' and let I be the $L'_{V'}$-labeling of v_c. In order to understand what a BFS loop search iteration actually proves in the context of sets of time points we now trace c, starting at a single time point $i \in I$, backwards through the corresponding partition L of the subgraph G'. We inductively define the following sets of clauses:

1. Let V'_i be the singleton set containing v_c: $V'_i = \{v_c\}$. Let C'_i be the singleton set containing c with set of time points $\{i\}$ assigned: $C'_i = \{ \underset{\{i\}}{c} \} = \{(\underset{\{i\}}{\mathbf{G}} (p_1 \underset{\{i\},\{i\}}{\vee} \ldots \underset{\{i\},\{i\}}{\vee} p_n))\}$.

2. For all $i \leq i'$ we define $C''_{i'}$ as follows. Let $V''_{i'}$ be the set of vertices in partition L that are backward reachable from some vertex in $V'_{i'}$ via edges generated by saturation restricted to rule [step-xx] and that are L_V-labeled with a clause generated by [BFS-loop-it-init-c]. Then $C''_{i'}$ is the set of all clauses L_V-labeling some vertex in $V''_{i'}$ with set of time points $\{i'\}$ assigned: $C''_{i'} = \{ \underset{\{i'\}}{c''} \mid \exists v'' \in V''_{i'} \,.\, c'' = L_V(v'')\}$.

3. For all $i < i'$ we define $C'_{i'}$ as follows. Let $V'_{i'}$ be the set of vertices in partition L that are backward reachable from vertices in $V''_{i'-1}$ via edges generated by [BFS-loop-it-sub]. Then $C'_{i'}$ is the set of all clauses L_V-labeling some vertex in $V'_{i'}$ with set of time points $\{i'\}$ assigned: $C'_{i'} = \{ \underset{\{i'\}}{c'} \mid \exists v' \in V'_{i'} \,.\, c' = L_V(v')\}$.

Intuitively, for a given i', $C''_{i'}$ represents a subset of the clauses that are required (in addition to the UC in SNF C^{uc}) to prove the clauses in $C'_{i'}$ at time point i' by saturation restricted to rule step-xx. In turn, $C'_{i'+1}$ is needed to establish $C''_{i'}$ by subsumption BFS-loop-it-sub. Note that, disregarding sets of time points, $C''_{i'}$ is bounded from above by the set of clauses in partition L that are generated by BFS-loop-it-init-c and $C'_{i'}$ is bounded from above by C' (line 13 in Alg. 1). Hence, disregarding sets of time points, the sequences $C''_{i'}$ and $C'_{i'}$ eventually will become cyclic.

Using the definition of $C''_{i'}$ and $C'_{i'}$ in 1.–3. above as well as the correctness of step-xx in the presence of sets of time points as argued above, we can infer that, assuming the UC θ^{uc}, it is provable that the conjunction of the clauses in $C''_{i'}$ at any time point $i' \geq i$ implies the conjunction of the clauses in $C'_{i'}$ at that time point. I.e., assuming θ^{uc}, it is provable that

$$\forall i \leq i' \,.\, (\bigwedge\nolimits_{c'' \in C''_{i'}} c'') \rightarrow (\bigwedge\nolimits_{c' \in C'_{i'}} c'). \tag{12}$$

Again using the definition of $C''_{i'}$ and $C'_{i'}$ in 1.–3. above as well as the correctness of BFS-loop-it-sub in the presence of sets of time points sets as argued above, we can infer that the conjunction of the clauses in $C'_{i'}$ at any time point $i' > i$ implies the conjunction of the clauses in $C''_{i'-1}$ at time point $i'-1$. I.e.,

$$\forall i < i' \,.\, (\bigwedge\nolimits_{c' \in C'_{i'}} c') \rightarrow (\bigwedge\nolimits_{c'' \in C''_{i'-1}} c''). \tag{13}$$

Notice that for all time points $i' \geq i$ any element of the set $C''_{i'}$ is of the form

$$(\underset{\{i'\}}{\mathbf{G}}\,((0) \underset{\{i'\},\{i'\}}{\vee} (\underset{\{i'+1\}}{\mathbf{X}}\,(q_1 \underset{\{i'+1\},\{i'+1\}}{\vee} \cdots \underset{\{i'+1\},\{i'+1\}}{\vee} q_n \underset{\{i'+1\},\{i'+1\}}{\vee} l)))).$$

I.e., we have

$$\forall i \leq i' \,.\, \forall c'' \in C''_{i'} \,.\, (\mathbf{X} \underset{\{i'+1\}}{\mathbf{G}}\, l) \rightarrow c''. \tag{14}$$

Finally, remember that $c = (\mathbf{G}(p_1 \vee \ldots \vee p_n))$ is a clause contained in C' of a successful BFS loop search iteration and that I is the $L'_{V'}$-labeling of the corresponding vertex v_c with $i \in I$. Taking (12) – (14) with some rewriting we obtain (15) to tell us what a successful BFS loop search with sets of time points actually proves:

$$(\underset{\{i\}}{\mathbf{G}}\,((p_1 \underset{\{i\},\{i\}}{\vee} \cdots \underset{\{i\},\{i\}}{\vee} p_n) \underset{\{i\},\{i\}}{\vee} (\underset{\{i+1\}}{\mathbf{X}}\, \underset{[i+1,\infty)}{\mathbf{G}}\, \underset{[i+1,\infty)}{\neg}\, l))). \tag{15}$$

For BFS-loop-conclusion1 assume $(\underset{I}{\mathbf{G}}((q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'}) \underset{I,I}{\vee} (\underset{[min(I),\infty)}{\mathbf{F}}\,(l))))$ and $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}}\, \underset{[min(I')+1,\infty)}{\mathbf{G}}\, \underset{[min(I')+1,\infty)}{\neg}\, l)))$. Let $i \in I \cap I'$. Now it's easy to see that if neither $q_1 \vee \ldots \vee q_{n'}$ nor $p_1 \vee \ldots \vee p_n$ hold at time point i, then l must hold. Hence, we have

$$(\underset{I\cap I'}{\mathbf{G}}\,((q_1 \underset{I\cap I',I\cap I'}{\vee} \cdots \underset{I\cap I',I\cap I'}{\vee} q_{n'}) \underset{I\cap I',I\cap I'}{\vee} (p_1 \underset{I\cap I',I\cap I'}{\vee} \cdots \underset{I\cap I',I\cap I'}{\vee} p_n) \underset{I\cap I',I\cap I'}{\vee} l)).$$

By Lemma 2 the set of time points $L'_{V'}$-labeling the target vertex is a subset of the sets of time points $L'_{V'}$-labeling the source vertices. As $I \cap I'$ is the largest set that is a subset of both I and I', these productions remain correct with sets of time points.

For aug1, aug2, and BFS-loop-conclusion2 assume $(\underset{I}{\mathbf{G}}((q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'}) \underset{I,I}{\vee} (\underset{[min(I),\infty)}{\mathbf{F}}\,(l))))$ and $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}}\, \underset{[min(I')+1,\infty)}{\mathbf{G}}\, \underset{[min(I')+1,\infty)}{\neg}\, l)))$. Let wl be fresh. Now it's easy to see that we can add the following clauses without affecting satisfiability of θ^{uc}: $(\underset{I}{\mathbf{G}}(q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'} \underset{I,I}{\vee} l \underset{I,I}{\vee} wl))$,

$(\underset{\mathbb{N}}{\mathbf{G}}((\underset{\mathbb{N}}{\neg} wl) \underset{\mathbb{N},\mathbb{N}}{\vee} (\underset{\mathbb{N}}{\mathbf{X}}(l \underset{\mathbb{N},\mathbb{N}}{\vee} wl))))$, and $(\underset{I'-1}{\mathbf{G}}((\underset{I'-1}{\neg} wl) \underset{I'-1,I'-1}{\vee} (\underset{I'}{\mathbf{X}}(p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n \underset{I',I'}{\vee} l))))$. By Lemma 2, for aug1 and BFS-loop-conclusion2 the set of time points $L'_{V'}$-labeling the target vertex is a subset of the sets of time points $L'_{V'}$-labeling the source vertices. For aug2 any set of time points is a subset of $\mathbb{N}$. Hence, these productions remain correct with sets of time points. This concludes the proof.

□

Proposition 2 (Complexity of UC Extraction with Sets of Time Points)**.** *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. We assume that for each vertex in the graph there is a list of incoming and outgoing edges. Sets are represented as arrays of bits of predetermined, fixed size with 1 bit for each potential set element. With n potential set elements that incurs cost $\mathcal{O}(1)$ for element addition, removal, and membership test as well as $\mathcal{O}(n)$ for set creation and reset to $\emptyset$. A list of length n incurs cost $\mathcal{O}(1)$ for creation, element addition, and emptiness check as well as $\mathcal{O}(n)$ for iterating over all of its elements. The vertex $v_\Box$ in the main partition is stored in a designated variable; the vertices L_V-labeled with clauses in C^{uc} in the main partition are stored in a list.

We proceed as follows. (i) As preparation we reverse all edges in the subgraph G'. (ii) We turn the subgraph G' into a unary NFA[5] by treating all edges[6] $L_{E'}$-labeled with 0 as ε-transitions and applying a standard method for elimination of ε-transitions in NFA [HU79]. That leaves us with a NFA with only 1-$L_{E'}$-labeled transitions, i.e., a unary NFA. (iii) We initialize the sets of time points. (iv) We use an algorithm by Gawrychowski [Gaw11] extended to handle all final vertices in parallel to compute Parikh images.

Preparation Reversing all edges in the subgraph G' can be performed in time $\mathcal{O}(|V'| + |E'|)$.

Turning G' into a Unary NFA (i) Designate $v_\Box$ as the initial vertex: $\mathcal{O}(1)$. (ii) For each vertex compute the set of vertices reachable from that vertex via a sequence of 0-$L_{E'}$-labeled edges. This can be done, e.g., by using DFS from each vertex: $\mathcal{O}(|V'| \cdot (|V'| + |E'|))$. (iii) For each vertex compute the set of vertices reachable from that vertex via a 1-$L'_{V'}$-labeled edge followed by a sequence of 0-$L_{E'}$-labeled edges: $\mathcal{O}(|V'|^2 + |V'| \cdot |E'|)$. (iv) For each vertex compute the set of vertices reachable from that vertex via a sequence of 0-$L_{E'}$-labeled edges, followed by a 1-$L'_{V'}$-labeled edge, and followed by a sequence of 0-$L_{E'}$-labeled edges: $\mathcal{O}(|V'|^3)$. (v) Replace the set of edges E' with the edges such that there is one 1-$L'_{V'}$-labeled edge for each pair of vertices v, v' where v' is reachable from v via a sequence of edges as in the previous step. Call the new set of edges E'': $\mathcal{O}(|V'| + |E'| + |E''|)$. The overall cost for turning G' into a unary NFA is, therefore, $\mathcal{O}(|V'|^3 + |V'| \cdot (|V'| + |E'|) + |E''|)$.

Initializing Sets of Time Points Initialize all sets of time points with $\emptyset$ and then add 0 to those of clauses in C^{uc} that are labeling vertices in the main partition reachable via a sequence of 0-$L_{E'}$-edges from $v_\Box$. The required information is available from the conversion to a unary NFA. This can be done in time $\mathcal{O}(|C|)$.

[5] A *unary* NFA is a NFA over a unary alphabet.

[6] We use the terms "vertex" and "state" as well as "edge" and "transition" interchangeably.

Computing Parikh Images Extending Gawrychowski's algorithm [Gaw11] to handle multiple final vertices in parallel is straightforward. Essentially, when the original algorithm checks whether a single final vertex has been reached, the extended version carries out that check for each final vertex. To illustrate how the Parikh images can be computed in time $\mathcal{O}(|V'|^3 + |C| \cdot |V'|^2)$ we show the main part of the algorithm in Alg. 2–4. For an explanation of how it works see [Gaw11].

The first part (Alg. 2) is mostly preparation. It partitions the unary NFA into SCCs and computes the length of the shortest loop in each SCC. This can be done, e.g., using Tarjan's algorithm [Tar72] and then using BFS from each vertex. The overall cost for preparation is $\mathcal{O}(|V'| \cdot (|V'| + |E''|))$. The second part (Alg. 3) processes non-trivial SCCs. It can be carried out in time $\mathcal{O}(|V'|^3 + |C| \cdot |V'|^2)$. It is important to note that the sum of the sizes of all SCCs is bounded by the number of vertices V'. Moreover, each vertex can appear in each of the $2d$ frontiers only once per SCC. The third part (Alg. 4) processes trivial SCCs. It can be carried out in time $\mathcal{O}(|E''| + |C| \cdot |V'| + |V'|^2)$. Hence, the overall time to compute Parikh images is $\mathcal{O}(|V'|^3 + |C| \cdot |V'|^2)$.

Summing Up The time taken for all steps is bounded by $\mathcal{O}(|V'|^3 + |C| \cdot |V'|^2)$. This concludes the proof.

□

Theorem 4 (Unsatisfiability of UC in LTL with Sets of Time Points)**.** *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.*

Proof. Let $SNF(\phi)$ be the SNF of ϕ, let C^{uc} be the UC of $SNF(\phi)$ in SNF, and let θ^{uc} be the UC of $SNF(\phi)$ in SNF with sets of time points.

Translate the UC in LTL with set of time points, θ'^{uc}, into an SNF formula θ' with sets of time points as follows. First, obtain a formula in SNF (without sets of time points) by translating θ'^{uc} according to Def. 3 (disregarding sets of time points in θ'^{uc}). Then assign $\{0\}$ as a set of time points to all outer conjunctions. Finally, assign each clause a set of time points (see Def. 9) as detailed next.

Assign the set of time points $\{0\}$ to $x_{\theta'^{uc}}$. For each occurrence of a subformula ψ in θ'^{uc}, denote the set of time points assigned to ψ in θ'^{uc} with I_ψ. Let $x_\psi, x_{\psi'}, \ldots$ be the occurrences of propositions in a clause c that are marked blue boxed in Tab. 3 and that refer to occurrences of subformulas $\psi, \psi', \ldots$ in θ'^{uc}.

- If the clause c comes from translating a $\neg$, $\wedge$, or $\vee$ subformula, then assign c the set of time points $I_\psi \cup I_{\psi'} \cup \ldots$.
- If the clause c comes from translating a $\mathbf{X}$ subformula, then assign c the set of time points $I_\psi - 1$.
- If the clause c comes from a positive polarity occurrence of a $\mathbf{F}$ subformula or a negative polarity occurrence of a $\mathbf{G}$ subformula, then assign c the set of time points I_ψ.
- If the clause c comes from a positive polarity occurrence of a $\mathbf{G}$ subformula or a negative polarity occurrence of a $\mathbf{F}$ subformula, then assign a clause $c = (\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow (\mathbf{X}x_{\mathbf{G}\psi})))$ or $c = (\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\mathbf{X}\neg x_{\mathbf{F}\psi})))$ the set of time points $\mathbb{N}$ and a clause $c = (\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow x_\psi))$ or $c = (\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow (\neg x_\psi)))$ the set of time points I_ψ.
- If the clause c comes from a positive polarity occurrence of a $\mathbf{U}$ subformula or a negative polarity occurrence of a $\mathbf{R}$ subformula, then assign a clause $c = (\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (x_{\psi'} \vee x_\psi)))$ or $c = (\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg x_{\psi'}) \vee (\neg x_\psi))))$ the set of time points I_ψ, a clause $c = (\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (x_{\psi'} \vee (\mathbf{X}x_{\psi\mathbf{U}\psi'}))))$ or $c = (\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg x_{\psi'}) \vee (\mathbf{X}\neg x_{\psi\mathbf{R}\psi'}))))$ the set of time

points $I_{\psi'}$, and a clause $c = (\mathbf{G}(x_{\psi \mathbf{U} \psi'} \rightarrow (\mathbf{F} x_{\psi'})))$ or $c = (\mathbf{G}((\neg x_{\psi \mathbf{R} \psi'}) \rightarrow (\mathbf{F} \neg x_{\psi'})))$ the set of time points $I_{\psi'}$.

- If the clause c comes from a negative polarity occurrence of a $\mathbf{U}$ subformula or a positive polarity occurrence of a $\mathbf{R}$ subformula, then assign a clause $c = (\mathbf{G}((\neg x_{\psi \mathbf{U} \psi'}) \rightarrow (\neg x_{\psi'})))$ or $c = (\mathbf{G}(x_{\psi \mathbf{R} \psi'} \rightarrow x_{\psi'}))$ the set of time points $I_{\psi'}$ and a clause $c = (\mathbf{G}((\neg x_{\psi \mathbf{U} \psi'}) \rightarrow ((\neg x_{\psi}) \vee (\mathbf{X} \neg x_{\psi \mathbf{U} \psi'}))))$ or $c = (\mathbf{G}(x_{\psi \mathbf{R} \psi'} \rightarrow (x_{\psi} \vee (\mathbf{X} x_{\psi \mathbf{R} \psi'}))))$ the set of time points I_{ψ}.

Note that in the first three cases as well as in the last case this is an exact reversal of Def. 11, while in the remaining cases this is a potentially inexact reversal of Def. 11 as at least one set of time points may be larger than that of the corresponding clause in θ^{uc}.

As in the proof of Thm. 2 (see [Sch12a]) θ' contains a superset of the clauses of θ^{uc} if sets of time points are disregarded. Moreover, by construction, the sets of time points in θ' are supersets of the sets of time points in θ^{uc}. Hence, provided the translation from θ'^{uc} into θ' is satisfiability preserving, with Lemma 5, we have that θ'^{uc} is unsatisfiable.

It is now left to show that the translation from θ'^{uc} into θ' preserves satisfiability. Assume a satisfying assignment π for θ'^{uc}. Extend π to a satisfying assignment π' for θ' as follows: For each occurrence of a subformula τ in θ'^{uc} that is not a Boolean constant or an atomic proposition introduce a fresh proposition x_{τ} and assign it the truth values of τ in θ'^{uc} on the satisfying assignment according to Def. 6. It is easy to see that $(\pi', 0)$ fulfills $x_{\theta^{uc}}$. The fact that π' is a satisfying assignment for the remaining clauses of θ' follows from the semantics of LTLp. □

Algorithm 2: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. Preparation.

```
1  Partition 𝒜 into SCCs;
2  foreach SCC A do
3      Find the length of the shortest loop in A;
4  Create an empty set of vertices sforbidden;
5  Create a list of vertices lfinal and assign it the list of vertices L_V-labeled with clauses in C^uc;
```

Algorithm 3: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. Processing non-trivial SCCs.

```
1  foreach SCC A do
2      d ← length of the shortest loop in A;
3      Create a set of vertices scurrentscc and a list of vertices lcurrentscc;
4      scurrentscc ← vertices of A; lcurrentscc ← vertices of A;
5      foreach 0 ≤ i < d do
6          Create empty sets of vertices simage_{0,i}, sreached_{0,i}, sfrontier_{0,i};
7          Create empty sets of vertices simage_{1,i}, sreached_{1,i}, sfrontier_{1,i};
8          Create empty lists of vertices limage_{0,i}, lfrontier_{0,i}, limage_{1,i}, lfrontier_{1,i};
9          Create an empty set of vertices sfinalseen_i;
10     sreached_{0,0} ← sfrontier_{0,0} ← {v□};
11     lfrontier_{0,0} ← [v□];
12     i ← 0;
13     while (lfrontier_{0,i mod d} ≠ []) ∨ (lfrontier_{1,i mod d} ≠ []) do
14         i_m ← i mod d;
15         i_{m,next} ← (i+1) mod d;
16         simage_{0,i_{m,next}} ← sfrontier_{0,i_{m,next}} ← simage_{1,i_{m,next}} ← sfrontier_{1,i_{m,next}} ← ∅;
17         limage_{0,i_{m,next}} ← lfrontier_{0,i_{m,next}} ← limage_{1,i_{m,next}} ← lfrontier_{1,i_{m,next}} ← [];
18         foreach v ∈ lfrontier_{0,i_m} do
19             foreach target vertex v' of each outgoing edge of v do
20                 if v' ∈ sforbidden then continue;
21                 if v' ∈ scurrentscc then
22                     if v' ∉ simage_{1,i_{m,next}} then
23                         simage_{1,i_{m,next}} ← simage_{1,i_{m,next}} ∪ {v'};
24                         limage_{1,i_{m,next}} ← limage_{1,i_{m,next}} ∘ [v'];
25                 else
26                     if v' ∉ simage_{0,i_{m,next}} then
27                         simage_{0,i_{m,next}} ← simage_{0,i_{m,next}} ∪ {v'};
28                         limage_{0,i_{m,next}} ← limage_{0,i_{m,next}} ∘ [v'];
29         foreach v ∈ limage_{0,i_{m,next}} do
30             if v ∉ sreached_{0,i_{m,next}} then
31                 sreached_{0,i_{m,next}} ← sreached_{0,i_{m,next}} ∪ {v};
32                 sfrontier_{0,i_{m,next}} ← sfrontier_{0,i_{m,next}} ∪ {v};
33                 lfrontier_{0,i_{m,next}} ← lfrontier_{0,i_{m,next}} ∘ [v];
34         foreach v ∈ lfrontier_{1,i_m} do
35             foreach target vertex v' of each outgoing edge of v do
36                 if v' ∈ sforbidden then continue;
37                 if v' ∉ simage_{1,i_{m,next}} then
38                     simage_{1,i_{m,next}} ← simage_{1,i_{m,next}} ∪ {v'};
39                     limage_{1,i_{m,next}} ← limage_{1,i_{m,next}} ∘ [v'];
40         foreach v ∈ limage_{1,i_{m,next}} do
41             if v ∉ sreached_{1,i_{m,next}} then
42                 sreached_{1,i_{m,next}} ← sreached_{1,i_{m,next}} ∪ {v};
43                 sfrontier_{1,i_{m,next}} ← sfrontier_{1,i_{m,next}} ∪ {v};
44                 lfrontier_{1,i_{m,next}} ← lfrontier_{1,i_{m,next}} ∘ [v];
45         foreach v ∈ lfinal do
46             if (v ∉ sfinalseen_{i_{m,next}}) ∧ (v ∈ sfrontier_{1,i_{m,next}}) then
47                 Add d·ℕ + i + 1 to the Parikh image of L_V(v);
48                 sfinalseen_{i_{m,next}} ← sfinalseen_{i_{m,next}} ∪ {v};
49         i ← i + 1;
50     foreach v ∈ lcurrentscc do
51         sforbidden ← sforbidden ∪ {v};
```

Algorithm 4: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. Processing trivial SCCs.

```
1  Create a set of vertices scurr; scurr ← {v□};
2  Create an empty set of vertices snext;
3  Create a list of vertices lcurr; lcurr ← [v□];
4  Create an empty list of vertices lnext;
5  i ← 0;
6  while lcurr ≠ [] do
7      foreach v ∈ lcurr do
8          foreach target vertex v' of each outgoing edge of v do
9              if v' ∈ sforbidden then continue;
10             if v' ∉ snext then
11                 snext ← snext ∪ {v'};
12                 lnext ← lnext ∘ [v'];
13     foreach v ∈ lfinal do
14         if v ∈ snext then
15             Add 0·ℕ + i + 1 to the Parikh image of L_V(v);
16     scurr ← snext; snext ← ∅;
17     lcurr ← lnext; lnext ← [];
```

C Additional Examples

C.1 An Example of Disjuncts of an Invariant Holding at Different Time Points

This example (16) is based on a subset of the specification of a buffer [Blo+07]. The subset regulates the communication between a sender and the buffer, where the sender can issue requests to the buffer ($StoB\_REQ_0$) and the buffer can acknowledge requests to the sender ($BtoS\_ACK_0$). For details see [Blo+07].

$$
\begin{aligned}
&(\neg StoB\_REQ_0) \wedge (\neg BtoS\_ACK_0) && \text{(16a)}\\
\wedge\ &(\mathbf{G}((StoB\_REQ_0 \wedge \neg BtoS\_ACK_0) \rightarrow \mathbf{X} StoB\_REQ_0)) && \text{(16b)}\\
\wedge\ &(\mathbf{G}(BtoS\_ACK_0 \rightarrow \mathbf{X}\neg StoB\_REQ_0)) && \text{(16c)}\\
\wedge\ &(\mathbf{GF}((StoB\_REQ_0 \wedge BtoS\_ACK_0) \vee ((\neg StoB\_REQ_0) \wedge (\neg BtoS\_ACK_0)))) && \text{(16d)}\\
\wedge\ &(\mathbf{G}(((\neg StoB\_REQ_0) \wedge \mathbf{X} StoB\_REQ_0) \rightarrow \mathbf{X}\neg BtoS\_ACK_0)) && \text{(16e)}\\
\wedge\ &(\mathbf{G}(((\neg BtoS\_ACK_0) \wedge \mathbf{X} BtoS\_ACK_0) \rightarrow StoB\_REQ_0)) && \text{(16f)}\\
\wedge\ &(\mathbf{G}((BtoS\_ACK_0 \wedge StoB\_REQ_0) \rightarrow \mathbf{X} BtoS\_ACK_0)) && \text{(16g)}
\end{aligned}
$$

The engineer considers a scenario where a request is sent every fourth time point (17a), (17b). In this scenario she would like to verify an invariant (17c)–(17f). Not only does the invariant turn out to hold, but the excerpt of the UC (18a)–(18d) shows that each disjunct of the invariant holds every fourth time point.

$$
\begin{aligned}
&(\mathbf{X} tick) \wedge (\mathbf{G}(tick \rightarrow \mathbf{XXXX} tick)) && \text{(17a)}\\
\wedge\ &(\mathbf{G}((tick \rightarrow StoB\_REQ_0) \wedge ((\mathbf{X} tick) \rightarrow (\neg StoB\_REQ_0)))) && \text{(17b)}\\
\wedge\ &(\neg(\mathbf{G}((StoB\_REQ_0 \wedge \neg BtoS\_ACK_0) && \text{(17c)}\\
&\qquad \vee (StoB\_REQ_0 \wedge BtoS\_ACK_0) && \text{(17d)}\\
&\qquad \vee ((\neg StoB\_REQ_0) \wedge BtoS\_ACK_0) && \text{(17e)}\\
&\qquad \vee (\mathbf{X} tick)))) && \text{(17f)}
\end{aligned}
$$

$$
\begin{aligned}
&(\underset{\mathbb{N}}{\neg}(\underset{\mathbb{N}}{\mathbf{G}}((StoB\_REQ_0 \underset{\{4\mathbb{N}+1\},\{4\mathbb{N}+1\}}{\wedge} \underset{\{4\mathbb{N}+1\}}{\neg} BtoS\_ACK_0) && \text{(18a)}\\
&\qquad \underset{\{4\mathbb{N}+1\},\{4\mathbb{N}+0,4\mathbb{N}+2,4\mathbb{N}+3\}}{\vee} ((StoB\_REQ_0 \underset{\{4\mathbb{N}+2\},\{4\mathbb{N}+2\}}{\wedge} BtoS\_ACK_0) && \text{(18b)}\\
&\qquad \underset{\{4\mathbb{N}+2\},\{4\mathbb{N}+0,4\mathbb{N}+3\}}{\vee} (((\underset{\{4\mathbb{N}+3\}}{\neg} StoB\_REQ_0) \underset{\{4\mathbb{N}+3\},\{4\mathbb{N}+3\}}{\wedge} BtoS\_ACK_0) && \text{(18c)}\\
&\qquad \underset{\{4\mathbb{N}+3\},\{4\mathbb{N}+0\}}{\vee} (\underset{\{4\mathbb{N}+1\}}{\mathbf{X}} tick)))))) && \text{(18d)}
\end{aligned}
$$

C.2 An Example from the Business Process Domain

The following example (19) shows applicability and utility of our approach in the business process domain. It is based on example 3 in [HHT11]. We changed (19c) from $\mathbf{F}(i \wedge nr)$ to its current form, as this yields more interesting sets of time points, and we omitted the last constraint in [HHT11], as it is the most complicated yet does not contribute to what we would like to illustrate.

We restate the (slightly adapted) explanation from [HHT11]. (19a): An order (o) must occur. (19b): A payment (p) with non-repudiation (nr) must occur. (19c): An insurance submission (i) with non-repudiation must occur at time point 5. (19d): A goods delivery (g) must occur. (19e): Insurance before payment (p) is forbidden. (19f): If a payment occurs, it must occur at least three time points after the

order. (19g): Goods delivery before payment is forbidden. (19h): If an insurance submission occurs, it must occur either at the time point of the order or one time point later. And (19i): A golden (*gold*) customer must have goods delivered no later than three time points after the time point at which the payment is accomplished.

$$(\mathbf{F}o) \tag{19a}$$
$$\wedge\,(\mathbf{F}(p \wedge nr)) \tag{19b}$$
$$\wedge\,(\mathbf{XXXXX}(i \wedge nr)) \tag{19c}$$
$$\wedge\,(\mathbf{F}g) \tag{19d}$$
$$\wedge\,((\neg i)\mathbf{W}p) \tag{19e}$$
$$\wedge\,((\neg p)\mathbf{W}(o \wedge (\neg p) \wedge (\mathbf{X}\neg p) \wedge (\mathbf{XX}\neg p))) \tag{19f}$$
$$\wedge\,((\neg g)\mathbf{W}p) \tag{19g}$$
$$\wedge\,(\mathbf{G}(o \rightarrow \mathbf{XXG}\neg i)) \tag{19h}$$
$$\wedge\,(gold \rightarrow (\mathbf{G}(p \rightarrow (g \vee (\mathbf{X}g) \vee (\mathbf{XX}g) \vee (\mathbf{XXX}g))))) \tag{19i}$$

The UC with sets of time points in (20) consists of parts of (19c), (19e), (19f), and, (19h). We abbreviate the interval of time points from a to b (inclusive) as $[a,b]$. (20a) prescribes that i is 1 at time point 5. With (20b) this implies that p must become 1 between time points 0 and 5. Moreover, with (20d) o must be 0 from time point 0 to 3. Note, though, that the annotation of (20d) with sets of time points tells us that o having to be 0 matters only between time points 0 and 2. (20c) demands that at or before the time point at which p becomes 1 the right operand of its $\mathbf{W}$ operator becomes 1. This operand becoming 1 cannot happen between time points 0 and 2, as that would imply o being 1 at one of those time points. On the other hand, if it were to happen between time points 3 and 5, one of the conjuncts $\neg p$, $\mathbf{X}\neg p$, or $\mathbf{XX}\neg p$ would prevent p from being 1 at or before time point 5. Hence, (20) is unsatisfiable.

$$(\underset{\{1\}}{\mathbf{X}}\ \underset{\{2\}}{\mathbf{X}}\ \underset{\{3\}}{\mathbf{X}}\ \underset{\{4\}}{\mathbf{X}}\ \underset{\{5\}}{\mathbf{X}}\ (i \underset{\{5\},\emptyset}{\wedge} 1)) \tag{20a}$$
$$\underset{\{0\},\{0\}}{\wedge}((\underset{\{5\}}{\neg} i)\ \underset{\{5\},[0,5]}{\mathbf{W}}\ p) \tag{20b}$$
$$\underset{\{0\},\{0\}}{\wedge}((\underset{[0,5]}{\neg} p)\ \underset{[0,5],[0,5]}{\mathbf{W}}\ (o \underset{[0,2],[0,5]}{\wedge}((\underset{[0,5]}{\neg} p) \underset{[0,5],\{3,4\}}{\wedge}((\underset{\{4,5\}}{\mathbf{X}}\ \underset{\{4,5\}}{\neg} p) \underset{\{3,4\},\{3\}}{\wedge}(\underset{\{4\}}{\mathbf{X}}\ \underset{\{5\}}{\mathbf{X}}\ \underset{\{5\}}{\neg} p))))) \tag{20c}$$
$$\underset{\{0\},\{0\}}{\wedge}(\underset{[0,2]}{\mathbf{G}}\ (o \underset{[0,2],[0,2]}{\rightarrow} \underset{[1,3]}{\mathbf{X}}\ \underset{[2,4]}{\mathbf{X}}\ \underset{\{5\}}{\mathbf{G}}\ \underset{\{5\}}{\neg} i)) \tag{20d}$$

(20) is the UC we obtained with our implementation. It shows that extracting UCs from proofs does not necessarily lead to minimal ([Sch12b]: irreducible) or minimum ([Sch12b]: least-cost irreducible) UCs. Consider the variant of (20) with sets of time points removed. While this variant is a UC of (19) without sets of time points, it is not minimal: the last conjunct $\mathbf{XX}\neg p$ in (20c) could be replaced with 1 without making the result satisfiable. In (20) with sets of time points as shown above that subformula is required for unsatisfiability. However, $\underset{[0,2],[0,5]}{\wedge}((\underset{[0,5]}{\neg} p) \underset{[0,5],\{3,4\}}{\wedge}$ in (20c) could be replaced with $\underset{[0,2],[3,5]}{\wedge}((\underset{[3,5]}{\neg} p) \underset{[3,5],\{3,4\}}{\wedge}$ without sacrificing unsatisfiability.

D Additional Plots

Figure 4 shows the overhead that is incurred by extracting UCs with sets of time points by category. Figure 5 compares Gawrychowski's and Sawa's algorithm for computing sets of time points by category.

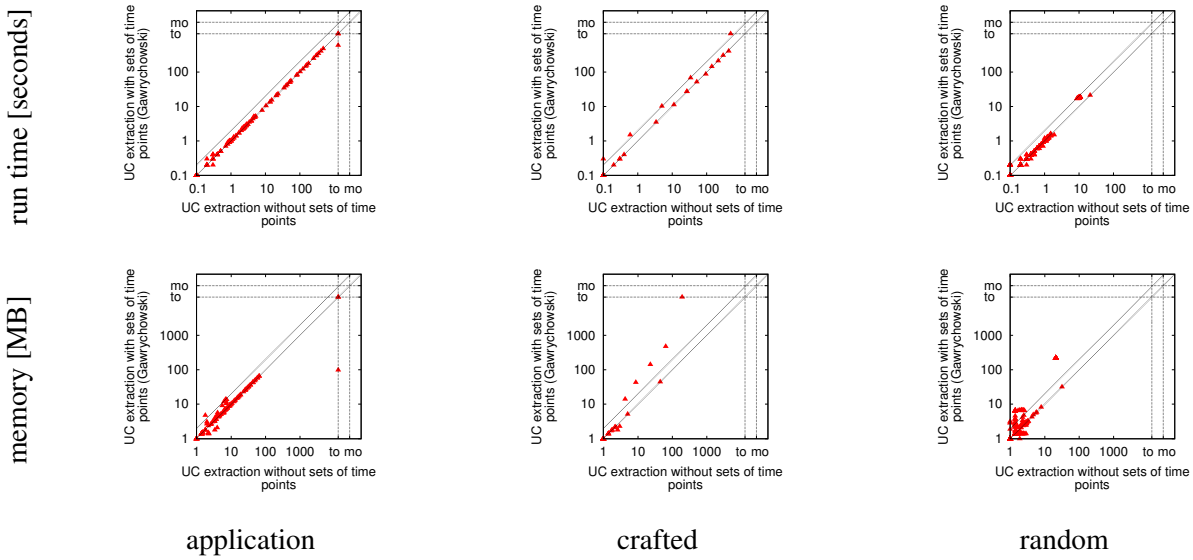


Figure 4: Overhead incurred by UC extraction in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**. In each graph extraction of UCs with time points using Gawrychowski's algorithm is on the y-axis and UC extraction without sets of time points is on the x-axis. The off-center diagonal shows where $y = 2x$.

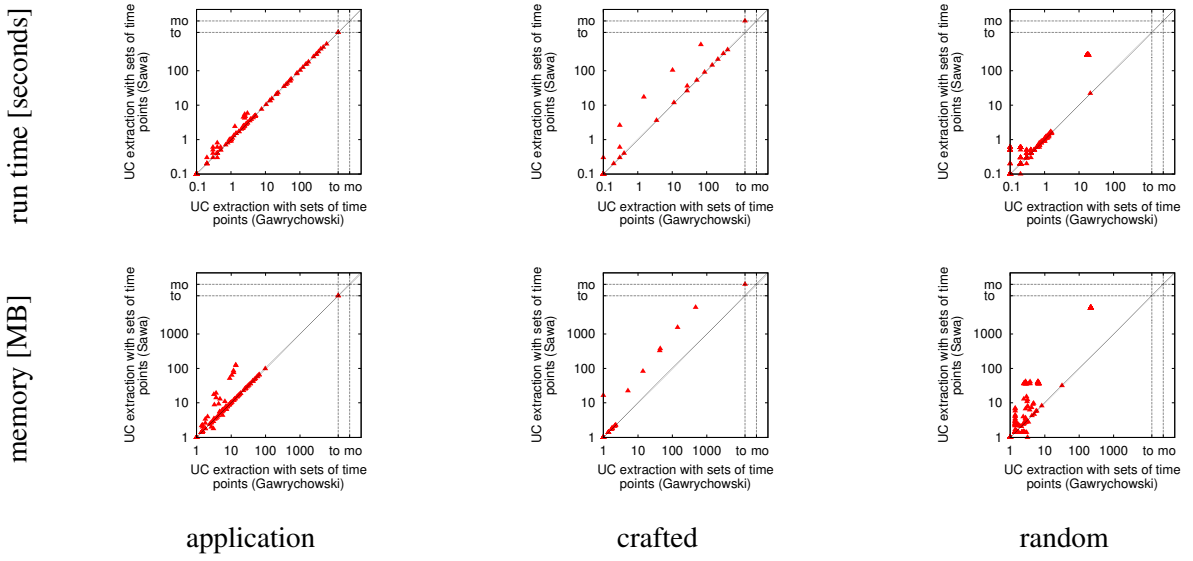


Figure 5: Comparison of using Sawa's algorithm (y-axis) versus Gawrychowski's algorithm (x-axis) for extracting UCs with sets of time points in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**.