Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance[☆]
(full version; r1450, March 21, 2016)

Viktor Schuppan

Abstract

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 information about the time points at which the subformulas of the UC are relevant for unsatisfiability. In previous work we showed how to obtain a UC in LTL by translating the LTL formula into a clausal normal form, applying temporal resolution, extracting a clausal UC from the resolution proof, and mapping the clausal UC back to a UC in LTL. In this paper we extend that method by extracting information at which time points the clauses of a clausal UC are relevant for unsatisfiability from a resolution proof and by transferring that information to a UC in LTL. We implement our method in TRP++, and we experimentally evaluate it.

Keywords: LTL, unsatisfiable cores, vacuity, temporal resolution, resolution graphs, Parikh images

1. Introduction

1.1. Motivation

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., [CCM+10]) automated support for determining a reason for unsatisfiability of a specification is crucial. For many specification languages it is possible to point out a part of the unsatisfiable specification, which has been obtained by removing or relaxing parts of the unsatisfiable specification and which is by itself unsatisfiable, as a reason for unsatisfiability (e.g., [Sch12b, BDTW93, CD91]). In some domains such as SAT (e.g., [GN03, ZM03, Hoo99]), SMT (e.g., [CGS11]), declarative specifications (e.g., [TCJ08]), and LTL (e.g., [Sch12b]) this is called an unsatisfiable core (UC).

LTL (e.g., [Pnu77, Eme90]) and its relatives are important specification languages for reactive systems (e.g., [EF06]) and for business processes (e.g., [PvdA06]). Experience in verification (e.g., [BBDER01, Kup06]) and in synthesis (e.g., [BGJ+07]) has lead to specifications in LTL becoming objects of analysis themselves. Clearly, determining satisfiability of a specification in LTL is an important check (e.g., [RV10]), and providing a UC for an unsatisfiable specification can help the user track down the problem (e.g., [AGH+12]). Besides checking satisfiability other, less simplistic, ways to examine an LTL specification ϕ exist [PSC+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

[☆] A preliminary version of this paper appeared in [Sch13].

Email address: Viktor.Schuppan@gmx.de (Viktor Schuppan)
URL: http://www.schuppan.de/viktor/ (Viktor Schuppan)

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., [BBDER01, KV03, AFF+03, GC04, FKSFV08, Kup06]).

Trying to help users to understand counterexamples in verification, which are essentially witnesses to satisfiable formulas, is a well established research topic (see, e.g., [BBDC+09] for some references). In particular, it is common to add information to a counterexample on which parts of a counterexample are relevant at which points in time (e.g., [RS04, BBDC+09]). According to [BBDC+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, Sch15], are an important feature of industrial hardware verification tools (see, e.g., [BBDER01, AFF+03]). In the academic world UCs are an important part of design methods for embedded systems (e.g., [PSC+06]) as well as for business processes (e.g., [AGH+12]). Despite this relevance of UCs efforts to provide additional information in the context of UCs or vacuity have remained isolated (e.g., [SDGC10]). In this paper we suggest to enhance UCs for LTL with information on the time points at which their subformulas are relevant for unsatisfiability.

1.2. Example

As illustration consider the example in (1). It can be read as "globally p and next time not p" (an alternative verbalization to "globally" is "always"). It is evaluated on infinite words over the alphabet $\{\emptyset, \{p\}\}$; intuitively, a word maps each time point in $\mathbb{N} = 0, 1, 2, \ldots$ to the set of atomic propositions TRUE at that time point. The first conjunct, $\mathbf{G}p$, requires p to be TRUE at all time points, which of course includes time point 1. The second conjunct, $\mathbf{X}\neg p$, requires p to be FALSE at time point 1. Clearly, on any word at most one of the two conjuncts can hold, i.e., (1) is unsatisfiable.

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

When (1) is evaluated on some word π according to the standard semantics of LTL (see Sec. 3), (1) and both of its conjuncts, $\mathbf{G}p$ and $\mathbf{X}\neg p$, are evaluated 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).

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

Remember that the second conjunct, $\mathbf{X}\neg p$, requires p to be FALSE at time point 1. Therefore, to conclude unsatisfiability of (1) it is sufficient to know that the first conjunct, $\mathbf{G}p$, requires p to be TRUE at time point 1; the fact that $\mathbf{G}p$ requires p to be TRUE also at all time points in $\mathbb{N} \setminus \{1\}$ is immaterial. This means that in the evaluation of $\mathbf{G}p$ the operand p would only need to be evaluated at time point 1. At all other time points in $\mathbb{N} \setminus \{1\}$ it could be replaced with, e.g., TRUE without losing unsatisfiability. Using this information, (2) can be modified by replacing $\mathbb{N}$ below $\mathbf{G}$ with $\{1\}$, obtaining (3). (3) can be seen as a UC of (1).

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

1.3. Contributions

Enhancing UCs for LTL with Sets of Time Points In [Sch12b, Sch15] a basic notion of UCs for LTL is used that replaces positive polarity occurrences of subformulas of an LTL formula ϕ with TRUE and negative polarity occurrences of subformulas of ϕ with FALSE provided that the modified formula is still unsatisfiable. After [Sch12b] we term that notion of UCs "UCs for LTL via syntax trees". In this paper we extend that notion of UCs by incorporating information on the time points at which the occurrences of the subformulas of a UC, which were not replaced with TRUE or FALSE, are relevant for unsatisfiability.

UCs for LTL with sets of time points can help users understand why a UC is unsatisfiable by making the following information explicit. (i) Sets of time points can show that invariants only need to hold at certain

time points rather than always to guarantee unsatisfiability. For an example see (3). (ii) Sets of time points can make cyclic interaction between subformulas that lead to unsatisfiability clear, including period and offset of the cycle. This is particularly noteworthy because, as is well known, LTL cannot count (e.g., [Wol83]). For an example see (5). (iii) Finally, because positive and negative polarity occurrences of propositions need to interact at the same time points to obtain unsatisfiability, sets of time points can limit the subformulas that need to be taken into account when trying to understand why a UC is unsatisfiable. For an example see (10).

Making the above mentioned additional information explicit to help users' understanding is achieved by providing a more fine-grained notion of UCs for LTL rather than by providing a proof of unsatisfiability as an explanation. This makes it unnecessary for users to learn a proof calculus and keeps the door open for applications other than debugging (see below). Note also that not all proof calculi that might be chosen as a means to explain the unsatisfiability of a UC will make information regarding the relevance of parts of a UC at certain time points as clear to the user as our proposed notion of UCs for LTL with sets of time points.[1] Hence, depending on the proof calculus, our suggestion to enhance UCs for LTL with sets of time points may be orthogonal to the idea of explaining the unsatisfiability of an LTL formula by providing a proof.

The notion of UCs for LTL with sets of time points naturally extends the notion of UCs for LTL via syntax trees as follows. The notion of UCs for LTL via syntax trees removes whole occurrences of subformulas from an unsatisfiable LTL formula by replacing them with TRUE or FALSE depending on polarity. The notion of UCs for LTL with sets of time points extends that by replacing occurrences of subformulas with TRUE or FALSE depending on polarity only at specific time points. Hence, removing a whole occurrence of a subformula ψ by replacing it with TRUE or FALSE as is done in the notion of UCs for LTL via syntax trees is the limiting case of assigning that occurrence of ψ an empty set of time points at which that occurrence of ψ is relevant in the notion of UCs for LTL with sets of time points. For a proof see Prop. 5.

The notion of UCs for LTL with sets of time points is more fine-grained than the notions of UCs for LTL in previous work [Sch12b, Sch15, AGH+12, GHST13, HH11, HSH12]. In [Sch12b] we discuss various notions of UCs for LTL, and we briefly mention the idea to indicate time points at which subformulas of a UC in LTL are relevant for unsatisfiability. However, 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, and it is not implemented. Later, example (4) is proposed, and it is conjectured that sets of time points can be obtained from a tableau method and that these sets are semilinear. In this paper we take up where we left off in [Sch12b] by formalizing the idea of indicating time points at which subformulas of a UC in LTL are relevant, showing how to obtain the required information, and providing an implementation and experimental evaluation. [Sch15] suggests, implements, and experimentally evaluates a method to obtain UCs for LTL via syntax trees. In addition, it extends that notion of UCs by pointing out which occurrences of atomic propositions interact in a UC. This extension is orthogonal to the extension presented in this paper; combining both is left as future work. [HH11] and [HSH12] use the notion of UCs for LTL via syntax trees. [AGH+12] and [GHST13] take as input a set of LTL formulas ϕ.[2] If that set of LTL formulas is unsatisfiable, then they produce a subset $\phi^{uc} \subseteq \phi$ that is still unsatisfiable. However, they treat the LTL formulas that are the elements of ϕ as atomic entities; i.e., they do not analyze whether all subformulas of the LTL formulas that make up ϕ^{uc} are required for unsatisfiability. This notion of UCs for LTL is less fine-grained than the notion of UCs for LTL via syntax trees.

Temporal Resolution-Based Method to Obtain UCs for LTL with Sets of Time Points In [Sch15] we present a temporal resolution-based method to obtain UCs for LTL via syntax trees. The method translates an LTL formula into a clausal normal form, applies temporal resolution, extracts a clausal UC from the resolution proof, and maps the clausal UC back to a UC in LTL. In this paper we extend that method to obtain information on the time points at which subformulas of a UC in LTL are relevant for unsatisfiability by performing a detailed analysis of the resolution proof. As a first step we determine whether in an application of a resolution rule a premise is time-shifted one step into the future with respect to the conclusion or not. Then we count the number

[1] Temporal resolution as used in this article is an example of a proof calculus that does not make information on temporal relevance immediately clear to the user.

[2] A set of LTL formulas ϕ is interpreted as the conjunction of the members of ϕ.

of such time-shifts that occur in any sequence of proof steps between a clause from the set of clauses whose unsatisfiability is being proved and the empty clause that concludes the proof. Finally, we transfer the information from the clausal UC to the UC in LTL. The analysis takes time cubic in the size of the resolution proof.[3]

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 [Sch15], which provides a clean framework for extracting information from the proof. Note, that while the BDD-based solver NuSMV [CCG+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 [HJSS95] and pltl [plt] provide access to a proof of unsatisfiability comparable to temporal resolution, yet tended to perform not as good on unsatisfiable instances in [SD11].

Publicly Available Implementation We implement our method in TRP++. We make the source code of our solver publicly available. We are not aware of any other tool that performs extraction of UCs for propositional LTL at that level of granularity. The granularity of existing tools has been discussed above; for a more extensive comparison we refer to [Sch15].

Experimental Evaluation Our experimental evaluation demonstrates that (i) non-trivial and interesting sets of time points at which subformulas of a UC in LTL are relevant for unsatisfiability are indeed obtained on practical examples and (ii) computation of time points is possible with quite acceptable overhead in terms of run time and memory usage.

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., [CTVW03, CRST07]). While our results might also benefit that application, we focus on debugging.

Conceptually, under the frequently legitimate assumption that a system description can be translated into an LTL formula, our results extend to vacuity for LTL (see [Sch15]).

1.4. Related Work

Related work concerning the granularity of other notions of UCs for LTL was just discussed in Sec. 1.3. Below we mention some work regarding the role of time points in other work in formal verification, the issue of granularity in notions of UCs in other domains, and possibilities for explaining proofs. For work related to the extraction of UCs for LTL in general we refer to [Sch15, Sch12b].

Simmonds et al. [SDGC10] use SAT-based bounded model checking (e.g., [BCCZ99, 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. Given a specification as an LTL formula and an infinite word that does not satisfy the specification Pill and Quaritsch [PQ13] use Reiter's approach to diagnosis [Rei87, GSW89] to compute the set of potential changes to the specification (diagnoses) such that the word satisfies the modified specification. The algorithm they use to compute diagnoses [Rei87, GSW89] amounts to enumerating all UCs of the specification conjoined with the word. The diagnoses they compute are at the granularity of an occurrence of a subformula; they avoid computing diagnoses that take time points into account because of the expected increase in computational effort. Some work [RS04, BBDC+09] determines the time points at which propositions in witnesses of satisfiable LTL formulas are relevant for satisfiability. An unsatisfiable LTL specification is an instance of overconstraint in a specification (e.g., [SSJ+03]); an example of an approach that deals with the complementary case of underconstraint is [Cla07], which finds pairs of output signals and time points in a specification such that the value of that output signal is not covered by the specification at that time point. Dong et al. [DSRS02] propose a fine-grained notion of vacuity for the modal μ-calculus (e.g., [Koz83, BS01]). In [DRS03] Dong et al. present a method and a tool to explore proofs of correctness in model checking the modal μ-calculus.

[3] Note that the size of the resolution proof may be exponential in the size of the given LTL formula [Sch15, FDP01, Dix98].

UCs are frequently used in description logics (e.g., [BCM+07]) to help debugging inconsistent ontologies. At first, subsets of axioms were proposed as a UC [BH95]. Later, more fine-grained notions of UC were introduced that are similar to our notion of a UC via syntax tree but go beyond that by allowing to replace concepts with more general concepts and to numerically relax cardinality restrictions (e.g., [HPS08, Kal06, SC03]). To help understanding more complex UCs it is suggested to extend a UC to a proof (e.g., [HPS10b, DHS05]). The proof may then be converted into natural language (e.g., [NPPW12]). For a more extensive discussion of UCs in description logics see [Hor11, Ngu13, Kal06, Den10].

In [GMP07] Grégoire et al. propose a notion of UCs for constraint satisfaction problems (e.g., [Apt03]), which is more fine-grained than previous notions in that it does not remove a whole constraint (which is interpreted in [GMP07] as a set of forbidden solutions) but only a part of a constraint (corresponding to a single or few forbidden solutions).

Helping users understand automatically generated proofs is a major concern in automated theorem proving (e.g., [RV01]). Some approaches are the transformation of proofs from a more machine-oriented calculus such as resolution to a more human-oriented calculus such as natural deduction (e.g., [Lin89]), the presentation of proofs at higher levels of abstraction (e.g., [Hua94]), the adaptation of the abstraction level in a user-dependent fashion with the possibility of user interaction (e.g., [Fie01]), the presentation of proofs in natural language (e.g., [Fie01]), and, of course, graphical user interfaces (e.g., [SHB+99]).

1.5. Structure of the Paper

This paper builds on a fair amount of previous work: we use temporal resolution as implemented in TRP++ [FDP01, Dix98, Dix97, HK03, HK04, trp] and its extension to extract UCs [Sch15]. To make this paper self-contained we provide a brief 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.

A reader who is mostly interested in the application of this work without necessarily understanding how it works is referred to Sec. 2 with several motivating examples. A walk-through of the core part of the approach by way of example is given in Sec. 6.1. Section 2 requires no and Sec. 6.1 only a basic idea of resolution.

We start with a number of motivating examples in Sec. 2. In Sec. 3 the more formal exposition begins with preliminaries. In Sec. 4 we restate the construction of a resolution graph and its use to obtain a UC from [Sch15]. The main technical part of our contribution in this paper can be found in Sec. 5 and 6 where we show how to compute the time points at which subformulas are relevant for unsatisfiability. We discuss our implementation and experimental evaluation in Sec. 7. Some other aspects are discussed 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 http://www.schuppan.de/viktor/theoreticalcomputerscience16/.

2. Motivating Examples

In this section we present examples that show the utility of UCs with sets of time points for debugging. The first example is more involved than the introductory example (1)–(3) in the previous section but still artificial to allow focusing entirely on sets of time points. The subsequent examples are then closer to real world situations. The UCs in this as well as in all other parts of this paper were obtained with our implementation, possibly except for minor rewriting. Note, though, that the examples in this section illustrate the benefits of UCs with sets of time points independent of the method they are obtained with. For an additional example from the business process domain see Appendix B.

We formally introduce LTL only in Sec. 3 and LTL with sets of time points (LTLp) only in Sec. 5. Readers entirely unfamiliar with LTL may therefore prefer to skip to Sec. 3 first. For readers with some knowledge of LTL we next provide an intuition on the semantics of LTLp but refer to the formal treatment in Sec. 5 for details.

Let $\psi \equiv \underset{I}{\circ_1}\psi'$ be an occurrence of a subformula with a unary operator $\circ_1$ and an operand ψ' that has a set of time points I (the case of binary operators is analogous). Assume that the valuation of ψ' is known for all time points and that we now would like to evaluate ψ at some time point i. The valuation of ψ at time point i is essentially determined in the same way as in the standard definition of LTL without sets of time points, except for the following. When in the evaluation of ψ at time point i the valuation of ψ' is required at some time point i' such that i' is not contained in I, then the valuation of ψ' at time point i' is replaced with TRUE if ψ has positive polarity and with FALSE if ψ has negative

polarity. As an example consider $\psi \equiv \underset{\{1\}}{\mathbf{G}}\, p$ from (3). When ψ is evaluated at time point 0 on some word π over $\{\emptyset, \{p\}\}$, then the valuation of ψ is obtained as the conjunction of TRUE at time point 0 (because 0 is not contained in $\{1\}$ and, therefore, TRUE is used rather than the valuation of p), the valuation of p at time point 1 (because 1 is contained in $\{1\}$ and, therefore, the valuation of p is used as in standard LTL without sets of time points), and TRUE at all time points ≥ 2 (because these time points are not contained in $\{1\}$ and, therefore, TRUE is used rather than the valuation of p): $((\pi, 0) \models \underset{\{1\}}{\mathbf{G}}\, p) \Leftrightarrow (\forall i \in \mathbb{N} \,.\, ((i \notin \{1\}) \vee ((\pi, i) \models p))) \Leftrightarrow (\text{TRUE} \wedge (p \in \pi[1]) \wedge \bigwedge_{i \geq 2} \text{TRUE})$.

2.1. *Every Second Time Point*

Consider (4). The first conjunct, p, forces p to be TRUE at time point 0. Triggered by that, the second conjunct, $\mathbf{G}(p \to \mathbf{XX}p)$, then forces p to be TRUE at even time points ≥ 2. Finally, the third conjunct, $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$, requires that eventually p becomes FALSE at two consecutive time points. Clearly, the first two conjuncts contradict the third, i.e., (4) is unsatisfiable.

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

In (5) we show (4) enhanced with sets of time points such that the result is still unsatisfiable. The three conjuncts p, $\mathbf{G}(p \to \mathbf{XX}p)$, and $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$ are evaluated only at time point 0 (see the sets of time points below the first two $\wedge$ operators). The set of time points below the $\mathbf{G}$ operator, $2 \cdot \mathbb{N}$, shows that the operand of the $\mathbf{G}$ operator, $p \to \mathbf{XX}p$, is evaluated only at even time points. This carries over to both operands of the $\to$ operator. Consequently, it is sufficient to evaluate $\mathbf{X}p$ at odd time points and its operand, p, at even time points > 0. In the last conjunct the operand of the $\mathbf{F}$ operator has to be evaluated at every time point; otherwise, if for some time point the operand of the $\mathbf{F}$ operator were replaced with TRUE, then $\mathbf{F}((\neg p) \wedge \mathbf{X}\neg p)$ would evaluate to TRUE. The sets of time points below the $\wedge$ operator in $(\neg p) \wedge \mathbf{X}\neg p$ show that the left conjunct, $\neg p$, contradicts p from the first part of (5) when $(\neg p) \wedge \mathbf{X}\neg p$ is evaluated at an even time point, and the right conjunct, $\mathbf{X}\neg p$, does the same when $(\neg p) \wedge \mathbf{X}\neg p$ is evaluated at an odd time point. Finally, when the left conjunct, $\neg p$, is evaluated at even time points, then so is its operand, p, and when the right conjunct, $\mathbf{X}\neg p$, is evaluated at time points $2 \cdot \mathbb{N} + 1$, then its operand, $\neg p$, as well as p itself are evaluated at time points $2 \cdot \mathbb{N} + 2$. We call (5) a UC of (4) in LTL with sets of time points.

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

Example (4) is still relatively small and, therefore, reasonably easy to comprehend in its entirety. Still, sets of time points in (5) highlight two important aspects. First, the operand of the second conjunct, $p \to \mathbf{XX}p$, is evaluated only every second time point, as can be seen from the set of time points below the $\mathbf{G}$ operator. Second, one of the two conjuncts in the operand $(\neg p) \wedge \mathbf{X}\neg p$ of the $\mathbf{F}$ operator contradicts p at even time points and the other one at odd time points, as evidenced by the sets of time points below the last $\wedge$ operator.

2.2. *A Lift Specification*

The example (6) modifies the example of a lift specification from [Sch15] (originally adapted from [Har05]) to illustrate the use of sets of time points in debugging. 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 TRUE if some button is pressed. If the lift moves up, then up must be TRUE; if it moves down, then up must be FALSE. u switches turns between actions by users of the lift (u is TRUE) and actions by the lift (u is FALSE). For a more detailed explanation we refer to [Har05].

$$(\neg u) \wedge (f_0) \wedge (\neg b_0) \wedge (\neg b_1) \wedge (\neg up) \tag{6a}$$
$$\wedge\ (\mathbf{G}((u \to \neg \mathbf{X}u) \wedge ((\neg \mathbf{X}u) \to u))) \tag{6b}$$
$$\wedge\ (\mathbf{G}(f_0 \to \neg f_1)) \tag{6c}$$
$$\wedge\ (\mathbf{G}((f_0 \to \mathbf{X}(f_0 \vee f_1)) \wedge (f_1 \to \mathbf{X}(f_0 \vee f_1)))) \tag{6d}$$
$$\wedge\ (\mathbf{G}(u \to ((f_0 \to \mathbf{X}f_0) \wedge ((\mathbf{X}f_0) \to f_0) \wedge (f_1 \to \mathbf{X}f_1) \wedge ((\mathbf{X}f_1) \to f_1)))) \tag{6e}$$
$$\wedge\ (\mathbf{G}((\neg u) \to ((b_0 \to \mathbf{X}b_0) \wedge ((\mathbf{X}b_0) \to b_0) \wedge (b_1 \to \mathbf{X}b_1) \wedge ((\mathbf{X}b_1) \to b_1)))) \tag{6f}$$
$$\wedge\ (\mathbf{G}(((b_0 \wedge \neg f_0) \to \mathbf{X}b_0) \wedge ((b_1 \wedge \neg f_1) \to \mathbf{X}b_1))) \tag{6g}$$
$$\wedge\ (\mathbf{G}((f_0 \wedge \mathbf{X}f_0) \to ((up \to \mathbf{X}up) \wedge ((\mathbf{X}up) \to up)))) \tag{6h}$$
$$\wedge\ (\mathbf{G}((f_1 \wedge \mathbf{X}f_1) \to ((up \to \mathbf{X}up) \wedge ((\mathbf{X}up) \to up)))) \tag{6i}$$
$$\wedge\ (\mathbf{G}(((f_0 \wedge \mathbf{X}f_1) \to up) \wedge ((f_1 \wedge \mathbf{X}f_0) \to \neg up))) \tag{6j}$$

$$\wedge\ (\mathbf{G}((sb \rightarrow (b_0 \vee b_1)) \wedge ((b_0 \vee b_1) \rightarrow sb))) \tag{6k}$$
$$\wedge\ (\mathbf{G}((f_0 \wedge \neg sb) \rightarrow (f_0 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f_0) \wedge \neg up))))) \tag{6l}$$
$$\wedge\ (\mathbf{G}((f_1 \wedge \neg sb) \rightarrow (f_1 \mathbf{U}(sb \mathbf{R}((\mathbf{F} f_0) \wedge \neg up))))) \tag{6m}$$
$$\wedge\ (\mathbf{G}((b_0 \rightarrow \mathbf{F} f_0) \wedge (b_1 \rightarrow \mathbf{F} f_1))) \tag{6n}$$

We first assume that an engineer is interested in seeing whether it is possible that b_1 is continuously pressed (7). As the UC with sets of time points in (8) shows this is impossible because b_1 must be FALSE at time point 0. Notice that (8) indicates that the operand of the **G** operator is only needed at time point 0 (trivial to see in this case).

$$\mathbf{G} b_1 \tag{7}$$
$$(\underset{\{0\}}{\neg}\, b_1) \underset{\{0\},\{0\}}{\wedge} \underset{\{0\}}{\mathbf{G}}\, b_1 \tag{8}$$

Now the engineer modifies her query such that b_1 is pressed continuously only from time point 1 on (9). That turns out to be impossible, too. The sets of time points in the UC in (10) highlight two aspects. First, the set of time points below the second **G** operator shows that also this time the press of b_1 is required only at one time point to obtain unsatisfiability. Second, the sets of time points show that for unsatisfiability the first occurrence of b_1 must interact with the third occurrence of b_1 (at time point 0) and the second occurrence of b_1 must interact with the fourth occurrence of b_1 (at time point 1).

$$\mathbf{X}\mathbf{G} b_1 \tag{9}$$
$$(\underset{\{0\}}{\neg} u) \underset{\{0\},\{0\}}{\wedge} ((\underset{\{0\}}{\neg} b_1) \underset{\{0\},\{0\}}{\wedge} ((\underset{\{0\}}{\mathbf{G}} ((\underset{\{0\}}{\neg} u) \underset{\{0\},\{0\}}{\rightarrow} ((\underset{\{1\}}{\mathbf{X}} b_1) \underset{\{0\},\{0\}}{\rightarrow} b_1))) \underset{\{0\},\{0\}}{\wedge} \underset{\{1\}}{\mathbf{X}}\, \underset{\{1\}}{\mathbf{G}} b_1)) \tag{10}$$

The engineer now tries to have b_1 pressed continuously only from time point 2 on and also obtains a UC that needs b_1 pressed only at a single time point for unsatisfiability (not shown). She suspects that the fact that b_1 is pressed continuously in her scenario may play no role for unsatisfiability 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.

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

Example (11) below is based on a subset of the specification of a buffer in [BGJ+07]. The subset regulates the communication between a sender and a buffer, where the sender can issue requests to the buffer (*req*) and the buffer can acknowledge requests to the sender (*ack*). (11) characterizes a behavior in which the following sequence of steps (i)–(iv) is repeated either infinitely often or until an instance of step (i) continues indefinitely: (i) A (finite or infinite) non-empty sequence of time points in which both *req* and *ack* are FALSE, (ii) followed by a finite non-empty sequence of time points in which *req* is TRUE and *ack* is FALSE, (iii) followed by a single time point in which both *req* and *ack* are TRUE, (iv) followed by a finite non-empty sequence of time points in which *req* is FALSE and *ack* is TRUE. In other words, a period of inactivity is followed by a request. The request remains asserted until it is eventually acknowledged. Once the request has been acknowledged, the request is deasserted at the following time point. Finally, the acknowledgment may be kept asserted for an arbitrary but finite amount of time, during which no further request may be issued. Then the cycle repeats itself.

$$(\neg req) \wedge (\neg ack) \tag{11a}$$
$$\wedge\ (\mathbf{G}((req \wedge \neg ack) \rightarrow \mathbf{X} req)) \tag{11b}$$
$$\wedge\ (\mathbf{G}(ack \rightarrow \mathbf{X} \neg req)) \tag{11c}$$
$$\wedge\ (\mathbf{G}\mathbf{F}((req \wedge ack) \vee ((\neg req) \wedge \neg ack))) \tag{11d}$$
$$\wedge\ (\mathbf{G}(((\neg req) \wedge \mathbf{X} req) \rightarrow \mathbf{X} \neg ack)) \tag{11e}$$
$$\wedge\ (\mathbf{G}(((\neg ack) \wedge \mathbf{X} ack) \rightarrow req)) \tag{11f}$$
$$\wedge\ (\mathbf{G}((ack \wedge req) \rightarrow \mathbf{X} ack)) \tag{11g}$$

The engineer considers a scenario in which a request is sent every fourth time point, starting at time point 1 (12a), (12b). In this scenario she would like to verify an invariant (12c)–(12f). Notice that the four disjuncts of the invariant essentially correspond to the four steps (i)–(iv) above (except for the occurrence of $\mathbf{X}$*tick* in (12c)). Hence, in a cycle of length 4 each disjunct of the invariant should happen at exactly one time point.

$$(\mathbf{X} tick) \wedge (\mathbf{G}(tick \rightarrow \mathbf{X}\mathbf{X}\mathbf{X}\mathbf{X} tick)) \tag{12a}$$

$$\wedge\ (\mathbf{G}((tick \rightarrow req) \wedge ((\mathbf{X}tick) \rightarrow \neg req))) \tag{12b}$$

$$\wedge\ \neg\mathbf{G}(((\neg req) \wedge (\neg ack) \wedge \mathbf{X}tick) \tag{12c}$$

$$\vee\ (req \wedge \neg ack) \tag{12d}$$

$$\vee\ (req \wedge ack) \tag{12e}$$

$$\vee\ ((\neg req) \wedge ack)) \tag{12f}$$

The fact that the conjunction of (11) and (12) is unsatisfiable (only) proves that the invariant holds. Inspection of the excerpt of the resulting UC with sets of time points for the invariant in (13a)–(13d) then explicitly confirms that indeed each disjunct of the invariant holds every fourth point in time and in the exact order of the steps (i)–(iv) above, as can be seen from the sets of time points below the $\wedge$ operators in each of the disjuncts.

$$\underset{\{0\}}{\neg}\ \underset{\mathbb{N}}{\mathbf{G}}(\quad ((\underset{4\cdot\mathbb{N}}{\neg}\ req) \underset{4\cdot\mathbb{N},4\cdot\mathbb{N}}{\wedge} ((\underset{4\cdot\mathbb{N}}{\neg}\ ack) \underset{4\cdot\mathbb{N},4\cdot\mathbb{N}}{\wedge} \underset{4\cdot\mathbb{N}+1}{\mathbf{X}}\ tick)) \tag{13a}$$

$$\underset{4\cdot\mathbb{N},\{4\cdot\mathbb{N}+1,4\cdot\mathbb{N}+2,4\cdot\mathbb{N}+3\}}{\vee} ((req \underset{4\cdot\mathbb{N}+1,4\cdot\mathbb{N}+1}{\wedge} \underset{4\cdot\mathbb{N}+1}{\neg}\ ack) \tag{13b}$$

$$\underset{4\cdot\mathbb{N}+1,\{4\cdot\mathbb{N}+2,4\cdot\mathbb{N}+3\}}{\vee} ((req \underset{4\cdot\mathbb{N}+2,4\cdot\mathbb{N}+2}{\wedge} ack) \tag{13c}$$

$$\underset{4\cdot\mathbb{N}+2,4\cdot\mathbb{N}+3}{\vee} ((\underset{4\cdot\mathbb{N}+3}{\neg}\ req) \underset{4\cdot\mathbb{N}+3,4\cdot\mathbb{N}+3}{\wedge} ack)))) \tag{13d}$$

3. Preliminaries

3.1. Semilinear Sets and Parikh Images

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 $\Psi : \Sigma^* \times \Sigma \rightarrow \mathbb{N}, (w, \sigma) \mapsto m$ 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].

3.2. LTL

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. The Boolean constants FALSE, 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 $(\psi\mathbf{U}\psi') \vee \mathbf{G}\psi$. 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. The *size* of an LTL formula ϕ is measured as the sum of the numbers of occurrences of atomic propositions, Boolean operators, and temporal operators in ϕ.

LTL is interpreted over words in $(2^{AP})^\omega$. For the semantics of LTL see Fig. 1. A word $\pi \in (2^{AP})^\omega$ *satisfies* an LTL formula ϕ iff $(\pi, 0) \models \phi$. A word π that satisfies ϕ is also called a *satisfying assignment* for ϕ. An LTL formula ϕ is *satisfiable* if there exists a word $\pi \in (2^{AP})^\omega$ that satisfies ϕ; otherwise, it is *unsatisfiable*. The problem of determining the satisfiability of an LTL formula is PSPACE-complete [SC85, HR83].

LTL can be extended with existential quantification over atomic propositions, termed EQLTL (e.g., [SVW87, Eme90]). The syntax and semantics of EQLTL are as follows. If $p_0, \ldots, p_n$ are atomic propositions in AP and ψ is an LTL formula, then $\exists p_0\ .\ \cdots \exists p_n\ .\ \psi$ is an EQLTL formula. A word π in $(2^{AP})^\omega$ satisfies $\exists p_0\ .\ \cdots \exists p_n\ .\ \psi$ iff there exists a word π' in $(2^{AP})^\omega$ such that (i) π' satisfies ψ and (ii) π differs from π' only in the valuation of $p_0, \ldots, p_n$. The satisfiability problem for EQLTL is PSPACE-complete [SVW87].

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

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

3.3. Separated Normal Form

Temporal resolution 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 $\forall 1 \leq j < j' \leq n \,.\, p_j \neq p_{j'}$ and $\forall 1 \leq j < j' \leq n' \,.\, q_j \neq q_{j'}$. 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*. $p_1 \vee \ldots \vee p_n$ is called the *now part* and $q_1 \vee \ldots \vee q_{n'}$ is called the **X** *part* of a global clause. l is called an *eventuality literal*. As usual an empty disjunction (resp. conjunction) stands for FALSE (resp. TRUE). () or ($\mathbf{G}()$), denoted $\Box$, stand for FALSE or $\mathbf{G}$(FALSE) 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 $\bigwedge_{1 \leq j \leq n} c_j$ is an LTL formula in *SNF*. Every LTL formula ϕ can be translated into an equisatisfiable formula ϕ' in SNF [FDP01].

3.4. Translating LTL into SNF

We use a structure-preserving translation (e.g., [PG86]) to translate an LTL formula into a set of SNF clauses. It is based on the tableau construction for LTL that is often used in (symbolic) model checking (see, e.g., [LP85, BCM+92, CGH97]).

Definition 1 (Translation from LTL into SNF)**.** *Let ϕ be an LTL formula over atomic propositions AP, and let $X = \{x, x', \ldots\}$ be a set of fresh atomic propositions that don't occur in ϕ. Assign to each occurrence of a subformula ψ in ϕ a Boolean value or a proposition according to column 2 of Tab. 1, which is used to reference ψ in the SNF clauses for its superformula. Moreover, assign to each occurrence of ψ a set of SNF clauses according to column 3 or 4 of Tab. 1. Let $SNF_{aux}(\phi)$ be the set of all SNF clauses obtained from ϕ that way. Then the* SNF of *ϕ is defined as $SNF(\phi) \equiv x_\phi \wedge \bigwedge_{c \in SNF_{aux}(\phi)} c$.*

Note that to make the SNF clauses in columns 3 and 4 of Tab. 1 and elsewhere in this article easier to understand we often use implication to formulate them. However, in TRP++ SNF clauses cannot contain implications and, therefore, in our implementation of Def. 1 implication is expanded using its definition. The fact that some propositions are marked blue boxed in Tab. 1 will be used later in Sec. 4 and 6 when translating a UC back from SNF to LTL. It is well known that ϕ and $SNF(\phi)$ are equisatisfiable and that a satisfying assignment for ϕ (resp. $SNF(\phi)$) can be extended (resp. restricted) to a satisfying assignment for $SNF(\phi)$ (resp. ϕ). Below we sometimes identify the SNF of ϕ, $SNF(\phi)$, with the set of SNF clauses $\{x_\phi\} \cup SNF_{aux}(\phi)$ that $SNF(\phi)$ is constructed from.

Remark 1 (Complexity Considerations Regarding the Translation from LTL into SNF)**.** *Let ϕ be an LTL formula over atomic propositions AP, and let $SNF(\phi)$ be the SNF of ϕ. It is easy to see that (i) for each occurrence of a Boolean or temporal operator in ϕ one fresh atomic proposition $x, x', \ldots$ is introduced in $SNF(\phi)$ by the translation, (ii) the number of clauses in $SNF(\phi)$ is linear in the size of ϕ, (iii) the size of $SNF(\phi)$ is linear in the size of ϕ, and (iv) $SNF(\phi)$ can be computed in time linear in the size of ϕ.*

Table 1: Translation from LTL into SNF.

Subformula	Proposition	SNF Clauses (positive polarity occurrences)	SNF Clauses (negative polarity occurrences)
TRUE/FALSE/p	TRUE/FALSE/p	none	none
$\neg\psi$	$x_{\neg\psi}$	$(\mathbf{G}(x_{\neg\psi} \rightarrow \neg x_{\psi}))$	$(\mathbf{G}((\neg x_{\neg\psi}) \rightarrow x_{\psi}))$
$\psi \vee \psi'$	$x_{\psi\vee\psi'}$	$(\mathbf{G}(x_{\psi\vee\psi'} \rightarrow (x_{\psi} \vee x_{\psi'})))$	$(\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow \neg x_{\psi}))$, $(\mathbf{G}((\neg x_{\psi\vee\psi'}) \rightarrow \neg x_{\psi'}))$
$\psi \wedge \psi'$	$x_{\psi\wedge\psi'}$	$(\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow x_{\psi}))$, $(\mathbf{G}(x_{\psi\wedge\psi'} \rightarrow x_{\psi'}))$	$(\mathbf{G}((\neg x_{\psi\wedge\psi'}) \rightarrow ((\neg x_{\psi}) \vee \neg x_{\psi'})))$
$\mathbf{X}\psi$	$x_{\mathbf{X}\psi}$	$(\mathbf{G}(x_{\mathbf{X}\psi} \rightarrow \mathbf{X} x_{\psi}))$	$(\mathbf{G}((\neg x_{\mathbf{X}\psi}) \rightarrow \mathbf{X}\neg x_{\psi}))$
$\psi\mathbf{U}\psi'$	$x_{\psi\mathbf{U}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (x_{\psi'} \vee x_{\psi})))$, $(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow (x_{\psi'} \vee \mathbf{X} x_{\psi\mathbf{U}\psi'})))$, $(\mathbf{G}(x_{\psi\mathbf{U}\psi'} \rightarrow \mathbf{F} x_{\psi'}))$	$(\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow \neg x_{\psi'}))$, $(\mathbf{G}((\neg x_{\psi\mathbf{U}\psi'}) \rightarrow ((\neg x_{\psi}) \vee \mathbf{X}\neg x_{\psi\mathbf{U}\psi'})))$
$\psi\mathbf{R}\psi'$	$x_{\psi\mathbf{R}\psi'}$	$(\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow x_{\psi'}))$, $(\mathbf{G}(x_{\psi\mathbf{R}\psi'} \rightarrow (x_{\psi} \vee \mathbf{X} x_{\psi\mathbf{R}\psi'})))$	$(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg x_{\psi'}) \vee \neg x_{\psi})))$, $(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow ((\neg x_{\psi'}) \vee \mathbf{X}\neg x_{\psi\mathbf{R}\psi'})))$, $(\mathbf{G}((\neg x_{\psi\mathbf{R}\psi'}) \rightarrow \mathbf{F}\neg x_{\psi'}))$
$\mathbf{F}\psi$	$x_{\mathbf{F}\psi}$	$(\mathbf{G}(x_{\mathbf{F}\psi} \rightarrow \mathbf{F} x_{\psi}))$	$(\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow \mathbf{X}\neg x_{\mathbf{F}\psi}))$, $(\mathbf{G}((\neg x_{\mathbf{F}\psi}) \rightarrow \neg x_{\psi}))$
$\mathbf{G}\psi$	$x_{\mathbf{G}\psi}$	$(\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow \mathbf{X} x_{\mathbf{G}\psi}))$, $(\mathbf{G}(x_{\mathbf{G}\psi} \rightarrow x_{\psi}))$	$(\mathbf{G}((\neg x_{\mathbf{G}\psi}) \rightarrow \mathbf{F}\neg x_{\psi}))$

Running Example 1. As an example consider (14) below.

$$\phi \equiv ((\mathbf{X}\neg p) \wedge \mathbf{G}\neg q) \wedge (p\mathbf{U}(q \wedge r)). \tag{14}$$

Using Def. 1 (14) is translated into the following set of SNF clauses (15):

$$\begin{aligned}
&\{(x_{\phi}),\\
&(\mathbf{G}(x_{\phi} \rightarrow x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q})), (\mathbf{G}(x_{\phi} \rightarrow x_{p\mathbf{U}(q\wedge r)})),\\
&(\mathbf{G}(x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q} \rightarrow x_{\mathbf{X}\neg p})), (\mathbf{G}(x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q} \rightarrow x_{\mathbf{G}\neg q})),\\
&(\mathbf{G}(x_{\mathbf{X}\neg p} \rightarrow \mathbf{X} x_{\neg p})),\\
&(\mathbf{G}(x_{\neg p} \rightarrow \neg p)),\\
&(\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow \mathbf{X} x_{\mathbf{G}\neg q})), (\mathbf{G}(x_{\mathbf{G}\neg q} \rightarrow x_{\neg q})),\\
&(\mathbf{G}(x_{\neg q} \rightarrow \neg q)),\\
&(\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \rightarrow (x_{q\wedge r} \vee p))), (\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \rightarrow (x_{q\wedge r} \vee \mathbf{X} x_{p\mathbf{U}(q\wedge r)}))), (\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \rightarrow \mathbf{F} x_{q\wedge r})),\\
&(\mathbf{G}(x_{q\wedge r} \rightarrow q)), (\mathbf{G}(x_{q\wedge r} \rightarrow r))\}.
\end{aligned} \tag{15}$$

This example will be continued in Sec. 4.2 to illustrate mapping a UC in SNF to a UC in LTL. □

3.5. *Temporal Resolution in* TRP++

Temporal resolution (TR) [Fis91] extends resolution (e.g., [Rob65, FM09, BG01]) to temporal logics such as LTL (e.g., [Fis91, FDP01]) and CTL (e.g., [BF99]). In this paper we use TR for LTL [Fis91, FDP01] with BFS for loop search [Dix98, Dix97, Dix96, Dix95] as implemented in TRP++ [HK03, HK04, trp], and we refer to the corresponding algorithm as the "TR algorithm" below.

Temporal resolution has been developed since the early 1990s [Fis91], and an extensive body of literature exists. It is out of the scope of this article to provide a detailed introduction or a tutorial on the subject. The following references are among the most suitable as an introduction to TR as needed for this article: [FDP01] provides a general overview of the method and is a good starting point, [Dix97, Dix98] explains BFS loop search as used in TRP++, and [HK04] covers the implementation of TR in TRP++. [Sch15] contains a concise description of the TR algorithm. In [Sch12a] we provide some intuition on temporal resolution with a slant towards BDD-based symbolic model checking (e.g., [BCM+92, CGP01]).

The TR algorithm takes as input a set of SNF clauses C. A number of production rules allow to derive new clauses from clauses in C and/or previously derived clauses, possibly under some side conditions. If the empty clause □ can be derived from C, then C is shown to be unsatisfiable. Otherwise, C is satisfiable.

The production rules used in the TR algorithm 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

Table 2: Production rules used in TRP++.

production rule	premise 1	partition	premise 2	partition	conclusion	partition
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	$c \equiv (\mathbf{G}(P \vee \mathbf{X}(q_1 \vee \ldots \vee q_{n'})))$ with $n' > 0$			M	c	L
BFS-loop-it-init-n	$(\mathbf{G}\ P)$			M	$(\mathbf{G}\,\mathbf{X}\,P)$	L
BFS-loop-it-init-c	$(\mathbf{G}\ P)$	L'	$(\mathbf{G}(Q \vee \mathbf{F}l))$	M	$(\mathbf{GX}(P \vee l))$	L
BFS-loop-it-sub	$(\mathbf{G}\ P)$			L	$(\mathbf{GX}(P \vee Q \vee l))$ 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

are described later. Let $P \equiv \bigvee_{j=1\ldots n_p} p_j$, $Q \equiv \bigvee_{j=1\ldots n_q} q_j$, $R \equiv \bigvee_{j=1\ldots n_r} r_j$, $S \equiv \bigvee_{j=1\ldots n_s} s_j$. Correspondingly, for $j \in \mathbb{N}$ let $P_j \equiv \bigvee_{j'=1\ldots n_{p,j}} p_{j,j'}$, $Q_j \equiv \bigvee_{j'=1\ldots n_{q,j}} q_{j,j'}$, etc.

Resolution between two initial clauses, between two global clauses, or between an initial and a global clause is performed by a straightforward extension of propositional resolution by the five production rules listed under "saturation". Resolution between a set of global clauses and an eventuality clause is more complex. Resolution with an eventuality clause $(\mathbf{G}(P \vee \mathbf{F}l))$ requires to find a set of global clauses that allows one 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.

An instance of loop search may take several iterations, called *BFS loop search iterations*. In a single BFS loop search iteration for some eventuality clause $(\mathbf{G}(P \vee \mathbf{F}l)) \in C$ one tries to show that a hypothetical fixed point is an actual fixed point. Concretely, one makes the hypothesis that (16) holds.

$$\bigwedge_{1 \le j \le n} (\mathbf{GX}(Q_j \vee R_j \vee l)) \tag{16}$$

Now assume that one can derive (17) by applying rule step-xx to (16) and any global clauses contained in or previously derived from C.

$$\bigwedge_{1 \le j \le n} (\mathbf{G}\ Q_j) \tag{17}$$

In that case one has shown that C and (16) imply (17). I.e., if some satisfying assignment for C, π, fulfills $\bigwedge_{1 \le j \le n}(Q_j \vee R_j \vee l)$ at time point $i+1$, then π must fulfill $\bigwedge_{1 \le j \le n} Q_j$ at time point i. Reversing the implication, if some Q_j is FALSE in π at time point i, then some $Q_{j'} \vee R_{j'} \vee l$ must be FALSE in π at time point $i+1$. Note that this means that both $Q_{j'}$ and l must be FALSE in π at time point $i+1$. Hence, by induction l must be FALSE in π for all time points after $i+1$. I.e., we have just shown that C implies (18).

$$\bigwedge_{1 \le j \le n} (\mathbf{G}(Q_j \vee \mathbf{XG}\neg l)) \tag{18}$$

Now, with $(\mathbf{G}(P \vee \mathbf{F}l)) \in C$, we can conclude that C implies (19).

$$\bigwedge_{1 \le j \le n} (\mathbf{G}(P \vee (Q_j \mathbf{W} l))) \tag{19}$$

The TR algorithm performs each BFS loop search iteration in a context that is separate from other BFS loop search iterations as well as from non-loop search. These contexts are essentially sets of SNF clauses, which we term *partitions*. Columns 3, 5, and 7 in Tab. 2 show the partitions that the premises of a production rule are taken from and the conclusion is put into. The main partition, M, contains C and the results of applying production rules init-ii, init-in, step-nn, step-nx, and step-xx. When a BFS loop search iteration is started, then first a new partition L is created. Production rules BFS-loop-it-init-x and BFS-loop-it-init-n copy all global clauses from M to L. Production rule BFS-loop-it-init-c creates the hypotheses for the loop search in L. Then production rule step-xx is applied inside L until no new clauses can be generated. Now production rule BFS-loop-it-sub comes into play. It does not produce a new clause but records the fact that the now part of some clause $(\mathbf{G}\ Q_j)$ obtained from production rule step-xx in L subsumes the $\mathbf{X}$ part of a clause $(\mathbf{GX}(Q_j \vee R_j \vee l))$ obtained from production rule BFS-loop-it-init-c in L. If rule BFS-loop-it-sub can be applied to every clause $(\mathbf{GX}(Q_j \vee R_j \vee l))$ obtained from production rule BFS-loop-it-init-c in L, then that BFS loop search iteration was successful. In that case the result of that BFS loop search iteration needs to be inserted into M. However, clearly (19) is not in SNF. Therefore, (19) is replaced with $\bigwedge_{1 \leq j \leq n}(\mathbf{G}(P \vee Q_j \vee l))$, $(\mathbf{G}(P \vee l \vee wl))$, $(\mathbf{G}((\neg wl) \vee \mathbf{X}(l \vee wl)))$, and $\bigwedge_{1 \leq j \leq n}(\mathbf{G}((\neg wl) \vee \mathbf{X}(Q_j \vee l)))$ in M, where wl is a fresh atomic proposition that is unique for each eventuality literal occurring in C. Notice that $(\mathbf{G}(P \vee l \vee wl))$ and $(\mathbf{G}((\neg wl) \vee \mathbf{X}(l \vee wl)))$ do not depend on the actual Q_j obtained in the BFS loop search iteration. These BFS loop search-independent clauses are therefore added to M before any BFS loop search is performed by the production rules listed under "augmentation". The remaining clauses are added to M by rules BFS-loop-conclusion1 and BFS-loop-conclusion2 when a successful BFS loop search iteration has been completed.

Phases of applying the rules listed under "saturation" in M and BFS loop search alternate until either $\Box$ has been derived in M (in which case C is unsatisfiable) or no new clauses can be generated any more (in which case C is satisfiable). For the exact sequencing of loop search and non-loop search phases of the TR algorithm we refer to the literature mentioned above.

We have not explained how to obtain the hypotheses for BFS loop search iterations. This is essentially a completeness issue and, therefore, not relevant for this paper. We only say here that hypotheses are either initialized to a default value or taken from a previous unsuccessful BFS loop search iteration for the same eventuality literal. This explains the entry L' for the partition of premise 1 of rule BFS-loop-it-init-c in Tab. 2.

The TR algorithm is a sound and complete decision procedure for the satisfiability of a set of SNF clauses [HK03, HK04, FDP01, Dix97, Dix98, Dix95]. We are not aware of a detailed complexity analysis of TR as implemented in TRP++; for complexity analyses of parts of the TR method relevant to the implementation in TRP++ see [FDP01, Dix98].

For example executions of the TR algorithm see the examples for constructing a resolution graph in Sec. 4.1.1.

4. UC Extraction via TR

In this section we describe our method from [Sch15] to construct UCs via TR. We present our method to obtain a UC in SNF via TR in Sec. 4.1. Then, in Sec. 4.2 we show how to map a UC in SNF back to LTL.

4.1. Extracting a UC in SNF

We now describe our method to extract a UC in SNF via TR from [Sch15]. Given an unsatisfiable set of SNF clauses C we would like to obtain a subset of C, C^{uc}, that is by itself unsatisfiable. The general idea of the method is unsurprising in that the production rules of the TR algorithm are used to construct a resolution graph whose nodes are labeled with SNF clauses and each of whose edges point from a vertex labeled with a premise to a vertex labeled with the conclusion of an application of a production rule. When the empty clause is derived, then the resolution graph is traversed backward from the empty clause to find the subset of C that was actually used to prove unsatisfiability, giving C^{uc} as desired.

4.1.1. Resolution Graphs

The presentation of a resolution graph in this paper makes a major change compared to [Sch15]. In [Sch15] a resolution graph is defined procedurally by constructing it during an execution of the TR algorithm. This requires the presentation of the TR algorithm at a fairly detailed level. In this paper we employ a definition of a resolution graph by the properties it exhibits. This allowed for the more abstract presentation of the TR algorithm in the previous section.

Another difference with respect to the presentation in [Sch15] is that in [Sch15] first an unoptimized and then an optimized resolution graph are presented. The optimized resolution graph drops some edges between selected premises and conclusions. The proofs that the resulting resolution graph and, therefore, also the resulting UCs still contain all the required information are fairly low level and mostly not required for the understanding of this paper. Therefore, here we present the optimized resolution graph right away (dropping "optimized") and simply refer to the results in [Sch15] where appropriate.

The definition of a resolution graph is stated in Def. 2 below. Essentially, it says that for each application of a production rule from Tab. 2 a new vertex is created, labeled with the SNF clause that is the conclusion of that production rule, and linked via incoming edges to the vertices that are labeled with the premises of that production rule. Notice that we repurpose the notion of partitions. In the previous section partitions grouped SNF clauses, while in Def. 2 they group vertices of a resolution graph (which of course are in turn labeled with SNF clauses). Moreover, we impose a total order on the vertices of a resolution graph. By forcing conclusions to be obtained only from premises that label smaller vertices and BFS loop search iterations to form a contiguous subsequence of vertices this allows to prevent circular reasoning.

Definition 2 (Resolution Graph)**.** *Let C be a set of SNF clauses. A* resolution graph *G for C is a directed graph consisting of (i) a totally ordered set of vertices V, (ii) a set of directed edges $E \subseteq V \times V$, (iii) a labeling of vertices with SNF clauses $L_V : V \to \mathbb{C}$, and (iv) a partitioning Q^V of the set of vertices V into one main partition M^V and a finite number of BFS loop search iteration partitions L^V_j: $Q^V : V = M^V \uplus L^V_0 \uplus \ldots \uplus L^V_n$,[4] fulfilling conditions 1–4 below.*

Let get_wl be a function that maps every eventuality literal l that occurs in C to a fresh atomic proposition wl: $get_wl : \{l \mid (\mathbf{G}(P \vee \mathbf{F}l)) \in C\} \mapsto \{wl \mid wl \in AP \text{ is fresh}\}$. Let get_l be a function that maps every BFS loop search iteration partition to an eventuality literal that occurs in C: $get_l : \{L^V_0, \ldots, L^V_n\} \mapsto \{l \mid (\mathbf{G}(P \vee \mathbf{F}l)) \in C\}$. Let $P \equiv \bigvee_{j=1\ldots n_p} p_j$, $Q \equiv \bigvee_{j=1\ldots n_q} q_j$, $R \equiv \bigvee_{j=1\ldots n_r} r_j$, $S \equiv \bigvee_{j=1\ldots n_s} s_j$.

1. *The vertices and edges obtained from conditions 3 and 4 below are the only vertices and edges of G.*
2. *In a partition vertex labels are unique: $\forall v, v' \in V \,.\, (v \neq v' \wedge ((v \in M^V \wedge v' \in M^V) \vee (\exists 0 \leq j \leq n \,.\, v \in L^V_j \wedge v' \in L^V_j))) \to L_V(v) \neq L_V(v')$.*
3. *Conditions for the main partition M^V:*
   (a) *Every SNF clause in C labels a vertex in M^V: $\forall c \in C \,.\, \exists v \in M^V \,.\, L_V(v) = c$. These vertices have no incoming edges: $\forall v \in M^V \,.\, L_V(v) \in C \to deg^-(v) = 0$.*[5]
   (b) *Every vertex v in M^V that is not labeled with an SNF clause from C is obtained from one of the following production rules:*
      - init-ii *v has two incoming edges from vertices $v' < v$, $v'' < v$ in M^V labeled with initial clauses $c' = (P \vee l)$ and $c'' = ((\neg l) \vee Q)$. v is labeled with an initial clause $c = (P \vee Q)$.*
      - init-in *v has two incoming edges from vertices $v' < v$, $v'' < v$ in M^V labeled with an initial clause $c' = (P \vee l)$ and a global clause with empty $\mathbf{X}$ part $c'' = (\mathbf{G}((\neg l) \vee Q))$. v is labeled with an initial clause $c = (P \vee Q)$.*
      - step-nn *v has two incoming edges from vertices $v' < v$, $v'' < v$ in M^V labeled with global clauses with empty $\mathbf{X}$ part $c' = (\mathbf{G}(P \vee l))$ and $c'' = (\mathbf{G}((\neg l) \vee Q))$. v is labeled with a global clause $c = (\mathbf{G}(P \vee Q))$.*
      - step-nx *v has two incoming edges from vertices $v' < v$, $v'' < v$ in M^V labeled with a global clause with empty $\mathbf{X}$ part $c' = (\mathbf{G}(P \vee l))$ and a global clause with non-empty $\mathbf{X}$ part $c'' = (\mathbf{G}(Q \vee \mathbf{X}((\neg l) \vee R)))$. v is labeled with a global clause $c = (\mathbf{G}(Q \vee \mathbf{X}(P \vee R)))$.*

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

[5] $deg^-(v)$ (resp. $deg^+(v)$) denotes the indegree (resp. outdegree) of a vertex v, i.e., its number of incoming (resp. outgoing) edges.

- step-xx *v has two incoming edges from vertices $v' < v$, $v'' < v$ in M^V labeled with global clauses with non-empty* **X** *part $c' = (\mathbf{G}(P \lor \mathbf{X}(Q \lor l)))$ and $c'' = (\mathbf{G}(R \lor \mathbf{X}((\neg l) \lor S)))$. v is labeled with a global clause $c = (\mathbf{G}(P \lor R \lor \mathbf{X}(Q \lor S)))$.*
- aug1 *v has an incoming edge from a vertex $v' < v$ in M^V labeled with an eventuality clause $c' = (\mathbf{G}(P \lor \mathbf{F}l))$. v is labeled with a global clause $c = (\mathbf{G}(P \lor l \lor wl))$ such that $wl = get_wl(l)$.*
- aug2 *There exists a vertex $v' < v$ in M^V labeled with an eventuality clause $c' = (\mathbf{G}(P \lor \mathbf{F}l))$. v is labeled with a global clause $c = (\mathbf{G}((\neg wl) \lor \mathbf{X}(l \lor wl)))$ such that $wl = get_wl(l)$.*
- BFS-loop-conclusion1 *v has an incoming edge from a vertex $v' < v$ in a successful BFS loop search iteration partition L_j^V labeled with a global clause with empty* **X** *part $c' = (\mathbf{G}\ P)$ and another incoming edge from a vertex $v'' < v$ in M^V labeled with an eventuality clause $c'' = (\mathbf{G}(Q \lor \mathbf{F}l))$ such that $l = get_l(L_j^V)$. v is labeled with a global clause $c = (\mathbf{G}(P \lor Q \lor l))$.*
- BFS-loop-conclusion2 *v has an incoming edge from a vertex $v' < v$ in a successful BFS loop search iteration partition L_j^V labeled with a global clause with empty* **X** *part $c' = (\mathbf{G}\ P)$. There exists a vertex $v'' < v$ in M^V labeled with an eventuality clause $c'' = (\mathbf{G}(Q \lor \mathbf{F}l))$ such that $l = get_l(L_j^V)$. v is labeled with a global clause $c = (\mathbf{G}((\neg wl) \lor \mathbf{X}(P \lor l)))$ such that $wl = get_wl(l)$.*

(c) *If a vertex v in the main partition M^V is labeled with the empty clause, then v has no outgoing edges: $\forall v \in M^V . L_V(v) = \Box \rightarrow deg^+(v) = 0$.*

4. *Conditions for a BFS loop search iteration partition L_j^V:*

   (a) *The vertices in L_j^V form a contiguous subsequence in the total order on V: $\forall v, v', v'' \in V . (v \leq v' \leq v'' \land v, v'' \in L_j^V) \rightarrow v' \in L_j^V$.*

   (b) *Every vertex v in L_j^V is obtained from one of the following production rules:*

   - BFS-loop-it-init-x *v has an incoming edge from a vertex $v' < v$ in M^V labeled with a global clause with non-empty* **X** *part $c' = (\mathbf{G}(P \lor \mathbf{X}Q))$. v is labeled with $c = c'$.*
   - BFS-loop-it-init-n *v has an incoming edge from a vertex $v' < v$ in M^V labeled with a global clause with empty* **X** *part $c' = (\mathbf{G}\ P)$. v is labeled with a global clause $c = (\mathbf{G}\,\mathbf{X}\,P)$.*
   - BFS-loop-it-init-c *Either $P \equiv \Box$ or there exists a vertex $v' < v$ in a different BFS loop search iteration partition $L_{j'}^V$ with $j' \neq j$, v' is labeled with a global clause with empty* **X** *part $c' = (\mathbf{G}\ P)$, and $get_l(L_{j'}^V) = get_l(L_j^V)$. v is labeled with a global clause $c = (\mathbf{GX}(P \lor l))$ such that $l = get_l(L_j^V)$.*
   - step-xx *v has two incoming edges from vertices $v' < v$, $v'' < v$ in L_j^V labeled with global clauses with non-empty* **X** *part $c' = (\mathbf{G}(P \lor \mathbf{X}(Q \lor l)))$ and $c'' = (\mathbf{G}(R \lor \mathbf{X}((\neg l) \lor S)))$. v is labeled with a global clause $c = (\mathbf{G}(P \lor R \lor \mathbf{X}(Q \lor S)))$.*

   (c) *A vertex v in L_j^V obtained from* BFS-loop-it-init-c *can have (at most) one incoming edge according to the following rule:*

   - BFS-loop-it-sub *v has an incoming edge from a vertex $v' > v$ in L_j^V labeled with a global clause with empty* **X** *part $c' = (\mathbf{G}\ P)$. v is labeled with a global clause with empty now part $c = (\mathbf{GX}(P \lor Q \lor l))$ such that $l = get_l(L_j^V)$. I.e., the now part of c' subsumes the* **X** *part of c.*

   (d) *L_j^V is successful iff every vertex v in L_j^V obtained from* BFS-loop-it-init-c *has an incoming edge obtained from* BFS-loop-it-sub*.*

In Lemma 1 we state that an optimized resolution graph according to [Sch15] is a resolution graph according to Def. 2. This is then used in Prop. 1 to establish that every unsatisfiable set of SNF clauses has a resolution graph with a vertex labeled with the empty clause in the main partition.

Lemma 1 (Optimized Resolution Graph in [Sch15] is Resolution Graph)**.** *An optimized resolution graph constructed according to Def. 2, 3, 7 in [Sch15] during an execution of the TR algorithm is a resolution graph.*

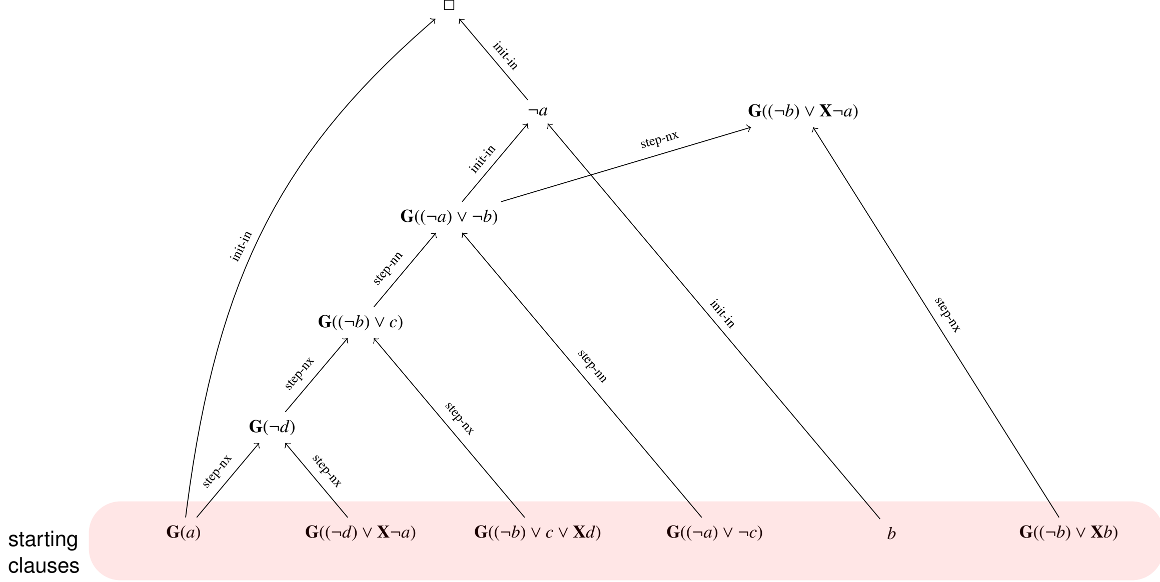

Figure 2: Example of a resolution graph without BFS loop search obtained from an execution of the TR algorithm.

Proof. (Idea.) By induction on the sequence of vertices in which the optimized resolution graph is constructed in Def. 2, 3, 7 in [Sch15] during an execution of the TR algorithm. □

Proposition 1 (Unsatisfiable Set of SNF Clauses has Resolution Graph with Empty Clause in Main Partition)**.** *Let C be an unsatisfiable set of SNF clauses. Then there exists a resolution graph G with set of vertices V, vertex labeling L_V, and main partition M^V such that M^V contains a vertex v that is labeled with the empty clause* $\Box$*:* $\exists v \in M^V \, . \, L_V(v) = \Box$.

Proof. This follows directly from the completeness of the TR algorithm, the construction of an optimized resolution graph in Def. 2, 3, 7 in [Sch15] during an execution of the TR algorithm, and Lemma 1. □

Running Example 2. In Fig. 2 we show an example of a resolution graph obtained from an execution of the TR algorithm that did not require BFS loop search. This and the following example will later be continued in Sec. 4.1.2 to illustrate the extraction of a UC in SNF and in Sec. 6 to illustrate the computation of sets of time points for a UC in SNF. The set of SNF clauses C to be solved contains b, $\mathbf{G}((\neg b) \vee c \vee \mathbf{X}d)$, $\mathbf{G}((\neg a) \vee \neg c)$, $\mathbf{G}((\neg d) \vee \mathbf{X}\neg a)$, $\mathbf{G}(a)$, and $\mathbf{G}((\neg b) \vee \mathbf{X}b)$. The first four clauses b, $\mathbf{G}((\neg b) \vee c \vee \mathbf{X}d)$, $\mathbf{G}((\neg a) \vee \neg c)$, $\mathbf{G}((\neg d) \vee \mathbf{X}\neg a)$ force a to be FALSE at time point 0 or 2. This is contradicted by the fifth clause $\mathbf{G}(a)$. Clearly, C is unsatisfiable. Notice that the sixth clause $\mathbf{G}((\neg b) \vee \mathbf{X}b)$ is not required for unsatisfiability; it is present in C to demonstrate later that SNF clauses not used in a proof of unsatisfiability are removed by our method of UC extraction.

In the following description we identify vertices with the SNF clauses they are labeled with. SNF clauses are connected with edges according to Def. 2, with corresponding labels on the edges. In Fig. 2 the TR algorithm proceeds from bottom to top. In the first row from the bottom (in the light red shaded rectangle) are the starting clauses from C. In the top row is only the empty clause □, which signals unsatisfiability of C. All SNF clauses not in C are derived from production rules listed under "saturation" in Tab. 2. For example, $\mathbf{G}(\neg d)$ in row 2 is obtained using production rule step-nx with $\mathbf{G}(a)$ in row 1 as premise 1 and $\mathbf{G}((\neg d) \vee \mathbf{X}\neg a)$ in row 1 as premise 2. In this example the main partition is the only partition, containing all SNF clauses. □

Running Example 3. In Fig. 3 we show a more complex example of a resolution graph obtained from an execution of the TR algorithm that also includes BFS loop search. 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 $\mathbf{G}((\neg b) \vee \mathbf{X}a)$

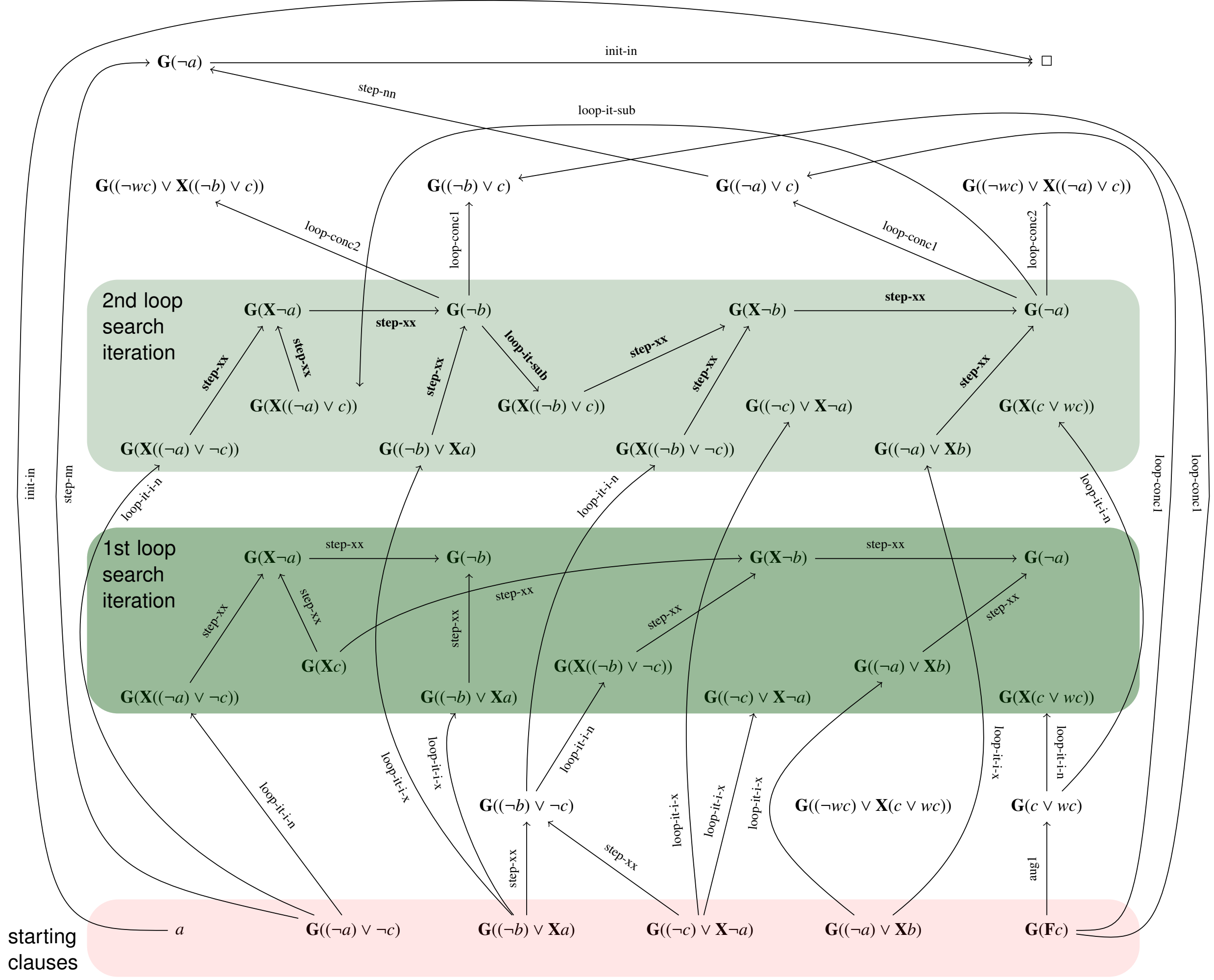

Figure 3: Example of a resolution graph with BFS loop search obtained from an execution of the TR algorithm.

force a to be TRUE 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 FALSE for two consecutive time points. Clearly, C is unsatisfiable. This example is based on the same idea as (4) in Sec. 2. However, the SNF obtained by our translation from LTL into SNF for (4) is larger than C, with the corresponding figure harder to fit on one page.

In the following description we proceed along the lines of the execution of the TR algorithm. "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 □ signaling unsatisfiability of C. The main partition contains the starting clauses from C and all SNF clauses not contained in either of the two dark and light green shaded rectangles. Each of the two dark and light green shaded rectangles represents a BFS loop search iteration partition.

Row 2 contains the SNF clauses that result from applying the rules listed under "saturation" and "augmentation" in Tab. 2 to C. In accordance with Def. 2 clause $\mathbf{G}((\neg wc) \vee \mathbf{X}(c \vee wc))$, which was obtained from clause $\mathbf{G}(\mathbf{F}c)$ by applying production rule aug2, has no incoming edge from its premise.

The dark green shaded rectangle is the partition for the first iteration of a BFS loop search for a loop in $\neg c$. Row 3 contains the result of BFS loop search initialization with production rules BFS-loop-it-init-x, BFS-loop-it-init-n, and BFS-loop-it-init-c. Clause $\mathbf{G}(\mathbf{X}c)$ was obtained by an application of production rule BFS-loop-it-init-c and, hence, has no

incoming edges from its premises. Row 4 then shows the SNF clauses obtained by applying production rule [step-xx] to the SNF clauses in this BFS loop search iteration partition. As none of the SNF clauses in row 4 subsumes $\Box$ (notice that $\mathbf{G}(\mathbf{X}c)$ can be thought of as $\mathbf{G}(\mathbf{X}(\text{FALSE} \vee c))$), this iteration terminates without having found a loop.

The second BFS loop search iteration partition is in the light green shaded rectangle. Again, row 5 contains the result of BFS loop search initialization and row 6 the SNF clauses obtained by applying production rule [step-xx]. Now there are two SNF clauses obtained from production rule [BFS-loop-it-init-c]: $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$ and $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$. This time, there are global clauses $\mathbf{G}(\neg a)$ and $\mathbf{G}(\neg b)$ in row 6 subsuming $\mathbf{G}(\mathbf{X}((\neg a) \vee c))$ and $\mathbf{G}(\mathbf{X}((\neg b) \vee c))$ in row 5 according to production rule [BFS-loop-it-sub]; hence, this BFS loop search iteration is successful.

In row 7 we are back in the main partition and show the conclusions obtained by applying production rules [BFS-loop-conclusion1] and [BFS-loop-conclusion2] to the just completed, successful BFS loop search iteration. Notice that in accordance with Def. 2 both clauses $\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg b) \vee c))$ and $\mathbf{G}((\neg wc) \vee \mathbf{X}((\neg a) \vee c))$ obtained from production rule [BFS-loop-conclusion2] have no incoming edge from their premise 2, $\mathbf{G}(\mathbf{F}c)$. The last row finally contains the derivation of $\Box$.

As a final note we remark that, while it may seem that some SNF clauses are not considered for BFS loop search initialization or during an application of the production rules listed under "saturation" in Tab. 2, this is due to either subsumption of one SNF 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. $\Box$

4.1.2. Extracting a UC in SNF from a Resolution Graph

Proposition 1 above allows, after a standard definition of a UC in SNF in Def. 3, to state the definition of UC extraction in SNF via TR in Def. 4. Theorem 1 then shows the correctness of UC extraction in SNF via TR.

Definition 3 (UC in SNF)**.** *Let C be an unsatisfiable set of SNF clauses. Let C^{uc} be an unsatisfiable subset of C. Then C^{uc} is a* UC of C in SNF.

Definition 4 (UC in SNF via TR)**.** *Let C be an unsatisfiable set of SNF clauses, let G be a resolution graph with $v_\Box$ the (unique) vertex in the main partition M^V of the resolution graph G labeled with the empty clause $\Box$. Let G' be the smallest subgraph of G that contains $v_\Box$ and all vertices in G (and the corresponding edges) that are backward reachable from $v_\Box$. The* UC of C in SNF via TR, *C^{uc}, is the subset of C such that there exists a vertex v in the subgraph G', labeled with $c \in C$, and contained in the main partition M^V of G: $C^{uc} = \{c \in C \mid \exists v \in V_{G'} \, . \, L_V(v) = c \wedge v \in M^V\}$.*

Theorem 1 (Unsatisfiability of UC in SNF via TR)**.** *Let C be an unsatisfiable set of SNF clauses, and let C^{uc} be a UC of C in SNF via TR. Then C^{uc} is unsatisfiable.*

Proof. (Sketch.) First, notice that in Def. 2 production rules [aug2], [BFS-loop-it-init-c], and [BFS-loop-conclusion2] have an eventuality clause as a premise but do not include an edge from that premise to the conclusion in the resolution graph. Hence, we need to show that despite the absence of corresponding edges in the resolution graph these premises are still included in a UC in SNF via TR. This can be proved as in Lemmas 2–4 in [Sch15].

Next we have to show for each of the production rules in Def. 2 that the conclusion (i) constitutes a correct inference ([init-ii], [init-in], [step-nn], [step-nx], [step-xx], [BFS-loop-conclusion1], [BFS-loop-conclusion2]), (ii) can be added to C without removing any satisfying assignments ([aug1], [aug2]), (iii) propagates information correctly from the main partition to a BFS loop search iteration partition ([BFS-loop-it-init-x], [BFS-loop-it-init-n], [BFS-loop-it-init-c]), or (iv) correctly records a subsumption relation ([BFS-loop-it-sub]). For production rules [init-ii], [init-in], [step-nn], [step-nx], and [step-xx] it is easy to see that in each case the conclusion follows from the premises. For production rules [aug1] and [aug2] one can show that adding the conclusions of these production rules to C preserves satisfiability of C [FDP01]. Production rules [BFS-loop-it-init-x] and [BFS-loop-it-init-n] clearly propagate information correctly from the main partition to a BFS loop search iteration partition. By construction production rule [BFS-loop-it-init-c] creates an SNF clause that contains the eventuality literal of some eventuality clause in C, and production rule [BFS-loop-it-sub] links two vertices such that the now part of the SNF clause labeling the source vertex implies the $\mathbf{X}$ part of the SNF clause labeling the target vertex. It is left to show that the conclusions of production rules [BFS-loop-conclusion1] and [BFS-loop-conclusion2] indeed follow from C^{uc}. This can be proved as in Lemma 5 in [Sch15].

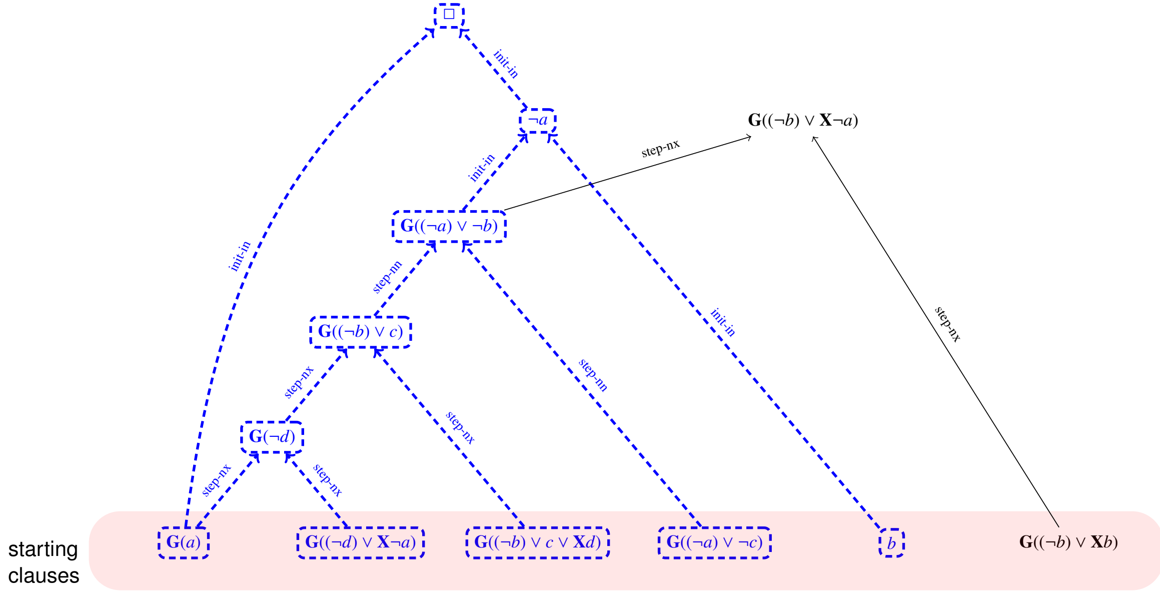

Figure 4: Example of extracting a UC in SNF from a resolution graph without BFS loop search.

Finally, remember that the main partition of the resolution graph contains a vertex $v_\square$ labeled with the empty clause □. By the above reasoning the empty clause □ follows from the UC in SNF via TR, C^{uc} and, hence, C^{uc} is unsatisfiable. □

As C^{uc} is a subset of C, we have the following corollary.

Corollary 1 (UC in SNF via TR is UC in SNF)**.** *Let C be an unsatisfiable set of SNF clauses, and let C^{uc} be a UC of C in SNF via TR. Then C^{uc} is a UC of C in SNF.*

Lemma 1 allows to construct a resolution graph for an unsatisfiable set of SNF clauses as a byproduct of an execution of the TR algorithm as in [Sch15]. As shown in [Sch15] this establishes a bound on the effort that is induced by our method for UC extraction in addition to the effort required by an execution of the TR algorithm that is linear in the effort required by an execution of the TR algorithm.

Proposition 2 (Added Complexity of UC Extraction)**.** *Let C be an unsatisfiable set of SNF clauses. Construction of C^{uc} according to Def. 4 can be performed in time exponential in $|AP| + log(|C|)$ in addition to the time required to run the TR algorithm.*

Running Example 2 (continuing from p. 15)**.** In Fig. 4 we continue the running example from Sec. 4.1.1 and show how to extract a UC in SNF from a resolution graph. That UC in SNF will later be enriched with sets of time points in Sec. 6, as will be the UC in the following example. The SNF clauses that are backward reachable from □ in the main partition are shown in blue with blue, thick, dashed boxes. The corresponding edges are also blue, thick, and dashed. The resulting UC comprises all SNF clauses in C except for **G**$((\neg b) \vee$ **X**$b)$, which is not backward reachable from the empty clause □. □

Running Example 3 (continuing from p. 15)**.** In Fig. 5 we continue the more complex running example from Sec. 4.1.1. SNF clauses and edges that are backward reachable from □ are highlighted in the same way as in the previous example. The resulting UC comprises all SNF clauses in C (note that this example shows the mechanism rather than the benefits of extracting UCs). □

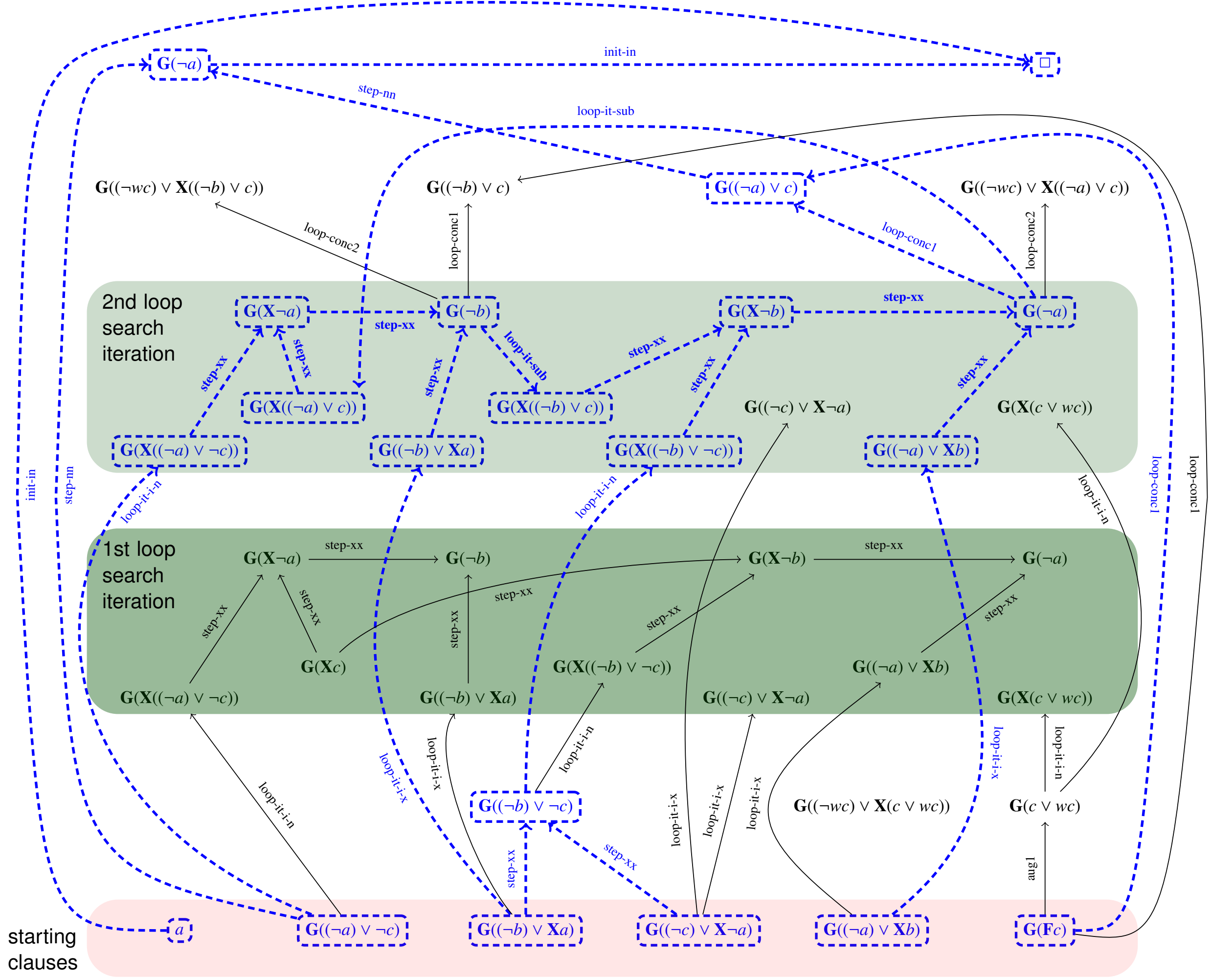

Figure 5: Example of extracting a UC in SNF from a resolution graph with BFS loop search.

4.2. Extracting a UC in LTL

A potential disadvantage of using TR for extracting UCs for LTL is the fact that TR does not work directly on LTL but on the clausal normal form SNF. Translating an LTL formula into an equisatisfiable formula in SNF is straightforward (see Sec. 3.4). However, for optimal support of a user who tries to track down the source of the unsatisfiability of a formula it is likely to be helpful if the UC that is presented to her is "syntactically close" to the formula that she provided as an input to the solver. Hence, it is necessary to map a UC obtained in SNF back to LTL.

In this subsection we restate our method from [Sch15] to map a UC in SNF back to a UC in LTL. The notion of UC in LTL that we use is one where occurrences of subformulas in the input formula provided by the user are replaced with TRUE or FALSE depending on polarity. The resulting UC in LTL is "syntactically close" to the input formula in two respects: (i) Syntactic structure: Seen from the root node of the syntax tree of the input formula the higher level syntactic structure of the syntax tree remains unchanged. (ii) Set of atomic propositions: While the translation from LTL into SNF in Def. 1 introduces fresh atomic propositions, which may appear in the UC in SNF, these fresh atomic propositions are not used in the UC in LTL. I.e., the UC in LTL only contains atomic propositions that are also present in the input formula.

Definition 5 provides a straightforward definition of a UC in LTL. Definition 6 then describes the mapping from a UC in SNF to a UC in LTL. An occurrence of a subformula ψ is *not* replaced with TRUE or FALSE in the UC in LTL if

the UC in SNF contains an SNF clause c such that x_ψ occurs in c in a position that is marked blue boxed in Tab. 1. The correctness of the construction is stated in Thm. 2. The main idea in the proof is to compare the SNF of ϕ and of its UC in LTL by partitioning the SNF clauses into three sets: one that is shared by the two SNFs, one that replaces some occurrences of propositions in $SNF(\phi)$ with TRUE or FALSE, and one whose SNF clauses are only in $SNF(\phi)$. Then one can show that the UC of ϕ in SNF must be contained in the first partition. For a formal proof see [Sch15].

Definition 5 (UC in LTL)**.** *(cf. Def. 10 of [Sch12b]) Let ϕ be an unsatisfiable LTL formula. Let ϕ^{uc} (i) be obtained from ϕ by replacing a set of positive polarity occurrences of subformulas of ϕ with* TRUE *and a set of negative polarity occurrences of subformulas of ϕ with* FALSE *and (ii) be unsatisfiable. Then ϕ^{uc} is a* UC of ϕ in LTL.

Definition 6 (UC in LTL from SNF)**.** *Let ϕ be an unsatisfiable LTL formula, let $SNF(\phi)$ be its SNF, and let C^{uc} be a UC of $SNF(\phi)$ in SNF. Then the* UC of ϕ in LTL from SNF*, 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. 1, replace ψ in ϕ with* TRUE *(resp.* FALSE*) iff C^{uc} contains no SNF clause with an occurrence of proposition x_ψ that is marked blue boxed in Tab. 1. (We are sloppy in that we "replace" subformulas of replaced subformulas, while in effect they simply vanish.)*

Theorem 2 (Unsatisfiability of UC in LTL from SNF)**.** *Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be a UC of ϕ in LTL from SNF. Then ϕ^{uc} is unsatisfiable.*

As a UC in LTL from SNF fulfills requirement (i) in Def. 5, we obtain the following corollary.

Corollary 2 (UC in LTL from SNF is UC in LTL)**.** *Let ϕ be an unsatisfiable LTL formula, and let ϕ^{uc} be a UC of ϕ in LTL from SNF. Then ϕ^{uc} is a UC of ϕ in LTL.*

Running Example 1 (continuing from p. 10)**.** We now continue the running example from Sec. 3.4 and show how to map a UC in SNF to a UC in LTL. The UC of (15) in SNF is shown in (20).

$$\begin{array}{l}\{(x_\phi),\\ (\mathbf{G}(x_\phi \to x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q})), (\mathbf{G}(x_\phi \to x_{p\mathbf{U}(q\wedge r)})),\\ (\mathbf{G}(x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q} \to x_{\mathbf{X}\neg p})), (\mathbf{G}(x_{(\mathbf{X}\neg p)\wedge\mathbf{G}\neg q} \to x_{\mathbf{G}\neg q})),\\ (\mathbf{G}(x_{\mathbf{X}\neg p} \to \mathbf{X}x_{\neg p})),\\ (\mathbf{G}(x_{\neg p} \to \neg p)),\\ (\mathbf{G}(x_{\mathbf{G}\neg q} \to \mathbf{X}x_{\mathbf{G}\neg q})), (\mathbf{G}(x_{\mathbf{G}\neg q} \to x_{\neg q})),\\ (\mathbf{G}(x_{\neg q} \to \neg q)),\\ (\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \to (x_{q\wedge r} \vee p))), (\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \to (x_{q\wedge r} \vee \mathbf{X}x_{p\mathbf{U}(q\wedge r)}))),\\ (\mathbf{G}(x_{q\wedge r} \to q))\}\end{array} \tag{20}$$

Two SNF clauses are removed from (15) in (20): $(\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \to \mathbf{F}x_{q\wedge r}))$ and $(\mathbf{G}(x_{q\wedge r} \to r))$. $x_{q\wedge r}$ and r occur in the removed SNF clauses in positions that are marked blue boxed in Tab. 1. However, $x_{q\wedge r}$ also occurs in a position that is marked blue boxed in Tab. 1 in clause $(\mathbf{G}(x_{p\mathbf{U}(q\wedge r)} \to (x_{q\wedge r} \vee p)))$, which is part of the UC in SNF in (20). Hence, according to Def. 6 r is the only occurrence of a subformula that can be removed in (14) to obtain a UC in LTL. The occurrence of r has positive polarity in (14) and, therefore, this occurrence is replaced with TRUE leading to the UC of (14) in LTL shown in (21).[6]

$$\phi^{uc} = ((\mathbf{X}\neg p) \wedge \mathbf{G}\neg q) \wedge (p\mathbf{U}(q \wedge \text{TRUE})) \tag{21}$$

This example will be continued Sec. 6.3 to illustrate mapping a UC in SNF with sets of time points to a UC in LTL with sets of time points. □

[6] Note that there is an alternative UC of (14) that uses only the second and third conjunct: $(\text{TRUE} \wedge \mathbf{G}\neg q) \wedge (\text{TRUE}\,\mathbf{U}(q \wedge \text{TRUE}))$. The TR algorithm exhaustively applies the production rules listed under "saturation" in Tab. 2 and stops when the empty clause is derived before the first BFS loop search is started, which prevents that alternative UC from being found. If BFS loop search were initiated at an arbitrary point during a possibly incomplete round of saturation, then also that alternative UC could be found using our method.

Table 3: Semantics of LTLp.

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

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 TRUE or FALSE depending on polarity. Note that this can be seen as an extension of a notion of UC in [Sch12b], where subformulas are replaced with TRUE or FALSE 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. For examples of LTLp formulas see Sec. 2.

Definition 7 (LTLp Syntax)**.** *The set of* LTLp formulas *is constructed inductively as follows. The Boolean constants* FALSE, TRUE $\in \mathbb{B}$ *and any atomic proposition* $p \in AP$ *are LTLp formulas. If* $I, I' \subseteq \mathbb{N}$ *are sets of time points and if* τ, τ' *are LTLp 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 8 (LTLp Semantics)**.** *Let* π *be a word in* $(2^{AP})^{\omega}$*, and let* i *be a time point in* $\mathbb{N}$*. The semantics of LTLp is given in Tab. 3.* 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. 8.

In Prop. 3–6 we state some properties of LTLp. The first three propositions link LTLp to methods for obtaining UCs from sets of elements by removing elements from those sets (e.g., [BDTW93, CD91, GN03, Zha03, Sch12b]) as follows. Proposition 3 allows to turn an LTL formula ϕ into an equivalent LTLp formula by simply adding $\mathbb{N}$ as set(s) of time points to all operators of ϕ. Then Prop. 4 shows that removing elements from a set of time points syntactically weakens an LTLp formula. Finally, Prop. 5 establishes that UCs in LTL that are obtained by replacing occurrences of

subformulas with TRUE or FALSE depending on polarity [Sch12b] are limit cases of UCs in LTLp that are obtained by removing elements from sets of time points. Below let *LTLp2LTL* denote the function that takes an LTLp formula θ and returns an LTL formula ϕ by removing all sets of time points.

Proposition 3 (LTLp with all sets of time points $\mathbb{N}$ is LTL)**.** *Let θ be an LTLp formula such that all sets of time points are $\mathbb{N}$, and let $\phi \equiv$ LTLp2LTL(θ). 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. 3 is a left operand of a disjunction that, with $i \notin \mathbb{N}$ being FALSE, evaluates to its right operand. The latter evaluates to the corresponding LTL operand by inductive assumption. The case for inclusion is analogous. □

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

Proposition 4 (Enlarging sets of time points strengthens positive and weakens negative polarity subformulas)**.** *Let θ be an LTLp formula, let χ be such a modification of θ such that LTLp2LTL(θ) = LTLp2LTL(χ) and all sets of time points in χ are (possibly non-strict) supersets of those in θ. Then $\chi \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \theta$ is valid.*

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 LTLp 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. 8 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 $\underset{I'}{\neg}\tau'$. □

Proposition 5 (An LTLp operator with sets of time points $\emptyset$ is equivalent to TRUE/FALSE)**.** *Let θ be an LTLp 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 TRUE (resp. FALSE) are equivalent.*

Proof. Directly from Def. 8: if τ has positive polarity, then the tests for non-inclusion (such as $i \notin I$) in Tab. 3 are left operands of disjunctions and evaluate to TRUE; the case for negative polarity is analogous. □

In Prop. 7 in Sec. 6 we show that the sets of time points that we obtain for UCs with our method are semilinear. Remember that LTL cannot count and, in particular, LTL cannot express the property that some atomic proposition p is TRUE at every even time point (but leaving it open what should happen at odd time points) (see, e.g., [Wol83]). It is easy to see that the LTLp formula $\underset{2\cdot\mathbb{N}}{\mathbf{G}}\, p$ expresses precisely that property. Hence, LTLp with semilinear sets of time points is strictly more expressive than LTL. In Prop. 6 below we establish an upper bound on the expressiveness of LTLp with semilinear sets of time points. We show that an LTLp formula with semilinear sets of time points can be expressed in EQLTL.

Proposition 6. *LTLp with sets of time points restricted to semilinear sets is no more expressive than EQLTL.*

Proof. Let θ be an LTLp formula with sets of time points $I_1, \ldots, I_n$. Each set of time points I_j in θ can be written as $\bigcup_{1 \leq j' \leq n_j} p_{j,j'} \cdot \mathbb{N} + o_{j,j'}$ for some $n_j \in \mathbb{N}, p_{j,1} \ldots, p_{j,n_j}, o_{j,1}, \ldots, o_{j,n_j} \in \mathbb{N}$. For $m \in \mathbb{N}$ let $\mathbf{X}^m\psi$ abbreviate $\underbrace{\mathbf{X} \ldots \mathbf{X}}_{m}\psi$.

Construct a EQLTL formula θ'' as follows:

1. For each set of time points I_j introduce $n_j + 1$ fresh Boolean propositions $q_j, q_{j,1}, \ldots, q_{j,n_j}$.
2. Let $\circ_1 \in \{\neg, \mathbf{X}, \mathbf{F}, \mathbf{G}\}$ and $\circ_2 \in \{\vee, \wedge, \mathbf{U}, \mathbf{R}\}$. Construct θ' from θ by replacing each
   (a) positive polarity occurrence of $\underset{I_j}{\circ_1}\tau$ in θ with $\circ_1(q_j \rightarrow \tau)$,

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

(c) positive polarity occurrence of $\tau \underset{I_j,I_{j'}}{\circ_2} \tau'$ in θ with $(q_j \rightarrow \tau) \circ_2 (q_{j'} \rightarrow \tau')$, and

(d) negative polarity occurrence of $\tau \underset{I_j,I_{j'}}{\circ_2} \tau'$ in θ with $(q_j \wedge \tau) \circ_2 (q_{j'} \wedge \tau')$.

3. Define θ'' as

$$
\begin{array}{c}
\exists q_{1,1} \,.\, \cdots \exists q_{1,n_1} \,.\, \exists q_1 \,. \\
\vdots \\
\exists q_{n,1} \,.\, \cdots \exists q_{n,n_n} \,.\, \exists q_n \,.\, (\\
(\bigwedge_{0 \leq j < o_{1,1}} \mathbf{X}^j \neg q_{1,1}) \wedge (\mathbf{X}^{o_{1,1}}(q_{1,1} \wedge \mathbf{G}(q_{1,1} \rightarrow ((\bigwedge_{1 \leq j < p_{1,1}} \mathbf{X}^j \neg q_{1,1}) \wedge (\mathbf{X}^{p_{1,1}} q_{1,1}))))) \wedge \\
\cdots \\
(\bigwedge_{0 \leq j < o_{1,n_1}} \mathbf{X}^j \neg q_{1,n_1}) \wedge (\mathbf{X}^{o_{1,n_1}}(q_{1,n_1} \wedge \mathbf{G}(q_{1,n_1} \rightarrow ((\bigwedge_{1 \leq j < p_{1,n_1}} \mathbf{X}^j \neg q_{1,n_1}) \wedge (\mathbf{X}^{p_{1,n_1}} q_{1,n_1}))))) \wedge \\
(\mathbf{G}(q_1 \leftrightarrow \bigvee_{1 \leq j \leq n_1} q_{1,j})) \wedge \\
\vdots \\
(\bigwedge_{0 \leq j < o_{n,1}} \mathbf{X}^j \neg q_{n,1}) \wedge (\mathbf{X}^{o_{n,1}}(q_{n,1} \wedge \mathbf{G}(q_{n,1} \rightarrow ((\bigwedge_{1 \leq j < p_{n,1}} \mathbf{X}^j \neg q_{n,1}) \wedge (\mathbf{X}^{p_{n,1}} q_{n,1}))))) \wedge \\
\cdots \\
(\bigwedge_{0 \leq j < o_{n,n_n}} \mathbf{X}^j \neg q_{n,n_n}) \wedge (\mathbf{X}^{o_{n,n_n}}(q_{n,n_n} \wedge \mathbf{G}(q_{n,n_n} \rightarrow ((\bigwedge_{1 \leq j < p_{n,n_n}} \mathbf{X}^j \neg q_{n,n_n}) \wedge (\mathbf{X}^{p_{n,n_n}} q_{n,n_n}))))) \wedge \\
(\mathbf{G}(q_n \leftrightarrow \bigvee_{1 \leq j \leq n_n} q_{n,j})) \wedge \\
\theta')
\end{array}
$$

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

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. This is the main contribution of this paper. We start by providing an intuition using one of our running examples in Sec. 6.1 before the formal exposition in Sec. 6.2 and 6.3.

6.1. Intuition

Running Example 2 (continuing from p. 18)**.** We now continue the running example from Sec. 4.1.2 in Fig. 6 to provide some intuition on how the information is obtained at which time points the SNF clauses that occur in a TR proof of unsatisfiability are relevant.

Consider the empty clause □ in the top row of Fig. 6, which signals unsatisfiability of the set of starting clauses C. It is obtained by an application of production rule [init-in]. Hence, in this case □ is an initial clause. This fact implies that the contradiction in the TR proof, which the resolution graph in Fig. 6 corresponds to, happens at time point 0. Therefore, □ is relevant in the TR proof at time point 0. This is indicated by labeling □ in Fig. 6 with {0} shown in a black box. Starting at □ we trace that information back through the resolution graph to obtain the sets of time points at which the other SNF clauses that participate in the proof of unsatisfiability are relevant. Notice that below we can restrict ourselves to the part of the resolution graph that is backward reachable from □.

Production rule [init-in] that □ was obtained from takes an initial clause and a global clause with empty $\mathbf{X}$ part as premises and resolves them to obtain the conclusion, which is also an initial clause. Production rule [init-in] can be interpreted as performing propositional resolution between its premises (the global clause being stripped of its $\mathbf{G}$ operator) at a single time point, namely, time point 0. Therefore, each of its premises is relevant in the TR proof at time point 0. This is indicated in Fig. 6 by labeling $\mathbf{G}(a)$ in row 1 and $\neg a$ in row 5 with sets of time point that contain 0 (the fact that the set of time points labeling $\mathbf{G}(a)$ also contains 2 will be explained later).

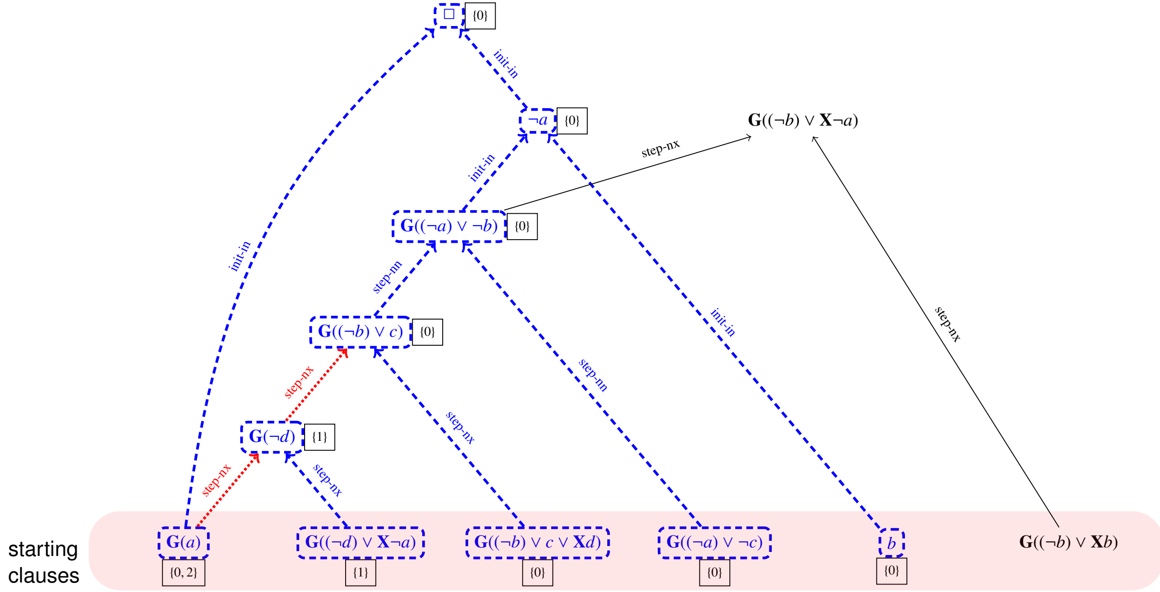

Figure 6: Example of computing sets of time points for a UC in SNF without BFS loop search.

As $\neg a$ in row 5 is also obtained from an instance of production rule init-in, its premises $\mathbf{G}((\neg a) \lor \neg b)$ in row 4 and b in row 1 are relevant in the TR proof at time point 0, indicated by the corresponding labels with $\{0\}$ in Fig. 6.

$\mathbf{G}((\neg a) \lor \neg b)$ in row 4 is produced by applying production rule step-nn to $\mathbf{G}((\neg b) \lor c)$ in row 3 and $\mathbf{G}((\neg a) \lor \neg c)$ in row 1. Production rule step-nn takes two global clauses with empty $\mathbf{X}$ part and produces a global clause with empty $\mathbf{X}$ part. It can be interpreted as performing propositional resolution between its premises (stripped of their $\mathbf{G}$ operators) at all time points in $\mathbb{N}$. Now assume that we know that the conclusion is relevant in the TR proof at some set of time points $I \subseteq \mathbb{N}$. Then we can infer that also both premises are relevant in the TR proof at least at the time points in I. Hence, the sets of time points labeling the two premises must contain I. In Fig. 6 the two premises $\mathbf{G}((\neg b) \lor c)$ in row 3 and $\mathbf{G}((\neg a) \lor \neg c)$ in row 1 are therefore labeled with $\{0\}$.

The next case is slightly more complicated. $\mathbf{G}((\neg b) \lor c)$ is obtained from an instance of production rule step-nx with premises $\mathbf{G}(\neg d)$ in row 2 and $\mathbf{G}((\neg b) \lor c \lor \mathbf{X}d)$ in row 1. Notice that the occurrence of $\neg d$ in the now part of $\mathbf{G}(\neg d)$ is resolved with the occurrence of d in the $\mathbf{X}$ part of $\mathbf{G}((\neg b) \lor c \lor \mathbf{X}d)$. In general, production rule step-nx takes a global clause with empty $\mathbf{X}$ part as premise 1 and a global clause with non-empty $\mathbf{X}$ part as premise 2 and resolves the now part of premise 1 with the $\mathbf{X}$ part of premise 2. Hence, this can be interpreted as first "time-shifting" premise 1 one step into the future and then performing propositional resolution between the modified premises at all time points in $\mathbb{N}$. If we know that the conclusion is relevant in the TR proof at some set of time points $I \subseteq \mathbb{N}$, then also premise 2 is relevant in the TR proof at least at the time points in I. For premise 1 we have to take the time-shift into account. Therefore, premise 1 is relevant in the TR proof at least at the time points in I shifted one step into the future, i.e., at the time points in $1 + I$. This leads to labels $1 + \{0\} = \{1\}$ for $\mathbf{G}(\neg d)$ in row 2 and $\{0\}$ for $\mathbf{G}((\neg b) \lor c \lor \mathbf{X}d)$ in row 1 in Fig. 6. The fact that the edge between premise 1 and the conclusion of production rule step-nx involves a time-shift is shown in Fig. 6 by marking this edge red, dotted.

Finally, $\mathbf{G}(\neg d)$ in row 2 is obtained from another instance of production rule step-nx in a similar fashion as in the previous case from $\mathbf{G}(a)$ and $\mathbf{G}((\neg d) \lor \mathbf{X}\neg a)$ in row 1. Premise $\mathbf{G}((\neg d) \lor \mathbf{X}\neg a)$ in row 1 is not the time-shifted premise and, therefore, labeled with the same set of time points as the conclusion, namely, $\{1\}$. Premise $\mathbf{G}(a)$ in row 1 is time-shifted in this production. Hence, because $\mathbf{G}(\neg d)$ in row 2 is relevant in the TR proof at time points $\{1\}$ in the TR proof, we know that $\mathbf{G}(a)$ in row 1 is relevant in the TR proof at least at time points $1 + \{1\} = \{2\}$. However, we know from before that $\mathbf{G}(a)$ in row 1 is also relevant in the TR proof at least at time points $\{0\}$. Therefore, we form the union of both sets of time points and obtain $\{0\} \cup \{2\} = \{0, 2\}$ as the set of time points at which $\mathbf{G}(a)$ in row 1 is relevant in the TR proof.

Now we have obtained for each clause in C^{uc} a set of time points at which it is relevant in the TR proof. In remains to use this information to produce a UC in SNF with sets of time points. This is done by a straightforward definition of what it means to label an SNF clause with sets of time points in Def. 10 and a corresponding definition of a UC in SNF with sets of time points in Def. 11.

In the description above we pushed the information on when clauses are relevant in the TR proof backwards from the empty clause $\Box$ towards the starting clauses C step by step until a fixed point was reached. This worked because the resolution graph in Fig. 6 is acyclic. However, as evidenced in the more complex running example in Fig. 5, resolution graphs may indeed be cyclic. Therefore, we need to extend our approach to actually compute sets of time points to handle cycles in resolution graphs.

Assume there is a path π from some starting clause $c \in C$ to $\Box$ in the resolution graph such that π traverses i edges that involve a time-shift. The set of time points I labeling $\Box$ is "pushed back" to c backwards on π such that each edge that involves a time-shift adds 1 to the current set of time points. I.e., c receives via π the set of time points $i + \{0\} = \{i\}$. To obtain the set of all time points at which c is relevant in the TR proof one has to take all paths from c to $\Box$ into account. Hence, c is relevant at time point i in the TR proof iff there exists a path from c to $\Box$ that traverses i edges that involve a time-shift. For example, in Fig. 6 $\mathbf{G}(a)$ in row 1 has two paths to $\Box$, one of which traverses 0, the other one 2 edges that involve a time-shift. This matches the set of time points $\{0, 2\}$ that is assigned to $\mathbf{G}(a)$ in row 1.

Now we are left with the problem to obtain the set of all numbers of edges that involve a time-shift on some path from c to $\Box$. Fortunately, it turns out that this is a well known problem in formal languages. If the resolution graph is seen as a transition-labeled nondeterministic finite automaton (NFA) over the alphabet 0 (no time-shift; blue, dashed edges in Fig. 6) and 1 (time-shift; red, dotted edges in Fig. 6) with initial state c and final state $\Box$, then the desired information for c is just the Parikh image [Par66] of the letter 1 in the regular language accepted by that NFA.[7] In Fig. 6 the NFA with initial state $\mathbf{G}(a)$ in row 1 and final state $\Box$ accepts the language $\{0, 11000\}$, whose Parikh image of the letter 1 is just $\{0, 2\}$ as required.

The approach is formalized in Def. 9 and Prop. 7. This example will be finished after formal definitions and proofs of correctness at the end of Sec. 6.2 when we illustrate the extraction of a UC in SNF with sets of time points. $\Box$

6.2. UCs in SNF with Sets of Time Points

Let C be an unsatisfiable set of SNF clauses, let G be a resolution graph with a vertex $v_{\Box}$ in the main partition that is L_V-labeled with $\Box$, and let G' be the subgraph according to Def. 4 with corresponding UC in SNF C^{uc}. In Def. 9 we start by labeling edges of G' with 1 if the source vertex is time-shifted one step into the future with respect to the target vertex and all other edges with 0. This requires to extend the intuition that we provided in Sec. 6.1 for the production rules listed under "saturation" in Tab. 2 to the full set of production rules. In particular, to see where a time-shift occurs, we need to identify pairs of premises and conclusions such that elements from the now part of a premise interact with elements from the $\mathbf{X}$ part of another premise and/or are propagated to the $\mathbf{X}$ part of the conclusion. The case of production rule step-nx was discussed in Sec. 6.1. Another, straightforward case is production rule BFS-loop-it-init-n: the now part of the premise becomes the $\mathbf{X}$ part of the conclusion. For production rule BFS-loop-it-sub the entire now part of the premise is propagated to the $\mathbf{X}$ part of the conclusion. Similarly, for production rule BFS-loop-conclusion2 the now part of premise 1 is propagated to the $\mathbf{X}$ part of the conclusion. Notice that there also occurs a time-shift between premise 1 and the conclusion of production rule BFS-loop-it-init-c. However, because the resolution graph does not contain an edge between instances of this premise and conclusion, this case can be ignored. All other pairs of premises and conclusions do not involve a time-shift. In the second part of Def. 9 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-shifts that occur on any path from v to $v_{\Box}$ in G'.

Definition 9 (Labeling Edges with Time-Shifts and Vertices with Sets of Time Points). *$L_{E'}$ is a labeling of the set of edges in G', E', with time-shifts in $\{0, 1\}$. $L_{E'}$ maps an edge e to 1 if e is an edge (i) from a vertex labeled with premise 1 to a vertex labeled with the conclusion obtained from production rule step-nx, (ii) from a vertex labeled with the premise to a vertex labeled with the conclusion obtained from production rule BFS-loop-it-init-n, (iii) from a*

[7] For technical reasons we will later reverse the edges in the resolution graph and switch the role of initial and final state.

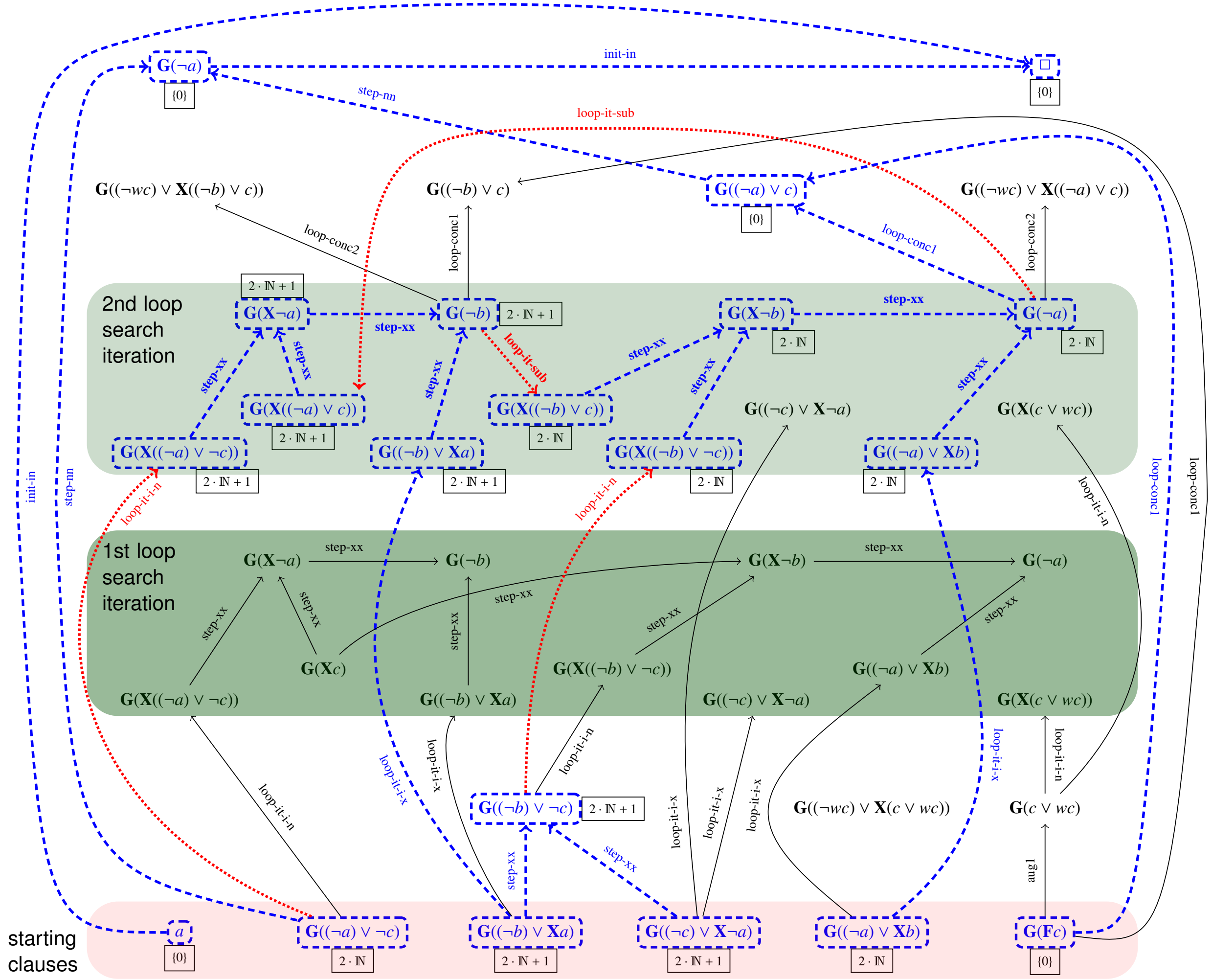

Figure 7: Example of computing sets of time points for a UC in SNF with BFS loop search.

vertex labeled with the premise to a vertex labeled with the conclusion obtained from production rule BFS-loop-it-sub, *or (iv) from a vertex labeled with premise 1 to a vertex labeled with the conclusion obtained from production rule* BFS-loop-conclusion2. *All other edges are mapped to 0.*

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.

Notice that $v_{\Box}$ can be L_V-labeled with either () (an initial clause) or $(\mathbf{G}())$ (a global clause). In the first case it's clear that the contradiction happens at time point 0. In the second case we could assume the contradiction to happen at any non-empty subset I of $\mathbb{N}$. In that case we could report to the user that some clause $c = L_V(v_c)$ is used in the TR proof of unsatisfiability at time points $I + L'_{V'}(v_c)$ rather than at time points $L'_{V'}(v_c)$. As this does not seem to add much information for the user (the fact that the empty clause derived at the end of the proof is a global clause can be easily seen from a log of the proof), we decided to assume the contradiction to happen at time point 0 in both cases.

Running Example 3 (continuing from p. 18)**.** We now continue the more complex running example from Sec. 4.1.2 in Fig. 7. This example will be finished later in this section when we illustrate the extraction of a UC in SNF with sets of time points. Edges in the subgraph backward reachable from □ that involve a time-shift between source

and target vertex according to Def. 9 are marked red, dotted. Backward reachable edges that involve no such time-shift are marked blue, dashed. In the backward reachable subgraph there are four edges that involve a time-shift between source and target vertex. Two of those originate from instances of production rule 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 production rule 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 production rule BFS-loop-conclusion2 that would be labeled with a time-shift of 1, if they were backward reachable from $\Box$: 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 7 contains no edges induced by an instance of production rule 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-shift 0.

Each SNF 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 c to $\Box$ according to Def. 9. 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 2). 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 SNF clauses comprising the second BFS loop search iteration (rows 5 and 6 in the light green shaded rectangle) can reach $\Box$ without passing through any other SNF 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-shift 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-shifts. 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 BFS 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 SNF clause. $\Box$

From now on we assume in this section that the edges and vertices of G' are labeled according to Def. 9. 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. 9. Proposition 7 establishes that the sets of time points obtained in Def. 9 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 2 (Sets of Time Points for Vertices Labeled with Initial Clauses are $\{0\}$)**.** *Every 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 $v_\Box$ the unique vertex in the main partition L_V-labeled with the empty clause $\Box$. 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 $v_\Box$ is an initial clause and $v_\Box$ and all vertices L_V-labeled with initial clauses are connected via edges $L_{E'}$-labeled with 0. The claim now follows with Def. 9 by induction on the distance of a vertex L_V-labeled with an initial clause from $v_\Box$. $\Box$

Lemma 3 (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 — possibly time-shifted —*

subset of the labeling $L'_{V'}(v)$:

$$L'_{V'}(v') \subseteq \begin{cases} (L'_{V'}(v) - 1) \cap \mathbb{N} & \text{if } v \text{ and } v' \text{ are labeled with premise 1 and the conclusion of production rule } \boxed{\text{step-nx}} \text{ or } \boxed{\text{BFS-loop-conclusion2}} \text{ or with the premise and the conclusion of production rule } \boxed{\text{BFS-loop-it-init-n}} \text{ or } \boxed{\text{BFS-loop-it-sub}}\text{; or} \\ L'_{V'}(v) & \text{otherwise.} \end{cases}$$

Proof. Directly by Def. 9. □

Proposition 7 (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 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. 9 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. 10). The definition of a UC in SNF with sets of time points is then immediate in Def. 11. Theorem 3 establishes correctness of the construction. In Prop. 8 we state an upper bound on the complexity of extracting a UC in SNF with sets of time points.

Definition 10 (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 LTLp formula:*[8]

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

Definition 11 (UC in SNF with Sets of Time Points)**.** *Let $c_{1,1}, \ldots, c_{1,n_1}$ be the initial clauses in C^{uc}, $c_{2,1}, \ldots, c_{2,n_2}$ the global clauses in C^{uc}, and $c_{3,1}, \ldots, c_{3,n_3}$ 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.*

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

Proof. In this proof 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. 10. 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 $\underset{I}{c}$.

For each production rule of TR we take two steps. First, we show, given some sets of time points labeling the premises of a production rule, what set of time points could correctly label the conclusion. Second, we show that our construction ensures that the set of time points actually labeling a clause is a subset of the set according to the previous step. We proceed as follows. We first show that inferences based on initial and step resolution rules $\boxed{\text{init-ii}}$, $\boxed{\text{init-in}}$, $\boxed{\text{step-nn}}$, $\boxed{\text{step-nx}}$, and $\boxed{\text{step-xx}}$ also hold when taking sets of time points into account. We continue with augmentation $\boxed{\text{aug1}}$ and $\boxed{\text{aug2}}$. Then we show that production rules $\boxed{\text{BFS-loop-it-init-x}}$, $\boxed{\text{BFS-loop-it-init-n}}$, $\boxed{\text{BFS-loop-it-init-c}}$, and $\boxed{\text{BFS-loop-it-sub}}$ required as part of BFS loop search also apply with sets of time points. Based on that we show what a BFS loop search with sets of time points actually proves, and we finally use that result to establish validity of the remaining BFS loop search conclusions $\boxed{\text{BFS-loop-conclusion1}}$ and $\boxed{\text{BFS-loop-conclusion2}}$.

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 $\boxed{\text{step-nn}}$ 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 production rules $\boxed{\text{init-ii}}$ and $\boxed{\text{init-in}}$, 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 production rules $\boxed{\text{step-nn}}$ and $\boxed{\text{step-xx}}$, 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 production rule $\boxed{\text{step-nx}}$, 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 2 and 3, 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.

For production rules $\boxed{\text{aug1}}$ and $\boxed{\text{aug2}}$ assume that some word π in $(2^{AP})^\omega$ satisfies θ^{uc}. Let $(\underset{I}{\mathbf{G}}(p_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} p_n \underset{I,I}{\vee} \underset{[min(I),\infty)}{\mathbf{F}} l))$ be an eventuality clause in θ^{uc}, and let wl be a fresh atomic proposition. Let π' be a word in $(2^{AP \cup \{wl\}})^\omega$ where π' restricted to AP is identical to π and wl is TRUE in π' in the smallest set of time points such that (i) if for some time point $i \in I$ both $p_1 \vee \ldots \vee p_n$ and l are FALSE in π', then wl is TRUE in π' at that time point: $\forall i \in \mathbb{N} \;.\; ((i \in I) \wedge (\forall 1 \leq j \leq n \;.\; p_j \notin \pi'[i]) \wedge (l \notin \pi'[i])) \rightarrow (wl \in \pi'[i])$, and (ii) if for some time point $i \in \mathbb{N}$ wl is TRUE in π', then wl is TRUE in π' either indefinitely or until l becomes TRUE in π': $\forall i \in \mathbb{N} \;.\; (wl \in \pi'[i]) \rightarrow ((l \in \pi'[i+1]) \vee (wl \in \pi'[i+1]))$. Clearly, π' satisfies θ^{uc}. Moreover, π' also satisfies the result of augmentation $(\underset{I}{\mathbf{G}}(p_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} p_n \underset{I,I}{\vee} l \underset{I,I}{\vee} wl))$ and $(\underset{\mathbb{N}}{\mathbf{G}}((\underset{\mathbb{N}}{\neg} wl) \underset{\mathbb{N},\mathbb{N}}{\vee} (\underset{\mathbb{N}}{\mathbf{X}}(l \underset{\mathbb{N},\mathbb{N}}{\vee} wl))))$. Using Lemma 3 for production rule $\boxed{\text{aug1}}$ and the fact that every set of time points is a subset of $\mathbb{N}$ for production rule $\boxed{\text{aug2}}$, we can conclude that all instances of augmentation are correct also when taking sets of time points into account.

The role of production rules $\boxed{\text{BFS-loop-it-init-x}}$ and $\boxed{\text{BFS-loop-it-init-n}}$ is the propagation of information from the main partition to the loop partition. By Lemma 3 it is clear that information is propagated correctly also with sets of time points. The conclusions of production rule $\boxed{\text{BFS-loop-it-init-c}}$ are disjunctions of the literal l a loop is searched for and parts of a hypothetical fixed point (see also the proof of Lemma 5 in [Sch15]). A successful BFS loop search iteration proves a hypothetical fixed point to be an actual fixed point. As can be seen below when we show what a successful BFS loop search iteration actually proves, it does not matter where the hypothetical fixed point comes from. Neither is it required that the literal l a loop is searched for is derived from an actual eventuality clause. Hence, no information needs to be propagated correctly by production rule $\boxed{\text{BFS-loop-it-init-c}}$ for the result of a successful BFS loop search iteration to be correct, and therefore there is nothing more to prove w.r.t. production rule $\boxed{\text{BFS-loop-it-init-c}}$ when taking sets of time points into account. Production rule $\boxed{\text{BFS-loop-it-sub}}$ continues to record a correct relation between two clauses by a similar argument as for the rules used in saturation.

Now we show what a successful BFS loop search iteration actually proves in the presence of sets of time points. Let L be the partition of a successful BFS loop search iteration restricted to the subgraph G', let C' be the set of global clauses with empty $\mathbf{X}$ part labeling a vertex in L, and let $c = (\mathbf{G}(p_1 \vee \ldots \vee p_n)) \in C'$. Let v_c be the corresponding vertex in G' and let I be the $L'_{V'}$-labeling of v_c. We trace c, starting at a single time point $i \in I$, backward through L. 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 production rule [step-xx] and that are L_V-labeled with a clause generated by production rule [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 production rule [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 the set of clauses that are required (in addition to the clauses produced by production rules [BFS-loop-it-init-x] and [BFS-loop-it-init-n]) to prove the clauses in $C'_{i'}$ at time point i' by saturation restricted to production 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 production rule [BFS-loop-it-init-c] and $C'_{i'}$ is bounded from above by C'. 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 production rule [step-xx] in the presence of sets of time points as argued above, we can infer that, assuming the clauses produced by production rules [BFS-loop-it-init-x] and [BFS-loop-it-init-n], 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 the clauses produced by production rules [BFS-loop-it-init-x] and [BFS-loop-it-init-n], it is provable that

$$\forall i \leq i' \,.\, (\bigwedge\nolimits_{c'' \in C''_{i'}} c'') \rightarrow (\bigwedge\nolimits_{c' \in C'_{i'}} c'). \tag{22}$$

Again using the definition of $C''_{i'}$ and $C'_{i'}$ in 1.–3. above as well as the correctness of production rule [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.,

$$\forall i < i' \,.\, (\bigwedge\nolimits_{c' \in C'_{i'}} c') \rightarrow (\bigwedge\nolimits_{c'' \in C''_{i'-1}} c''). \tag{23}$$

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

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

I.e., we have

$$\forall i \leq i' \,.\, \forall c'' \in C''_{i'} \,.\, (\mathbf{X} \underset{\{i'+1\}}{\mathbf{G}} l) \rightarrow c''. \tag{24}$$

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 (22) – (24) with some rewriting we obtain (25) to tell us what a successful BFS loop search with sets of time points actually proves:

$$(\underset{\{i\}}{\mathbf{G}}((p_1 \underset{\{i\},\{i\}}{\vee} \ldots \underset{\{i\},\{i\}}{\vee} p_n) \underset{\{i\},\{i\}}{\vee} (\underset{\{i+1\}}{\mathbf{X}} \underset{[i+1,\infty)}{\mathbf{G}} \underset{[i+1,\infty)}{\neg} l))). \tag{25}$$

For production rule [BFS-loop-conclusion1] consider an eventuality clause $(\underset{I}{\mathbf{G}}(q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'} \underset{I,I}{\vee} \underset{[min(I),\infty)}{\mathbf{F}} l))$ and a clause $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}} \underset{[min(I')+1,\infty)}{\mathbf{G}} \underset{[min(I')+1,\infty)}{\neg} l)))$ obtained from a successful BFS loop search iteration for 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

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

By Lemma 3 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', production rule $\boxed{\text{BFS-loop-conclusion1}}$ remains correct with sets of time points.

For production rule $\boxed{\text{BFS-loop-conclusion2}}$ consider an eventuality clause $(\underset{I}{\mathbf{G}}(q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'} \underset{I,I}{\vee} \underset{[min(I),\infty)}{\mathbf{F}} l))$, the results of augmentation for the eventuality clause $(\underset{I}{\mathbf{G}}(q_1 \underset{I,I}{\vee} \ldots \underset{I,I}{\vee} q_{n'} \underset{I,I}{\vee} l \underset{I,I}{\vee} wl))$ and $(\underset{\mathbb{N}}{\mathbf{G}}((\underset{\mathbb{N}}{\neg} wl) \underset{\mathbb{N},\mathbb{N}}{\vee} (\underset{\mathbb{N}}{\mathbf{X}}(l \underset{\mathbb{N},\mathbb{N}}{\vee} wl))))$, and a clause $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}} \underset{[min(I')+1,\infty)}{\mathbf{G}} \underset{[min(I')+1,\infty)}{\neg} l)))$ obtained from a successful BFS loop search iteration for l. Assume a word π' in $(2^{AP \cup \{wl\}})^{\omega}$ that satisfies 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 $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}} \underset{[min(I')+1,\infty)}{\mathbf{G}} \underset{[min(I')+1,\infty)}{\neg} l)))$, and whose valuation of wl is as in the case for augmentation above. We show that π' also satisfies $c \equiv (\underset{I'-1}{\mathbf{G}}((\underset{I'-1}{\neg} wl) \underset{I'-1,I'-1}{\vee} (\underset{I'}{\mathbf{X}}(p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n \underset{I',I'}{\vee} l))))$. Notice that, given our valuation of wl in π', if wl is TRUE in π' at some time point i, then l becomes TRUE in π' at some later time point $i' > i$. First consider the case that for each time point $i \in \mathbb{N}$ i is not in $I' - 1$, or wl is FALSE in π' at time point i, or $p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n$ is TRUE in π' at time point $i + 1$, or l is TRUE in π' at time point $i + 1$. In that case π' also satisfies c. Now consider the remaining case in which there exists a time point $i \in \mathbb{N}$ such that $i \in I' - 1$, $wl \in \pi'[i]$, $\forall 0 < j \leq n \,.\, p_j \notin \pi'[i+1]$, and $l \notin \pi'[i+1]$. However, this is impossible. On the one hand, wl being TRUE in π' at time point i implies that l becomes TRUE in π' for some $i' > i$. On the other hand, $\forall 0 < j \leq n \,.\, p_j \notin \pi'[i+1]$, $l \notin \pi'[i+1]$, and $(\underset{I'}{\mathbf{G}}((p_1 \underset{I',I'}{\vee} \ldots \underset{I',I'}{\vee} p_n) \underset{I',I'}{\vee} (\underset{I'+1}{\mathbf{X}} \underset{[min(I')+1,\infty)}{\mathbf{G}} \underset{[min(I')+1,\infty)}{\neg} l)))$ imply that l cannot become TRUE in π' after i. Hence, π' also satisfies c. Using Lemma 3, we can conclude that all instances of production rule $\boxed{\text{BFS-loop-conclusion2}}$ are correct also when taking sets of time points into account.

We have just shown that all productions in G' remain correct in the presence of time points. Moreover, by construction of G' a contradiction is obtained at time point 0 from θ^{uc}. Hence, θ^{uc} is unsatisfiable. □

Proposition 8 (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 $O(|V'|^3)$, where $|V'|$ is exponential in $|AP| + log(|C|)$.*[9]

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 $O(1)$ for element addition, removal, and membership test as well as $O(n)$ for set creation and reset to $\emptyset$. A list of length n incurs cost $O(1)$ for creation, element addition, and emptiness check as well as $O(n)$ 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. Sets of time points are represented as lists of linear sets, which, in turn, are stored as pairs of naturals.

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[10] by treating all edges[11] $L_{E'}$-labeled with 0 as ϵ-transitions and applying a standard method for elimination of ϵ-transitions in NFA [HU79]. That leaves us with an 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 $O(|V'| + |E'|)$.

Turning G' into a Unary NFA. (i) Designate $v_{\Box}$ as the initial vertex: $O(1)$. (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: $O(|V'| \cdot (|V'| + |E'|))$. (iii) For each vertex compute the set of vertices reachable from that vertex via a 1-$L_{E'}$-labeled edge followed by a sequence of 0-$L_{E'}$-labeled edges: $O(|V'|^2 + |V'| \cdot |E'|)$. (iv) For each vertex compute the set and list of vertices reachable from that vertex via a sequence of 0-$L_{E'}$-labeled edges, followed by a 1-$L_{E'}$-labeled edge, and followed by a sequence of 0-$L_{E'}$-labeled edges: $O(|V'|^3)$. (v) Replace the set of edges E' with

[9] Note that when using the translation from LTL into SNF in Def. 1 both $|AP|$ and $|C|$ are linear in the size of the input LTL formula.
[10] A *unary* NFA is an NFA over a unary alphabet.
[11] We use the terms "vertex" and "state" as well as "edge" and "transition" interchangeably.

the edges such that there is one 1-$L_{E'}$-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'': $O(|V'| + |E'| + |E''|)$. The overall cost for turning G' into a unary NFA is, therefore, $O(|V'|^3 + |V'| \cdot (|V'| + |E'|) + |E''|)$.

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 $O(|C^{uc}|)$.

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. Due to space constraints we refer to Appendix A for details of the algorithm and the analysis of its complexity. It turns out that the overall time to compute Parikh images is $O(|V'|^3 + |C^{uc}| \cdot |V'|^2 + |V'| \cdot |E''|)$ (the original algorithm in [Gaw11] also runs in time cubic in the number of vertices).

Summing Up. The time taken for all steps is bounded by $O(|V'|^3 + |C^{uc}| \cdot |V'|^2 + |V'| \cdot (|V'| + |E'|) + |V'| \cdot |E''|) = O(|V'|^3)$. For a proof of the fact that $|V'|$ is exponential in $|AP| + log(|C|)$ see Lemma 1 in [Sch15]. This concludes the proof. □

Running Example 2 (continuing from p. 23)**.** We now apply Def. 11 to the running example in Fig. 6 and obtain (26) as a UC in SNF with sets of time points. Notice that $(\mathbf{G}((\neg b) \vee \mathbf{X} b))$ is not contained in (26) and that all sets of time points are subsets of $\{0, 1, 2\}$. This concludes this running example.

$$b \underset{\{0\},\{0\}}{\wedge} (\underset{\{0\}}{\mathbf{G}}((\neg b) \underset{\{0\},\{0\}}{\vee} c \underset{\{0\},\{0\}}{\vee} \underset{\{1\}}{\mathbf{X}} d)) \underset{\{0\},\{0\}}{\wedge} (\underset{\{0\}}{\mathbf{G}}((\underset{\{0\}}{\neg} a) \underset{\{0\},\{0\}}{\vee} \underset{\{0\}}{\neg} c)) \underset{\{0\},\{0\}}{\wedge} (\underset{\{1\}}{\mathbf{G}}((\underset{\{1\}}{\neg} d) \underset{\{1\},\{1\}}{\vee} \underset{\{2\}}{\mathbf{X}} \underset{\{2\}}{\neg} a)) \underset{\{0\},\{0\}}{\wedge} (\underset{\{0,2\}}{\mathbf{G}} a) \tag{26}$$

□

Running Example 3 (continuing from p. 26)**.** We also finish the more complex running example in Fig. 7 by applying Def. 11 and obtaining (27) 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 SNF clause shows that only a single occurrence of c is required for unsatisfiability. Finally, the fourth SNF clause has $\neg c$ at even time points, while the fifth SNF clause becomes relevant at odd time points; thus all potential occurrences of c are covered.

$$\begin{gathered} a \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}}{\mathbf{G}}((\underset{2\cdot\mathbb{N}}{\neg} a) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\vee} \underset{2\cdot\mathbb{N}+1}{\mathbf{X}} b)) \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}+1}{\mathbf{G}}((\underset{2\cdot\mathbb{N}+1}{\neg} b) \underset{2\cdot\mathbb{N}+1,2\cdot\mathbb{N}+1}{\vee} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} a)) \underset{\{0\},\{0\}}{\wedge} \\ (\underset{2\cdot\mathbb{N}}{\mathbf{G}}((\underset{2\cdot\mathbb{N}}{\neg} a) \underset{2\cdot\mathbb{N},2\cdot\mathbb{N}}{\vee} \underset{2\cdot\mathbb{N}}{\neg} c)) \underset{\{0\},\{0\}}{\wedge} (\underset{2\cdot\mathbb{N}+1}{\mathbf{G}}((\underset{2\cdot\mathbb{N}+1}{\neg} c) \underset{2\cdot\mathbb{N}+1,2\cdot\mathbb{N}+1}{\vee} \underset{2\cdot\mathbb{N}+2}{\mathbf{X}} \underset{2\cdot\mathbb{N}+2}{\neg} a)) \underset{\{0\},\{0\}}{\wedge} (\underset{\{0\}}{\mathbf{G}}(\underset{\mathbb{N}}{\mathbf{F}} c)) \end{gathered} \tag{27}$$

□

6.3. UCs in LTL with Sets of Time Points

Definition 12 adds sets of time points to a UC in LTL by transferring them from a UC in SNF with sets of time points to a UC in LTL. The mapping of a UC in SNF with sets of time points to a UC in LTL with sets of time points is a direct extension of the corresponding mapping without sets of time points in Def. 6. Remember that in Def. 6 an occurrence of a subformula ψ is *not* replaced with TRUE or FALSE in the UC in LTL if the UC in SNF contains an SNF clause c such that x_ψ occurs in c in a position that is marked blue boxed in Tab. 1. To obtain the set of time points at which subformula ψ is relevant we simply take the union of the sets of time points that are labeling the occurrences of x_ψ that are marked blue boxed in Tab. 1 in the SNF clauses of the UC in SNF. The proof idea for Thm. 4 is similar to that of Thm. 2 (see [Sch15]), but in addition we need to define a translation from the corresponding fragment of LTLp into SNF with sets of time points, which must be shown to be satisfiability-preserving.

Definition 12 (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(ϕ) be its SNF, let ϕ^{uc} be the UC of ϕ in LTL, and let θ^{uc} be the UC of SNF(ϕ) in SNF with sets of time points. Construct the* UC of ϕ in LTL with sets of time points*, θ'^{uc}, by assigning a set of time points I to each occurrence of a subformula ψ in ϕ^{uc} as follows. Let $I', I'', \ldots$ be the sets of time points of the occurrences of the proposition x_ψ in θ^{uc} that are marked blue boxed in Tab. 1. 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.*

Table 4: Satisfiability-preserving translation from a UC in LTL with sets of time points into SNF used in the proof of Thm. 4.

Subformula	Proposition	SNF Clauses (positive polarity occurrences)	SNF Clauses (negative polarity occurrences)
TRUE/FALSE/p	TRUE/FALSE/p	none	none
$\underset{I}{\neg}\psi$	$x_{\neg\psi}$	$(\underset{I}{\mathbf{G}}(x_{\neg\psi} \underset{I,I}{\rightarrow} \underset{I}{\neg} x_\psi))$	$(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\neg\psi}) \underset{I,I}{\rightarrow} x_\psi))$
$\psi \underset{I,I'}{\vee} \psi'$	$x_{\psi\vee\psi'}$	$(\underset{I}{\mathbf{G}}(x_{\psi\vee\psi'} \underset{I,I}{\rightarrow}(x_\psi \underset{I,I}{\vee} x_{\psi'})))$	$(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\psi\vee\psi'}) \underset{I,I}{\rightarrow} \underset{I}{\neg} x_\psi))$, $(\underset{I'}{\mathbf{G}}((\underset{I'}{\neg} x_{\psi\vee\psi'}) \underset{I',I'}{\rightarrow} \underset{I'}{\neg} x_{\psi'}))$
$\psi \underset{I,I'}{\wedge} \psi'$	$x_{\psi\wedge\psi'}$	$(\underset{I}{\mathbf{G}}(x_{\psi\wedge\psi'} \underset{I,I}{\rightarrow} x_\psi))$, $(\underset{I'}{\mathbf{G}}(x_{\psi\wedge\psi'} \underset{I',I'}{\rightarrow} x_{\psi'}))$	$(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\psi\wedge\psi'}) \underset{I,I}{\rightarrow}((\underset{I}{\neg} x_\psi) \underset{I,I}{\vee} \underset{I}{\neg} x_{\psi'})))$
$\underset{I}{\mathbf{X}}\psi$	$x_{\mathbf{X}\psi}$	$(\underset{I-1}{\mathbf{G}}(x_{\mathbf{X}\psi} \underset{I-1,I-1}{\rightarrow} \underset{I}{\mathbf{X}} x_\psi))$	$(\underset{I-1}{\mathbf{G}}((\underset{I-1}{\neg} x_{\mathbf{X}\psi}) \underset{I-1,I-1}{\rightarrow} \underset{I}{\mathbf{X}} \underset{I}{\neg} x_\psi))$
$\psi \underset{I,I'}{\mathbf{U}} \psi'$	$x_{\psi\mathbf{U}\psi'}$	$(\underset{I}{\mathbf{G}}(x_{\psi\mathbf{U}\psi'} \underset{I,I}{\rightarrow}(x_{\psi'} \underset{I,I}{\vee} x_\psi)))$, $(\underset{I'}{\mathbf{G}}(x_{\psi\mathbf{U}\psi'} \underset{I',I'}{\rightarrow}(x_{\psi'} \underset{I',I'}{\vee} \underset{I'+1}{\mathbf{X}} x_{\psi\mathbf{U}\psi'})))$, $(\underset{I'}{\mathbf{G}}(x_{\psi\mathbf{U}\psi'} \underset{I',I'}{\rightarrow} \underset{[min(I'),\infty)}{\mathbf{F}} x_{\psi'}))$	$(\underset{I'}{\mathbf{G}}((\underset{I'}{\neg} x_{\psi\mathbf{U}\psi'}) \underset{I',I'}{\rightarrow} \underset{I'}{\neg} x_{\psi'}))$, $(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\psi\mathbf{U}\psi'}) \underset{I,I}{\rightarrow}((\underset{I}{\neg} x_\psi) \underset{I,I}{\vee} \underset{I+1}{\mathbf{X}} \underset{I+1}{\neg} x_{\psi\mathbf{U}\psi'})))$
$\psi \underset{I,I'}{\mathbf{R}} \psi'$	$x_{\psi\mathbf{R}\psi'}$	$(\underset{I'}{\mathbf{G}}(x_{\psi\mathbf{R}\psi'} \underset{I',I'}{\rightarrow} x_{\psi'}))$, $(\underset{I}{\mathbf{G}}(x_{\psi\mathbf{R}\psi'} \underset{I,I}{\rightarrow}(x_\psi \underset{I,I}{\vee} \underset{I+1}{\mathbf{X}} x_{\psi\mathbf{R}\psi'})))$	$(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\psi\mathbf{R}\psi'}) \underset{I,I}{\rightarrow}((\underset{I}{\neg} x_{\psi'}) \underset{I,I}{\vee} \underset{I}{\neg} x_\psi)))$, $(\underset{I'}{\mathbf{G}}((\underset{I'}{\neg} x_{\psi\mathbf{R}\psi'}) \underset{I',I'}{\rightarrow}((\underset{I'}{\neg} x_{\psi'}) \underset{I',I'}{\vee} \underset{I'+1}{\mathbf{X}} \underset{I'+1}{\neg} x_{\psi\mathbf{R}\psi'})))$, $(\underset{I'}{\mathbf{G}}((\underset{I'}{\neg} x_{\psi\mathbf{R}\psi'}) \underset{I',I'}{\rightarrow} \underset{[min(I'),\infty)}{\mathbf{F}} \underset{[min(I'),\infty)}{\neg} x_{\psi'}))$
$\underset{I}{\mathbf{F}}\psi$	$x_{\mathbf{F}\psi}$	$(\underset{I}{\mathbf{G}}(x_{\mathbf{F}\psi} \underset{I,I}{\rightarrow} \underset{[min(I),\infty)}{\mathbf{F}} x_\psi))$	$(\underset{\mathbb{N}}{\mathbf{G}}((\underset{\mathbb{N}}{\neg} x_{\mathbf{F}\psi}) \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \underset{\mathbb{N}+1}{\mathbf{X}} \underset{\mathbb{N}+1}{\neg} x_{\mathbf{F}\psi}))$, $(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\mathbf{F}\psi}) \underset{I,I}{\rightarrow} \underset{I}{\neg} x_\psi))$
$\underset{I}{\mathbf{G}}\psi$	$x_{\mathbf{G}\psi}$	$(\underset{\mathbb{N}}{\mathbf{G}}(x_{\mathbf{G}\psi} \underset{\mathbb{N},\mathbb{N}}{\rightarrow} \underset{\mathbb{N}+1}{\mathbf{X}} x_{\mathbf{G}\psi}))$, $(\underset{I}{\mathbf{G}}(x_{\mathbf{G}\psi} \underset{I,I}{\rightarrow} x_\psi))$	$(\underset{I}{\mathbf{G}}((\underset{I}{\neg} x_{\mathbf{G}\psi}) \underset{I,I}{\rightarrow} \underset{[min(I),\infty)}{\mathbf{F}} \underset{[min(I),\infty)}{\neg} x_\psi))$

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 a set of SNF clauses with sets of time points θ' as in Def. 1 but using Tab. 4 instead of Tab. 1. Besides generating clauses (where Tab. 4 is identical to Tab. 1) Tab. 4 also shows how to transfer sets of time points from occurrences of LTL subformulas to SNF clauses. Note that, by the construction of θ'^{uc} from θ^{uc}, in θ'^{uc} we have that (i) for positive polarity occurrences of $\vee$-subformulas as well as for negative polarity occurrences of $\wedge$-subformulas $I = I'$, (ii) for occurrences of $\mathbf{X}$-subformulas I does not contain 0, and (iii) for positive polarity occurrences of $\mathbf{U}$-subformulas and for negative polarity occurrences of $\mathbf{R}$-subformulas $I \subseteq I'$. In most cases the translation in Tab. 4 is an exact reversal of Def. 12. Exceptions are positive polarity occurrences of $\mathbf{U}$-subformulas, negative polarity occurrences of $\mathbf{R}$-subformulas, negative polarity occurrences of $\mathbf{F}$-subformulas, and positive polarity occurrences of $\mathbf{G}$-subformulas. In each of those cases at least one set of time points in θ' may be larger than that of the corresponding clause in θ^{uc}.

As in the proof of Thm. 2 (see [Sch15]) 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}. With the unsatisfiability of θ^{uc} and Prop. 4 we have that θ' is unsatisfiable. Hence, provided the translation from θ'^{uc} into θ' is satisfiability preserving, 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 π. 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' is easy to verify given the above translation and the semantics of LTLp. This concludes the proof. □

It is easy to see that no subformula in (1) or (4) can be replaced with TRUE (for positive polarity occurrences) or FALSE (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. 6). 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.

Running Example 1 (continuing from p. 20)**.** As an example for Def. 12 we now continue the running example from Sec. 4.2. The unsatisfiability of (14) can be established by taking only time points 0 and 1 into account. Computing a UC in SNF with sets of time points from (20) results in (28) shown below. Notice that all sets of time points are

subsets of $\{0, 1\}$.

$$
\begin{aligned}
&\{(x_{\phi}),\\
&(\underset{\{0\}}{\mathbf{G}}(x_{\phi} \underset{\{0\},\boxed{\{0\}}}{\rightarrow} x_{(\mathbf{X}\neg p)\wedge(\mathbf{G}\neg q)})), (\underset{\{0\}}{\mathbf{G}}(x_{\phi} \underset{\{0\},\boxed{\{0\}}}{\rightarrow} x_{p\mathbf{U}(q\wedge r)})),\\
&(\underset{\{0\}}{\mathbf{G}}(x_{(\mathbf{X}\neg p)\wedge(\mathbf{G}\neg q)} \underset{\{0\},\boxed{\{0\}}}{\rightarrow} x_{\mathbf{X}\neg p})), (\underset{\{0\}}{\mathbf{G}}(x_{(\mathbf{X}\neg p)\wedge(\mathbf{G}\neg q)} \underset{\{0\},\boxed{\{0\}}}{\rightarrow} x_{\mathbf{G}\neg q})),\\
&(\underset{\{0\}}{\mathbf{G}}(x_{\mathbf{X}\neg p} \underset{\{0\},\{0\}}{\rightarrow} \underset{\boxed{\{1\}}}{\mathbf{X}} x_{\neg p})),\\
&(\underset{\{1\}}{\mathbf{G}}(x_{\neg p} \underset{\{1\},\{1\}}{\rightarrow} \underset{\boxed{\{1\}}}{\neg} p)),\\
&(\underset{\{0\}}{\mathbf{G}}(x_{\mathbf{G}\neg q} \underset{\{0\},\{0\}}{\rightarrow} \underset{\{1\}}{\mathbf{X}} x_{\mathbf{G}\neg q})), (\underset{\{0,1\}}{\mathbf{G}}(x_{\mathbf{G}\neg q} \underset{\{0,1\},\boxed{\{0,1\}}}{\rightarrow} x_{\neg q})),\\
&(\underset{\{0,1\}}{\mathbf{G}}(x_{\neg q} \underset{\{0,1\},\{0,1\}}{\rightarrow} \underset{\boxed{\{0,1\}}}{\neg} q)),\\
&(\underset{\{1\}}{\mathbf{G}}(x_{p\mathbf{U}(q\wedge r)} \underset{\{1\},\{1\}}{\rightarrow} (x_{q\wedge r} \underset{\boxed{\{1\}},\boxed{\{1\}}}{\vee} p))), (\underset{\{0\}}{\mathbf{G}}(x_{p\mathbf{U}(q\wedge r)} \underset{\{0\},\{0\}}{\rightarrow} (x_{q\wedge r} \underset{\boxed{\{0\}},\{0\}}{\vee} \underset{\{1\}}{\mathbf{X}} x_{p\mathbf{U}(q\wedge r)}))),\\
&(\underset{\{0,1\}}{\mathbf{G}}(x_{q\wedge r} \underset{\{0,1\},\boxed{\{0,1\}}}{\rightarrow} q))\}.
\end{aligned}
\tag{28}
$$

With Def. 12 we translate the UC in SNF (28) back to the UC in LTL (29). All sets of time points in (28) that are used to assign sets of time points in (29) are marked blue boxed. Most sets of time points in (29) are directly obtained from a set of time points in (28). The right hand operand of the **U**-subformula, $q \wedge r$, is assigned the union of the sets of time points labeling $x_{q\wedge r}$ in $(\underset{\{1\}}{\mathbf{G}}(x_{p\mathbf{U}(q\wedge r)} \underset{\{1\},\{1\}}{\rightarrow} (x_{q\wedge r} \underset{\{1\},\{1\}}{\vee} p)))$ and in $(\underset{\{0\}}{\mathbf{G}}(x_{p\mathbf{U}(q\wedge r)} \underset{\{0\},\{0\}}{\rightarrow} (x_{q\wedge r} \underset{\{0\},\{0\}}{\vee} \underset{\{1\}}{\mathbf{X}} x_{p\mathbf{U}(q\wedge r)})))$, $\{0, 1\}$. The sets of time points labeling $(\underset{\{0\}}{\mathbf{G}}(x_{\mathbf{G}\neg q} \underset{\{0\},\{0\}}{\rightarrow} \underset{\{1\}}{\mathbf{X}} x_{\mathbf{G}\neg q}))$ are not needed. This concludes this running example.

$$
\theta^{uc} = ((\underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg} p) \underset{\{0\},\{0\}}{\wedge} (\underset{\{0,1\}}{\mathbf{G}} \underset{\{0,1\}}{\neg} q)) \underset{\{0\},\{0\}}{\wedge} (p \underset{\{1\},\{0,1\}}{\mathbf{U}} (q \underset{\{0,1\},\emptyset}{\wedge} \text{TRUE})) \tag{29}
$$

□

7. Experimental Evaluation

Our implementation, examples, and log files are available from `http://www.schuppan.de/viktor/theoreticalcomputerscience16/`.

7.1. Implementation

We use the version of TRP++ extended with extraction of UCs from [Sch15] 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 [boo, SLL02].

In the preliminary version of this work [Sch13] we used sharing of same polarity occurrences of a subformula in the translation from LTL into SNF (this improves performance and, therefore, is the default setting). To avoid an influence on the sets of time points obtained we disabled sharing of same polarity occurrences of a subformula in the translation from LTL into SNF in this paper. The data available from `http://www.schuppan.de/viktor/theoreticalcomputerscience16/` include results with both sharing disabled and sharing enabled.

7.2. Algorithms for Extracting Sets of Time Points

We implemented extraction of sets of time points along the lines of the proofs of Prop. 7, 8. 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 $\epsilon^* 1 \epsilon^*$, 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.

Table 5: Overview of benchmark families. All instances are unsatisfiable.

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, WDMR08]	69	69	69	4605
	anzu_genbuf	[BGJ+07]	15	15	15	1924
	forobots	[BDF09]	25	25	25	635
crafted	**schuppan_O1formula**	[SD11]	27	27	27	4006
	schuppan_O2formula	[SD11]	8	8	6	91
	schuppan_phltl	[SD11]	4	4	4	125
random	**rozier_formulas**	[RV10]	61	61	61	155
	trp	[HS02]	397	397	392	1421

7.3. Benchmarks

Our examples, all of which are unsatisfiable, 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 [WDMR08, 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 FALSE during the translation from LTL into 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.

7.4. Aims of Experiments

With our experiments we aim to answer the following questions. The first two questions are related to the benefits of UCs with sets of time points, while the latter two questions cover the costs. (i) Which sets of time points are obtained in the UCs of which benchmark families, and how often are these sets of time points obtained? (ii) Do the sets of time points obtained reveal interesting information? For this question we will only provide a qualitative rather than a quantitative answer. (iii) How much run time and memory overhead does the extraction of UCs with sets of time points incur compared to the extraction of UCs without sets of time points? (iv) How does extraction of UCs with sets of time points using Gawrychowski's algorithm compare to using Sawa's algorithm in terms of run time and memory usage?

7.5. Setup

The experiments were performed on a laptop with Intel Core i7 M 620 processor at 2 GHz running Ubuntu 14.04. Run time and memory usage were measured with `run` [BJ]. The time and memory limits were 600 seconds and 6 GB.

7.6. Results

In the following we answer questions (i)–(iv) in sequence.

In Tab. 6 we show how often which sets of time points were obtained in the UCs of which benchmark families. The first column lists the sets of time points that occurred in any of the UCs that we obtained on our examples. Columns 2–9 then show for each benchmark family the number of occurrences of the sets of time points in the UCs from this benchmark family.[12] On the one hand, many sets of time points are relatively simple such as $\{0\}$, $\{1\}$, $\mathbb{N}$, or $\mathbb{N}+1$. On the other hand, many other sets of time points are obtained, including ones indicating cyclic behavior with periods greater than 1 as well as $\mathbb{N}$ shifted by offsets of up to 10. Finally, notice that also a frequent set of time points such as $\{0\}$ can provide interesting information if it occurs, e.g., below a $\mathbf{G}$ operator.

[12] Notice that while the exact numbers that we obtained in our experiments are stated in the table, they should be taken as an indication only. For example, in parts of the output of our implementation a single n-ary conjunction or disjunction is used with n sets of time points (one for each conjunct or disjunct). Alternatively, such a conjunction or disjunction could be written as a sequence of $n-1$ nested binary conjunctions or disjunctions with $2 \cdot (n-1)$ sets of time points, leading to an almost twofold difference in the number of sets of time points.

Table 6: Number of occurrences of sets of time points in UCs of benchmark families. No entry stands for 0.

	application			crafted			random	
set of time points	alaska_lift	anzu_-genbuf	forobots	schuppan_-O1formula	schuppan_-O2formula	schuppan_-phltl	rozier_-formulas	trp
$\{0\}$	1655	245	832	54	72	84	322	49774
$\{0, 1\}$	158		3				11	
$\{0, 1, 2\}$							3	
$\{0, 2\}$							1	
$\{1\}$	767	4	51	81			221	14219
$\{1, 2\}$	69						21	
$\{1, 2, 3\}$							6	
$\{1, 2, 3, 4\}$							1	
$\{1, 2, 4\}$							9	
$\{1, 3\}$							4	
$\{2\}$	167						41	
$\{2, 3\}$							7	
$\{2, 3, 4\}$							1	
$\{3\}$							11	
$\{4\}$							2	
$\mathbb{N}$	651	99	746		44	20	504	21434
$\mathbb{N} + 1$	427	121	1085		396	29	275	16572
$\mathbb{N} + 2$		104	423			23	65	1148
$\mathbb{N} + 3$		106	86			30	21	5
$\mathbb{N} + 4$		42	85			31	13	
$\mathbb{N} + 5$		30	92			21	8	
$\mathbb{N} + 6$		12				12		
$\mathbb{N} + 7$						13		
$\mathbb{N} + 8$						5		
$\mathbb{N} + 9$						5		
$\mathbb{N} + 10$						5		
$4 \cdot \mathbb{N}$		20						
$4 \cdot \mathbb{N} + 1$		72						
$\{4 \cdot \mathbb{N} + 1, 4 \cdot \mathbb{N} + 2, 4 \cdot \mathbb{N} + 3\}$		20						
$4 \cdot \mathbb{N} + 2$		24						
$\{4 \cdot \mathbb{N} + 2, 4 \cdot \mathbb{N} + 3\}$		24						
$\{4 \cdot \mathbb{N} + 2, 4 \cdot \mathbb{N} + 3, 4 \cdot \mathbb{N} + 4\}$		4						
$4 \cdot \mathbb{N} + 3$		20						
$\{4 \cdot \mathbb{N} + 3, 4 \cdot \mathbb{N} + 4\}$		4						
$4 \cdot \mathbb{N} + 4$		36						
$4 \cdot \mathbb{N} + 5$		20						
$5 \cdot \mathbb{N}$								368
$5 \cdot \mathbb{N} + 1$								506
$5 \cdot \mathbb{N} + 2$								506
$5 \cdot \mathbb{N} + 3$								506
$5 \cdot \mathbb{N} + 4$								506
$5 \cdot \mathbb{N} + 5$								138
$12 \cdot \mathbb{N}$								536
$12 \cdot \mathbb{N} + 1$								737
$12 \cdot \mathbb{N} + 2$								737
$12 \cdot \mathbb{N} + 3$								737
$12 \cdot \mathbb{N} + 4$								737
$12 \cdot \mathbb{N} + 5$								737
$12 \cdot \mathbb{N} + 6$								737
$12 \cdot \mathbb{N} + 7$								737
$12 \cdot \mathbb{N} + 8$								737
$12 \cdot \mathbb{N} + 9$								737
$12 \cdot \mathbb{N} + 10$								737
$12 \cdot \mathbb{N} + 11$								737
$12 \cdot \mathbb{N} + 12$								201

Sets of time points often help to understand why a UC is unsatisfiable. For some subfamilies of the **anzu_genbuf** and **trp** families sets of time points show that some subformulas are required only every 4th, 5th, or 12th time point. For subfamilies of the **anzu_genbuf** family sets of time points highlight that disjuncts of an invariant hold at different time points of a cyclic interaction between subformulas. In some subfamilies of the **alaska_lift** and the **schuppan_phltl** families occurrences of $\underset{\{0\}}{\mathbf{G}}\,\mathbf{F}\psi$ indicate that a single occurrence of ψ is sufficient for unsatisfiability rather than infinitely many occurrences. In families **alaska_lift**, **forobots**, and **rozier_formulas** there are instances in which — despite the presence of several temporal operators, including **G**, in the UC— all sets of time points of

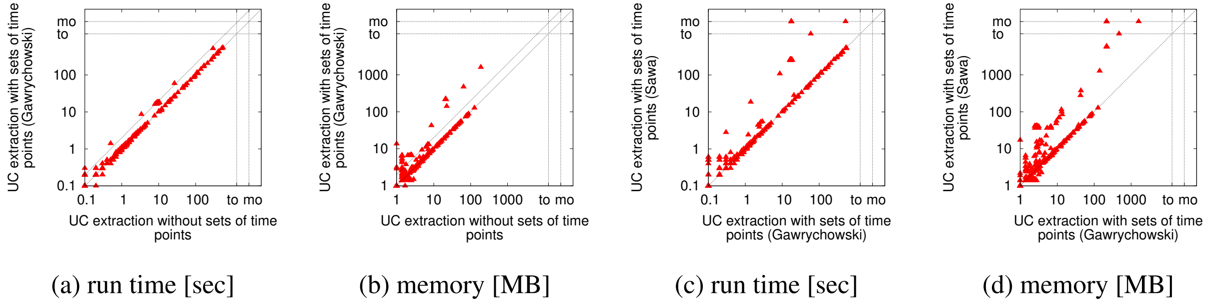

Figure 8: Overhead of UC extraction with sets of time points: (a) and (b) show run time and memory for UC extraction with sets of time points using Gawrychowski's algorithm (y-axis) versus UC extraction without sets of time points (x-axis). (c) and (d) compare run time and memory for Sawa's algorithm (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 $y = 2x$.

the UC are a subset of a small finite set such as $\{0, 1, 2, 3\}$ or even $\{0\}$. For the **schuppan_phltl** family (a temporal version of the pigeon hole problem (e.g., [BHvMW09]); n pigeon holes are turned into a single pigeon hole over n time points) sets of time points indicate how the conditions of mutual exclusivity for the hole are invoked one after the other. Finally, on some instances of the **rozier_formulas** family sets of time points help to identify which occurrences of a proposition interact to obtain unsatisfiability. For some of the above cases an example can be found in Sec. 2.

In Fig. 8 (a) and (b) we show the overhead that is incurred by extracting UCs with sets of time points compared to extracting UCs without sets of time points. Our data show that extraction of UCs with sets of time points is possible with quite acceptable overhead in run time and memory usage. In particular, out of the 701 instances we considered with UC extraction without sets of time points, a UC was obtained for 606 instances, 24 instances timed out, and 71 instances were not tried because easier instances of the same benchmark family had already timed out. With sets of time points enabled the same 606 instances were solved using Gawrychowski's algorithm and 7 less using Sawa's algorithm. For 75 % of all instances solved the run time and memory overhead is at most 10 % for UC extraction with sets of time points using Gawrychowski's algorithm over UC extraction without sets of time points. An analysis by category (for plots see Appendix C) shows that the corresponding maximum run time (resp., memory) overhead for instances of the **application** category is at most 50 % (resp., 111 %).

Gawrychowski's algorithm [Gaw11] has better worst case complexity than Sawa's algorithm [Saw13]. We also found it easier to understand and implement. Figure 8 (c) and (d) compare using Gawrychowski's and Sawa's algorithm for computing sets of time points. On our benchmarks Gawrychowski's algorithm tends to perform better than Sawa's algorithm, especially when the NFAs become larger.

All in all our experiments show that in many cases with a moderate overhead in run time and memory usage one can obtain sets of time points that provide interesting additional information for a UC.

8. Discussion

8.1. On the Notion of UCs with Sets of Time Points

Below we discuss some aspects related to our notion of UCs with sets of time points. Note that this paper is a first step in enhancing UCs for LTL with sets of time points. We therefore restrict ourselves to a notion of UCs for LTL with sets of time points that computes a single, non-minimal UC with sets of time points based on a single proof of unsatisfiability. Such notion can be expected to be required as an input to efficient solutions for some of the potential shortcomings mentioned below. As it turns out, it can also be computed with little overhead from a proof of unsatisfiability, which is assumed to be carried out anyway. Overcoming potential shortcomings is left as future work. Note also that the aspects below are mostly not specific to the enhancement of UCs for LTL sets of time points; rather, they apply to many notions of UC.

A notion of explanation of unsatisfiability for LTL formulas cannot disregard syntax completely. By definition every unsatisfiable LTL formula has the same, empty set of satisfying assignments. Therefore, any notion of explanation of unsatisfiability for LTL formulas that were *solely* based on the semantics of the unsatisfiable formula *as a whole*

would be the same for all unsatisfiable LTL formulas. Hence, while this doesn't rule out that semantical aspects are taken into account when coming up with a notion of explanation of unsatisfiability for LTL formulas, the syntax of an unsatisfiable LTL formula will have some role to play in a non-trivial notion of explanation of unsatisfiability for LTL formulas. For a brief account of attempting an approach that is "blind to syntax" and then discovering the importance of syntax for explanations in description logics see [Hor11], p. 153.

There are properties tied to the syntax of an unsatisfiable LTL formula that are worth pointing out. Consider, for example, a formula of the form $\phi \equiv \psi \wedge \psi$. Even if a perfect explanation were provided why ψ is unsatisfiable, it would likely be worth pointing out to the user that with respect to the unsatisfiability of ϕ one of the two copies of ψ is redundant. Note that in a realistically sized real world specification such redundancy might not be so obvious. In fact, in formal verification vacuity is concerned with pointing out redundancies in specifications (e.g., [BBDER01, KV03, AFF+03, GC04, FKSFV08, Kup06]), and vacuity is closely related to UCs for LTL [Sch12b, Sch15].

An explanation of an unsatisfiable LTL formula should be linked to the input as provided by the user. As stated in Sec. 4.2 an explanation of an unsatisfiable LTL formula should be provided in a form that allows the user to make the connection between the explanation and the input formula she provided. It is therefore frequent practice to map back a UC obtained in some lower level formalism to the input or something close to the input that the user provided (e.g., [SSJ+03, Sch12b, Kal06]). Similarly, when proofs are used as explanations, it is recommended by some not to use proof calculi for proof presentation that transform the input provided by the user too much; an example for this are resolution calculi relying on a transformation into a clausal normal form (e.g., [Bun99, GG13]).

The UCs with sets of time points obtained with our implementation are sensitive to syntax. As discussed above it is probably inevitable that syntax plays a role in determining a UC of an unsatisfiable LTL formula ϕ. We now show an instance of sensitivity to syntax in our implementation: applying something apparently innocuous such as commutativity of conjunction in ϕ may lead to a different UC with different sets of time points. The TR algorithm stops as soon as the empty clause has been derived in the main partition. Hence, the order in which SNF clauses are considered has an influence on which proof will be obtained. Because the syntax of ϕ determines that order, the UC obtained depends on the syntax of ϕ. Using the "right" options our implementation produces (31) as a UC for (30) and (33) as a UC for (32). A potential remedy is producing more than one UC. As syntax will have to play some role, normalization of the input formula is likely to be only a partial remedy.[13]

$$(\mathbf{G}(p \wedge q) \wedge (\neg p)) \wedge \mathbf{X}\neg q \tag{30}$$

$$(\underset{\{1\}}{\mathbf{G}}\, q) \underset{\{0\},\{0\}}{\wedge} \underset{\{1\}}{\mathbf{X}} \underset{\{1\}}{\neg}\, q \tag{31}$$

$$(\mathbf{G}(p \wedge q) \wedge (\mathbf{X}\neg q)) \wedge \neg p \tag{32}$$

$$(\underset{\{0\}}{\mathbf{G}}\, p) \underset{\{0\},\{0\}}{\wedge} \underset{\{0\}}{\neg}\, p \tag{33}$$

We restrict ourselves to non-minimal UCs in this paper. The fact that some part of an LTL formula ϕ is used at some time point in a proof of unsatisfiability of ϕ does not mean that that part of ϕ at that time point is necessary to prove unsatisfiability of ϕ. To obtain a minimal UC one can often use deletion-based extraction of minimal UCs (e.g., [CD91, BDTW93, Zha03, MS12, Sch15]). The method repeatedly attempts to remove parts of a UC. The modified UC is tested for satisfiability. If it is unsatisfiable, then the removal is made permanent; if not, it is undone. While this is simple to implement, the additional satisfiability tests may be costly. Hence, often a two step process is employed where first a potentially non-minimal UC is obtained using some cheap means (such as extraction from a proof of unsatisfiability), which is then minimized with deletion-based extraction of UCs (see, e.g., [CD91, BDTW93, Zha03, MS12, Sch15]). More advanced solutions exist for computing minimally unsatisfiable sets of clauses in Boolean satisfiability (see [KBK09] for references).

While some of these approaches might carry over to computing minimal sets of unsatisfiable SNF clauses, it is unclear whether any of them can help to compute minimal sets of time points to annotate a UC in LTL with. When

[13] In fact, using default options our implementation performs some normalization and produces the same UC for both (30) and (32).

considering to minimize sets of time points there are at least two candidate notions of minimality. As shown in Prop. 7 the sets of time points we obtain are semilinear sets, i.e., finite sets of linear sets. Therefore, one notion of minimality could minimize the number of linear sets in each semilinear set. This could be implemented, e.g., with deletion-based extraction of UCs using Prop. 6. This notion requires semilinear sets that are represented by more than one linear set to get started. Another notion of minimality could minimize the number of natural numbers in each set of time points. Notice that now we are possibly dealing with infinitely many candidates for removal; hence, simple deletion-based extraction of UCs may not be applicable. Notions of minimum UCs are likely to be more involved both conceptually and computationally.

We restrict ourselves to a single UC in this paper. Clearly, there is no guarantee that a single UC obtained from one particular proof of the unsatisfiability of ϕ is representative for all UCs of ϕ. A common remedy in other logics is therefore to compute all UCs for a given formula (e.g., [LS08, KPHS07]). Some work classifies parts of UCs according to how often they occur in UCs (e.g., [KLM06, HPS10a]).

8.2. Display of UCs with Sets of Time Points

Judging from our experience with our implementation, which provides a command line interface only, sets of time points should not be too dominant in the display of a UC for LTL with sets of time points. Instead, in a GUI the display of the UC should be guided by the "normal" LTL part of the UC with sets of time points subordinate to the "normal" LTL part. To help a user who is debugging a UC focus, it could be useful to allow the user to selectively switch the display of sets of time points on and off for parts of the UC. Another possibility could be to develop a heuristic that tries to distinguish interesting from uninteresting sets of time points and displays only the interesting ones. Some criteria for a set of time points to be classified as interesting can be derived from the discussion of our experimental results in Sec. 7. An example criterion for a set of time points to be deemed uninteresting could be the fact that it can be easily derived from the set of time points of a superformula: for example, in $\underset{I}{\circ_1} \underset{I'}{\neg} \psi$, where $\circ_1$ is a unary operator, it makes no sense for I and I' to be different (the case of a binary operator is analogous).

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 other algorithms, be they tableau-based (e.g., [HH11]), BDD-based (e.g., [BCM+92, CGH97] and possibly utilizing [SB06, JSB06]), or SAT-based (see, e.g., [Bie09]). One could investigate obtaining sets of time points by solving a system of constraints over sets of time points based on Lemmas 2, 3 rather than the approach based on Parikh images explored here. It would also 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). LTL realizability (e.g., [PR89]) asks whether in a two player game one player (a system to be implemented) has a strategy against the other player (the environment that the system should be able to cope with) such that every play satisfies an LTL formula; if no such strategy exists, then an unrealizable core can be computed (e.g., [Sch12b]). Possibly, sets of time points could be used to enhance such unrealizable cores. One could also investigate fine-grained notions of UCs for branching time temporal logics. Finally, note that in general improving the granularity of a notion of UC may be interesting when a notion of UC is based on removing complex constraints from a set of constraints or complex statements from a Boolean combination of statements without proceeding to simplify the complex constraints or statements themselves (for examples see LTL (e.g., [AGH+12, GHST13, CRST07]) or SMT (e.g., [CGS11])).

Acknowledgments

I am grateful to Boris Konev and Michel Ludwig for making TRP++ and TSPASS (which are the basis of this paper) including their LTL translators available and for answering my questions. I thank Alessandro 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. I also thank the reviewers of the current and previous iterations of this article for their helpful feedback. Initial parts of the work were performed while working under a grant by the Provincia Autonoma di Trento (project EMTELOS).

References

[AFF+03] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. Enhanced Vacuity Detection in Linear Temporal Logic. In W. Hunt Jr. and F. Somenzi, editors, *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2003. doi:10.1007/978-3-540-45069-6_35. 2, 38

[AGH+12] A. Awad, R. Goré, Z. Hou, J. Thomson, and M. Weidlich. An iterative approach to synthesize business process templates from compliance rules. *Inf. Syst.*, 37(8):714–736, 2012. doi:10.1016/j.is.2012.05.001. 1, 2, 3, 39

[Apt03] K. Apt. *Principles of constraint programming*. Cambridge University Press, 2003. 5

[BBDC+09] I. Beer, S. Ben-David, H. Chockler, A. Orni, and R. Trefler. Explaining Counterexamples Using Causality. In A. Bouajjani and O. Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2009. doi:10.1007/978-3-642-02658-4_11. 2, 4

[BBDER01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient Detection of Vacuity in Temporal Model Checking. *Formal Methods in System Design*, 18(2):141–163, 2001. doi:10.1023/A:1008779610539. 1, 2, 38

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In R. Cleaveland, editor, *TACAS*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999. doi:10.1007/3-540-49059-0_14. 4

[BCM+92] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *Inf. Comput.*, 98(2):142–170, 1992. doi:10.1016/0890-5401(92)90017-A. 9, 10, 39

[BCM+07] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2007. 5

[BDF09] A. Behdenna, C. Dixon, and M. Fisher. Deductive Verification of Simple Foraging Robotic Behaviours. *International Journal of Intelligent Computing and Cybernetics*, 2(4):604–643, 2009. doi:10.1108/17563780911005818. 35

[BDTW93] R. Bakker, F. Dikker, F. Tempelman, and P. Wognum. Diagnosing and Solving Over-Determined Constraint Satisfaction Problems. In *IJCAI*, pages 276–281, 1993. URL: http://ijcai.org/Past%20Proceedings/IJCAI-93-VOL1/PDF/039.pdf. 1, 21, 38

[BF99] A. Bolotov and M. Fisher. A clausal resolution method for CTL branching-time temporal logic. *J. Exp. Theor. Artif. Intell.*, 11(1):77–93, 1999. doi:10.1080/095281399146625. 10

[BG01] L. Bachmair and H. Ganzinger. Resolution Theorem Proving. In Robinson and Voronkov [RV01], pages 19–99. 10, 17

[BGJ+07] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, Compile, Run: Hardware from PSL. In S. Glesner, J. Knoop, and R. Drechsler, editors, *COCV*, volume 190(4) of *ENTCS*, pages 3–16. Elsevier, 2007. doi:10.1016/j.entcs.2007.09.004. 1, 7, 35

[BH95] F. Baader and B. Hollunder. Embedding Defaults into Terminological Knowledge Representation Formalisms. *J. Autom. Reasoning*, 14(1):149–180, 1995. doi:10.1007/BF00883932. 5

[BHvMW09] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. 37, 40, 41, 42

[Bie09] A. Biere. Bounded Model Checking. In Biere et al. [BHvMW09], pages 457–481. doi:10.3233/978-1-58603-929-5-457. 4, 39

[BJ] A. Biere and T. Jussila. Benchmark tool run. URL: http://fmv.jku.at/run/. 35

[boo] http://www.boost.org/doc/libs/release/libs/graph/. 34

[BS01] J. Bradfield and C. Stirling. Modal Logics and mu-Calculi: An Introduction. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 293–330. Elsevier, 2001. 4

[Bun99] A. Bundy. A Survey of Automated Deduction. In M. Wooldridge and M. Veloso, editors, *Artificial Intelligence Today*, volume 1600 of *Lecture Notes in Computer Science*, pages 153–174. Springer, 1999. doi:10.1007/3-540-48317-9_6. 38

[CCG+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 E. Brinksma and K. Larsen, editors, *CAV*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002. doi:10.1007/3-540-45657-0_29. 4

[CCM+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 J. Kramer, J. Bishop, P. Devanbu, and S. Uchitel, editors, *ICSE (2)*, pages 109–118. ACM, 2010. doi:10.1145/1810295.1810312. 1

[CD91] J. Chinneck and E. Dravnieks. Locating Minimal Infeasible Constraint Sets in Linear Programs. *INFORMS Journal on Computing*, 3(2):157–168, 1991. doi:10.1287/ijoc.3.2.157. 1, 21, 38

[CGH97] E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. *Formal Methods in System Design*, 10(1):47–71, 1997. doi:10.1023/A:1008615614281. 9, 39

[CGP01] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2001. 10

[CGS11] A. Cimatti, A. Griggio, and R. Sebastiani. Computing Small Unsatisfiable Cores in Satisfiability Modulo Theories. *J. Artif. Intell. Res. (JAIR)*, 40:701–728, 2011. URL: http://jair.org/papers/paper3196.html. 1, 39

[Cla07] K. Claessen. A Coverage Analysis for Safety Property Lists. In *FMCAD*, pages 139–145. IEEE Computer Society, 2007. doi:10.1109/FAMCAD.2007.32. 4

[CRST07] A. Cimatti, M. Roveri, V. Schuppan, and S. Tonetta. Boolean Abstraction for Temporal Logic Satisfiability. In W. Damm and H. Hermanns, editors, *CAV*, volume 4590 of *Lecture Notes in Computer Science*, pages 532–546. Springer, 2007. doi:10.1007/978-3-540-73368-3_53. 4, 39

[CTVW03] E. Clarke, M. Talupur, H. Veith, and D. Wang. SAT Based Predicate Abstraction for Hardware Verification. In E. Giunchiglia and A. Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2003. doi:10.1007/978-3-540-24605-3_7. 4

[Den10] X. Deng. *Explanation and Diagnosis Services for Unsatisfiability and Inconsistency in Description Logics*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, 2010. URL: http://spectrum.library.concordia.ca/979526/. 5

[DHS05] X. Deng, V. Haarslev, and N. Shiri. A Framework for Explaining Reasoning in Description Logics. In T. Roth-Berghofer and S. Schulz, editors, *Explanation-Aware Computing, Papers from the 2005 AAAI Fall Symposium, November 4-6, 2005, Arlington, Virginia*, volume FS-05-04 of *AAAI Technical Report*, pages 55–61. AAAI Press, 2005. 5

[Dix95] C. Dixon. *Strategies for Temporal Resolution*. PhD thesis, Department of Computer Science, University of Manchester, 1995. URL: http://apt.cs.manchester.ac.uk/ftp/pub/TR/UMCS-95-12-1.ps.Z. 10, 12

[Dix96] C. Dixon. Search Strategies for Resolution in Temporal Logics. In M. McRobbie and J. Slaney, editors, *CADE*, volume 1104 of *Lecture Notes in Computer Science*, pages 673–687. Springer, 1996. doi:10.1007/3-540-61511-3_121. 10

[Dix97] C. Dixon. Using Otter for Temporal Resolution. In H. Barringer, M. Fisher, D. Gabbay, and G. Gough, editors, *ICTL*, volume 16 of *Applied Logic Series*, pages 149–166. Springer, 1997. doi:10.1007/978-94-015-9586-5_8. 5, 10, 12

[Dix98] C. Dixon. Temporal Resolution Using a Breadth-First Search Algorithm. *Ann. Math. Artif. Intell.*, 22(1-2):87–115, 1998. doi:10.1023/A:1018942108420. 4, 5, 10, 12

[DRS03] Y. Dong, C. Ramakrishnan, and S. Smolka. Model Checking and Evidence Exploration. In *ECBS*, pages 214–223. IEEE Computer Society, 2003. doi:10.1109/ECBS.2003.1194802. 4

[DSRS02] Y. Dong, B. Sarna-Starosta, C. Ramakrishnan, and S. Smolka. Vacuity Checking in the Modal Mu-Calculus. In H. Kirchner and C. Ringeissen, editors, *AMAST*, volume 2422 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2002. doi:10.1007/3-540-45719-4_11. 4

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

[Eme90] E. Emerson. Temporal and Modal Logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics (B)*, pages 995–1072. Elsevier and MIT Press, 1990. 1, 8

[FDP01] M. Fisher, C. Dixon, and M. Peim. Clausal temporal resolution. *ACM Trans. Comput. Log.*, 2(1):12–56, 2001. doi:10.1145/371282.371311. 4, 5, 9, 10, 12, 17

[Fie01] A. Fiedler. *User-Adaptive Proof Explanation*. PhD thesis, Naturwissenschaftlich-Technische Fakultät I, Universität des Saarlandes, 2001. URL: http://scidok.sulb.uni-saarland.de/volltexte/2004/182/. 5

[Fis91] M. Fisher. A Resolution Method for Temporal Logic. In *IJCAI*, pages 99–104, 1991. URL: http://ijcai.org/Past%20Proceedings/IJCAI-91-VOL1/PDF/017.pdf. 4, 9, 10

[FKSFV08] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Vardi. A Framework for Inherent Vacuity. In H. Chockler and A. Hu, editors, *HVC*, volume 5394 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2008. doi:10.1007/978-3-642-01702-5_7. 2, 38

[FM09] J. Franco and J. Martin. A History of Satisfiability. In Biere et al. [BHvMW09], pages 3–74. doi:10.3233/978-1-58603-929-5-3. 10

[FN92] M. Fisher and P. Noël. Transformation and Synthesis in MetateM. Part I: Propositional MetateM. Technical Report UMCS-92-2-1, University of Manchester, Department of Computer Science, 1992. URL: http://apt.cs.manchester.ac.uk/ftp/pub/TR/UMCS-92-2-1.ps.Z. 9

[Gaw11] P. Gawrychowski. Chrobak Normal Form Revisited, with Applications. In B. Bouchou-Markhoff, P. Caron, J. Champarnaud, and D. Maurel, editors, *CIAA*, volume 6807 of *Lecture Notes in Computer Science*, pages 142–153. Springer, 2011. doi:10.1007/978-3-642-22256-6_14. 31, 32, 34, 37, 45

[GC04] A. Gurfinkel and M. Chechik. How Vacuous Is Vacuous? In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 451–466. Springer, 2004. doi:10.1007/978-3-540-24730-2_34. 2, 38

[GG13] M. Ganesalingam and W. Gowers. A fully automatic problem solver with human-style output, 2013. Available at arXiv:1309.4501 [cs.AI]. 38

[GHST13] R. Goré, J. Huang, T. Sergeant, and J. Thomson. Finding Minimal Unsatisfiable Subsets in Linear Temporal Logic using BDDs, 2013. Available from http://www.timsergeant.com/files/pltlmup/gore_huang_sergeant_thomson_mus_pltl.pdf. 3, 39

[GMP07] É. Grégoire, B. Mazure, and C. Piette. MUST: Provide a Finer-Grained Explanation of Unsatisfiability. In C. Bessiere, editor, *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 317–331. Springer, 2007. doi:10.1007/978-3-540-74970-7_24. 5

[GN03] E. Goldberg and Y. Novikov. Verification of Proofs of Unsatisfiability for CNF Formulas. In *DATE*, pages 10886–10891. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10008. 1, 21

[GSW89] R. Greiner, B. Smith, and R. Wilkerson. A Correction to the Algorithm in Reiter's Theory of Diagnosis. *Artif. Intell.*, 41(1):79–88, 1989. doi:10.1016/0004-3702(89)90079-9. 4

[Har05] A. Harding. *Symbolic Strategy Synthesis For Games With LTL Winning Conditions*. PhD thesis, University of Birmingham, 2005. 6, 35

[HH11] F. Hantry and M. Hacid. Handling Conflicts in Depth-First Search for LTL Tableau to Debug Compliance Based Languages. In E. Pimentel and V. Valero, editors, *FLACOS*, volume 68 of *EPTCS*, pages 39–53, 2011. doi:10.4204/EPTCS.68.5. 3, 39

[HHT11] F. Hantry, M. Hacid, and R. Thion. Detection of Conflicting Compliance Rules. In *EDOCW*, pages 419–428. IEEE Computer Society, 2011. doi:10.1109/EDOCW.2011.57. 47

[HJSS95] A. Heuerding, G. Jäger, S. Schwendimann, and M. Seyfried. Propositional Logics on the Computer. In P. Baumgartner, R. Hähnle, and J. Posegga, editors, *TABLEAUX*, volume 918 of *Lecture Notes in Computer Science*, pages 310–323. Springer, 1995. doi:10.1007/3-540-59338-1_44. 4

[HK03] U. Hustadt and B. Konev. TRP++ 2.0: A Temporal Resolution Prover. In F. Baader, editor, *CADE*, volume 2741 of *Lecture Notes in Computer Science*, pages 274–278. Springer, 2003. doi:10.1007/978-3-540-45085-6_21. 4, 5, 10, 12, 17

[HK04] U. Hustadt and B. Konev. TRP++: A temporal resolution prover. In M. Baaz, J. Makowsky, and A. Voronkov, editors, *Collegium Logicum*, volume 8, pages 65–79. Kurt Gödel Society, 2004. 4, 5, 10, 12

[Hoo99] H. Hoos. Heavy-Tailed Behaviour in Randomised Systematic Search Algorithms for SAT? Technical Report 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. 1

[Hor11] M. Horridge. *Justification Based Explanation in Ontologies*. PhD thesis, School of Computer Science, Faculty of Engineering and Physical Sciences, University of Manchester, 2011. URL: http://www.escholar.manchester.ac.uk/uk-ac-man-scw:131699. 5, 38

[HPS08] M. Horridge, B. Parsia, and U. Sattler. Laconic and Precise Justifications in OWL. In A. Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thirunarayan, editors, *ISWC*, volume 5318 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2008. doi:10.1007/978-3-540-88564-1_21. 5

[HPS10a] M. Horridge, B. Parsia, and U. Sattler. Justification Masking in OWL. In V. Haarslev, D. Toman, and G. Weddell, editors, *DL*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010. URL: http://ceur-ws.org/Vol-573/paper_31.pdf. 39

[HPS10b] M. Horridge, B. Parsia, and U. Sattler. Justification Oriented Proofs in OWL. In P. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Pan, I. Horrocks, and B. Glimm, editors, *ISWC (I)*, volume 6496 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2010. doi:10.1007/978-3-642-17746-0_23. 5

[HR83] J. Halpern and J. Reif. The Propositional Dynamic Logic of Deterministic, Well-Structured Programs. *Theor. Comput. Sci.*, 27:127–165, 1983. doi:10.1016/0304-3975(83)90097-X. 8

[HS02] U. Hustadt and R. A. Schmidt. Scientific Benchmarking with Temporal Logic Decision Procedures. In D. Fensel, F. Giunchiglia, D. McGuinness, and M. Williams, editors, *KR*, pages 533–546. Morgan Kaufmann, 2002. 35

[HSH12] F. Hantry, L. Saïs, and M. Hacid. On the complexity of computing minimal unsatisfiable LTL formulas. *Electronic Colloquium on Computational Complexity (ECCC)*, 19(69), 2012. URL: http://eccc.hpi-web.de/report/2012/069. 3

[HU79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979. 31, 34

[Hua94] X. Huang. Reconstructing Proofs at the Assertion Level. In A. Bundy, editor, *CADE*, volume 814 of *Lecture Notes in Computer Science*, pages 738–752. Springer, 1994. doi:10.1007/3-540-58156-1_53. 5

[Jos12] N. Josuttis. *The C++ Standard Library: A Tutorial and Reference*. Addison-Wesley, 2 edition, 2012. 34

[JSB06] T. Jussila, C. Sinz, and A. Biere. Extended Resolution Proofs for Symbolic SAT Solving with Quantification. In A. Biere and C. Gomes, editors, *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 54–60. Springer, 2006. doi:10.1007/11814948_8. 39

[Kal06] A. Kalyanpur. *Debugging and Repair of OWL Ontologies*. PhD thesis, University of Maryland, College Park, 2006. URL: http://hdl.handle.net/1903/3820. 5, 38

[KBK09] H. Kleine Büning and O. Kullmann. Minimal Unsatisfiability and Autarkies. In Biere et al. [BHvMW09], pages 339–401. doi:10.3233/978-1-58603-929-5-339. 38

[KLM06] O. Kullmann, I. Lynce, and J. Marques-Silva. Categorisation of Clauses in Conjunctive Normal Forms: Minimally Unsatisfiable Sub-clause-sets and the Lean Kernel. In A. Biere and C. Gomes, editors, *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 22–35. Springer, 2006. doi:10.1007/11814948_4. 39

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. *Theor. Comput. Sci.*, 27:333–354, 1983. doi:10.1016/0304-3975(82)90125-6. 4

[KPHS07] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding All Justifications of OWL DL Entailments. In K. Aberer, K.-S. Choi, N. Fridman Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, editors, *ISWC*, volume 4825 of *Lecture Notes in Computer Science*, pages 267–280. Springer, 2007. doi:10.1007/978-3-540-76298-0_20. 39

[Kup06] O. Kupferman. Sanity Checks in Formal Verification. In C. Baier and H. Hermanns, editors, *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006. doi:10.1007/11817949_3. 1, 2, 38

[KV03] O. Kupferman and M. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003. doi:10.1007/s100090100062. 2, 38

[Lin89] C. Lingenfelder. Structuring Computer Generated Proofs. In N. Sridharan, editor, *IJCAI*, pages 378–383. Morgan Kaufmann, 1989. URL: http://ijcai.org/Past%20Proceedings/IJCAI-89-VOL1/PDF/060.pdf. 5

[LP85] O. Lichtenstein and A. Pnueli. Checking That Finite State Concurrent Programs Satisfy Their Linear Specification. In M. Van Deusen, Z. Galil, and B. Reid, editors, *POPL*, pages 97–107. ACM Press, 1985. doi:10.1145/318593.318622. 9

[LS08] M. Liffiton and K. Sakallah. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *J. Autom. Reasoning*, 40(1):1–33, 2008. doi:10.1007/s10817-007-9084-z. 39

[MS12] J. Marques-Silva. Computing Minimally Unsatisfiable Subformulas: State of the Art and Future Directions. *Multiple-Valued Logic and Soft Computing*, 19(1-3):163–183, 2012. 38

[Ngu13] T. Nguyen. *Generating Natural Language Explanations For Entailments In Ontologies*. PhD thesis, The Open University, 2013. URL: http://oro.open.ac.uk/id/eprint/39116. 5

[NPPW12] T. Nguyen, R. Power, P. Piwek, and S. Williams. Planning Accessible Explanations for Entailments in OWL Ontologies. In B. Di Eugenio, S. McRoy, A. Gatt, A. Belz, A. Koller, and K. Striegnitz, editors, *INLG*, pages 110–114. The Association for Computer Linguistics, 2012. URL: http://www.aclweb.org/anthology/W12-1518. 5

[Par66] R. Parikh. On Context-Free Languages. *J. ACM*, 13(4):570–581, 1966. doi:10.1145/321356.321364. 8, 25, 28

[PG86] D. Plaisted and S. Greenbaum. A Structure-Preserving Clause Form Translation. *J. Symb. Comput.*, 2(3):293–304, 1986. doi:10.1016/S0747-7171(86)80028-1. 9

[plt] http://users.cecs.anu.edu.au/~rpg/PLTLProvers/. 4

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57. IEEE, 1977. doi:10.1109/SFCS.1977.32. 1

[PQ13] I. Pill and T. Quaritsch. Behavioral Diagnosis of LTL Specifications at Operator Level. In F. Rossi, editor, *IJCAI*. IJCAI/AAAI, 2013. URL: http://ijcai.org/papers13/Papers/IJCAI13-160.pdf. 4

[PR89] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293. 39

[PSC+06] I. Pill, S. Semprini, R. Cavada, M. Roveri, R. Bloem, and A. Cimatti. Formal analysis of hardware requirements. In E. Sentovich, editor, *DAC*, pages 821–826. ACM, 2006. doi:10.1145/1146909.1147119. 1, 2

[PvdA06] M. Pesic and W. van der Aalst. A Declarative Approach for Flexible Business Processes Management. In J. Eder and S. Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006. doi:10.1007/11837862_18. 1

[Rei87] R. Reiter. A Theory of Diagnosis from First Principles. *Artif. Intell.*, 32(1):57–95, 1987. doi:10.1016/0004-3702(87)90062-2. 4

[Rob65] J. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965. doi:10.1145/321250.321253. 10

[RS04] K. Ravi and F. Somenzi. Minimal Assignments for Bounded Model Checking. In K. Jensen and A. Podelski, editors, *TACAS*, volume 2988 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004. doi:10.1007/978-3-540-24730-2_3. 2, 4

[RV01] J. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001. 5, 40

[RV10] K. Rozier and M. Vardi. LTL satisfiability checking. *STTT*, 12(2):123–137, 2010. doi:10.1007/s10009-010-0140-3. 1, 35

[Sal73] A. Salomaa. *Formal Languages*. Academic Press, 1973. 8

[Saw13] Z. Sawa. Efficient Construction of Semilinear Representations of Languages Accepted by Unary Nondeterministic Finite Automata. *Fundam. Inform.*, 123(1):97–106, 2013. doi:10.3233/FI-2013-802. 34, 37

[SB06] C. Sinz and A. Biere. Extended Resolution Proofs for Conjoining BDDs. In D. Grigoriev, J. Harrison, and E. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2006. doi:10.1007/11753728_60. 39

[SC85] A. Sistla and E. Clarke. The Complexity of Propositional Linear Temporal Logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837. 8

[SC03] S. Schlobach and R. Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In G. Gottlob and T. Walsh, editors, *IJCAI*, pages 355–362. Morgan Kaufmann, 2003. URL: http://ijcai.org/Past%20Proceedings/IJCAI-2003/PDF/053.pdf. 5

[Sch12a] V. Schuppan. Extracting unsatisfiable cores for LTL via temporal resolution, 2012. Available at arXiv:1212.3884v1 [cs.LO]. 10

[Sch12b] V. Schuppan. Towards a notion of unsatisfiable and unrealizable cores for LTL. *Sci. Comput. Program.*, 77(7-8):908–939, 2012. doi:10.1016/j.scico.2010.11.004. 1, 2, 3, 4, 20, 21, 22, 27, 33, 38, 39, 47

[Sch13] V. Schuppan. Enhancing Unsatisfiable Cores for LTL with Information on Temporal Relevance. In L. Bortolussi and H. Wiklicky, editors, *QAPL*, volume 117 of *EPTCS*, pages 49–65, 2013. doi:10.4204/EPTCS.117.4. 1, 34

[Sch15] V. Schuppan. Extracting Unsatisfiable Cores for LTL via Temporal Resolution. *Acta Inf.*, 53(3):247–299, 2016. doi:10.1007/s00236-015-0242-1. 2, 3, 4, 5, 6, 10, 12, 13, 14, 15, 17, 18, 19, 20, 29, 32, 33, 34, 38

[SD11] V. Schuppan and L. Darmawan. Evaluating LTL Satisfiability Solvers. In T. Bultan and P. Hsiung, editors, *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 397–413. Springer, 2011. doi:10.1007/978-3-642-24372-1_28. 4, 35

[SDGC10] J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. Exploiting resolution proofs to speed up LTL vacuity detection for BMC. *STTT*, 12(5):319–335, 2010. doi:10.1007/s10009-009-0134-1. 2, 4

[SHB+99] J. Siekmann, S. Hess, C. Benzmüller, L. Cheikhrouhou, A. Fiedler, H. Horacek, M. Kohlhase, K. Konrad, A. Meier, E. Melis, M. Pollet, and V. Sorge. $\mathcal{L}\Omega\mathcal{UI}$: $\mathcal{L}$ovely ΩMEGA $\mathcal{U}$ser $\mathcal{I}$nterface. *Formal Asp. Comput.*, 11(3):326–342, 1999. doi:10.1007/s001650050053. 5

[SL95] A. Stepanov and M. Lee. The Standard Template Library. Technical Report 95-11(R.1), HP Laboratories, 11 1995. URL: http://www.stepanovpapers.com/STL/DOC.PDF. 34

[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. 34

[SSJ+03] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *ASE*, pages 94–105. IEEE Computer Society, 2003. doi:10.1109/ASE.2003.1240298. 4, 38

[SVW87] A. Sistla, M. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theor. Comput. Sci.*, 49:217–237, 1987. doi:10.1016/0304-3975(87)90008-9. 8

[Tar72] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972. doi:10.1137/0201010. 45

[TCJ08] E. Torlak, F. Sheng-Ho Chang, and D. Jackson. Finding Minimal Unsatisfiable Cores of Declarative Specifications. In J. Cuéllar, T. Maibaum, and K. Sere, editors, *FM*, volume 5014 of *Lecture Notes in Computer Science*, pages 326–341. Springer, 2008. doi:10.1007/978-3-540-68237-0_23. 1

[trp] http://www.csc.liv.ac.uk/~konev/software/trp++/. 4, 5, 10

[WDMR08] M. De Wulf, L. Doyen, N. Maquet, and J. Raskin. Antichains: Alternative Algorithms for LTL Satisfiability and Model-Checking. In C. Ramakrishnan and J. Rehof, editors, *TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 63–77. Springer, 2008. doi:10.1007/978-3-540-78800-3_6. 35

[Wol83] P. Wolper. Temporal Logic Can Be More Expressive. *Information and Control*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5. 3, 22

[Zha03] L. Zhang. *Searching for Truth: Techniques for Satisfiability of Boolean Formulas*. PhD thesis, Department of Electrical Engineering, Princeton University, 2003. 21, 38

[ZM03] L. Zhang and S. Malik. Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. In *DATE*, pages 10880–10885. IEEE Computer Society, 2003. doi:10.1109/DATE.2003.10014. 1

Appendix A. Gawrychowski's Algorithm Extended to Handle Multiple Final Vertices in Parallel

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 $O(|V'|^3 + |C^{uc}| \cdot |V'|^2 + |V'| \cdot |E''|)$ we show the main part of the algorithm in Alg. 1a–1c. An explanation of how it works is out of the scope of this paper; for that, please refer to [Gaw11]. Notice that the 3 parts of the algorithm are intended to be executed in order Alg. 1a, Alg. 1b, Alg. 1c; below Alg. 1c is included before Alg. 1b only to improve page layout. The parts that are affected by handling multiple final vertices in parallel are as follows: (i) in Alg. 1a: line 5, (ii) in Alg. 1b: lines 9, 45–48, and (iii) in Alg. 1c: lines 13–15. These lines are marked blue and prefixed with a * below.

The first part (Alg. 1a) 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 $O(|V'| \cdot (|V'| + |E''|) + |C^{uc}|)$. The second part (Alg. 1b) processes non-trivial SCCs. It can be carried out in time $O(|V'|^3 + |C^{uc}| \cdot |V'|^2 + |V'| \cdot |E''|)$. 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. 1c) processes trivial SCCs. It can be carried out in time $O(|V'|^2 + |C^{uc}| \cdot |V'| + |V'| \cdot |E''|)$. Hence, the overall time to compute Parikh images is $O(|V'|^3 + |C^{uc}| \cdot |V'|^2 + |V'| \cdot |E''|)$.

Algorithm 1a: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. First part: preparation.

```
1   Partition 𝒜 into SCCs;                              /* Tarjan's algorithm: O(|V'| + |E''|) */
2   foreach SCC A do                                                                /* O(|V'|) */
3     └ Find the length of the shortest loop in A;  /* BFS from each vertex: O(|V'|·(|V'| + |E''|)) */
4   Create an empty set of vertices sforbidden;                                     /* O(|V'|) */
*5  Create a list of vertices lfinal and assign it the list of vertices L_V-labeled with clauses in C^uc;   /* O(|C^uc|) */
```

Algorithm 1c: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. Third part: processing trivial SCCs.

```
1   Create a set of vertices scurr; scurr ← {v□};                                   /* O(|V'|) */
2   Create an empty set of vertices snext;                                          /* O(|V'|) */
3   Create a list of vertices lcurr; lcurr ← [v□];                                  /* O(1) */
4   Create an empty list of vertices lnext;                                         /* O(1) */
5   i ← 0;                                                                          /* O(1) */
6   while lcurr ≠ [] do          /* At most |V'| forward iterations in a graph of trivial SCCs: O(|V'|) */
7     foreach v ∈ lcurr do                                                          /* O(|V'|^2) */
8       foreach target vertex v' of each outgoing edge of v do                      /* O(|V'|·|E''|) */
9         if v' ∈ sforbidden then continue;                                         /* O(|V'|·|E''|) */
10        if v' ∉ snext then                                                        /* O(|V'|·|E''|) */
11          snext ← snext ∪ {v'};                                                   /* O(|V'|^2) */
12          lnext ← lnext ∘ [v'];                                                   /* O(|V'|^2) */

*13   foreach v ∈ lfinal do                                                         /* O(|C^uc|·|V'|) */
*14     if v ∈ snext then                                                           /* O(|C^uc|·|V'|) */
*15       Add 0·ℕ + i + 1 to the Parikh image of L_V(v);                            /* O(|C^uc|·|V'|) */

16    scurr ← snext; snext ← ∅;                                                     /* O(|V'|^2) */
17    lcurr ← lnext; lnext ← [];                                                    /* O(|V'|) */
18    i ← i + 1;                                                                    /* O(|V'|) */
```

Algorithm 1b: Gawrychowski's algorithm extended to handle multiple final vertices in parallel. Second part: processing non-trivial SCCs.

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

Appendix B. An Example from the Business Process Domain

The example (B.1) shows applicability and utility of our approach in the business process domain. It is based on example 3 in [HHT11]. We changed (B.1c) from $\mathbf{F}(i \land nr)$ 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]. (B.1a): An order (o) must occur. (B.1b): A payment (p) with non-repudiation (nr) must occur. (B.1c): An insurance submission (i) with non-repudiation must occur at time point 5. (B.1d): A goods delivery (g) must occur. (B.1e): Insurance before payment (p) is forbidden. (B.1f): If a payment occurs, it must occur at least three time points after the order. (B.1g): Goods delivery before payment is forbidden. (B.1h): If an insurance submission occurs, it must occur either at the time point of the order or one time point later. And (B.1i): A golden ($gold$) customer must have goods delivered no later than three time points after the time point at which the payment is accomplished.

$$(\mathbf{F}o) \tag{B.1a}$$
$$\land\ (\mathbf{F}(p \land nr)) \tag{B.1b}$$
$$\land\ (\mathbf{XXXXX}(i \land nr)) \tag{B.1c}$$
$$\land\ (\mathbf{F}g) \tag{B.1d}$$
$$\land\ ((\neg i)\mathbf{W}p) \tag{B.1e}$$
$$\land\ ((\neg p)\mathbf{W}(o \land (\neg p) \land (\mathbf{X}\neg p) \land (\mathbf{XX}\neg p))) \tag{B.1f}$$
$$\land\ ((\neg g)\mathbf{W}p) \tag{B.1g}$$
$$\land\ (\mathbf{G}(o \rightarrow \mathbf{XXG}\neg i)) \tag{B.1h}$$
$$\land\ (gold \rightarrow (\mathbf{G}(p \rightarrow (g \lor (\mathbf{X}g) \lor (\mathbf{XX}g) \lor (\mathbf{XXX}g))))) \tag{B.1i}$$

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

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

(B.2) 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 (B.2) with sets of time points removed. While this variant is a UC of (B.1) without sets of time points, it is not minimal: the last conjunct $\mathbf{XX}\neg p$ in (B.2c) could be replaced with TRUE without making the result satisfiable. In (B.2) with sets of time points as shown above that subformula is required for unsatisfiability. However, $\underset{[0,2],[0,5]}{\land} ((\underset{[0,5]}{\neg} p) \underset{[0,5],\{3,4\}}{\land}$ in (B.2c) could be replaced with $\underset{[0,2],[3,5]}{\land} ((\underset{[3,5]}{\neg} p) \underset{[3,5],\{3,4\}}{\land}$ without sacrificing unsatisfiability. In the latter version the sets of time points in $\underset{[0,2],[3,5]}{\land}$ highlight the fact that $o \land (\neg p) \land (\mathbf{X}\neg p) \land (\mathbf{XX}\neg p)$ in (B.1f) cannot become TRUE from time point 0 to 2 because of o and from time point 3 to 5 because of $(\neg p) \land (\mathbf{X}\neg p) \land (\mathbf{XX}\neg p)$.

Appendix C. Additional Plots

Figure C.9 shows the overhead that is incurred by extracting UCs with sets of time points by category. Figure C.10 compares Gawrychowski's and Sawa's algorithm for computing sets of time points by category.

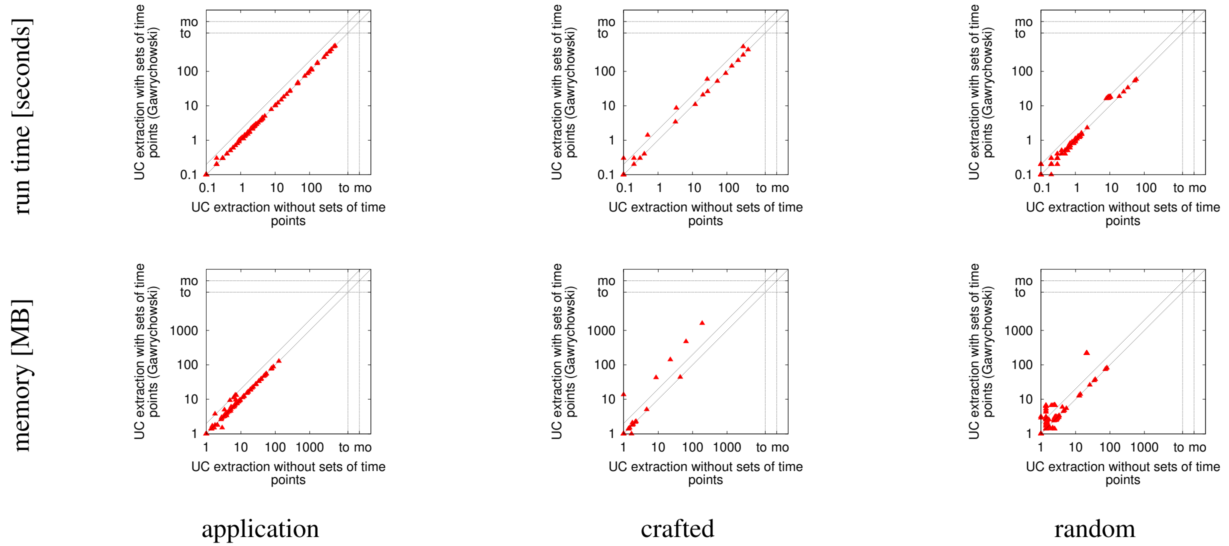

Figure C.9: Overhead incurred by UC extraction in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**. In each graph extraction of UCs with time points using Gawrychowski's algorithm is on the y-axis and UC extraction without sets of time points is on the x-axis. The off-center diagonal shows where $y = 2x$.

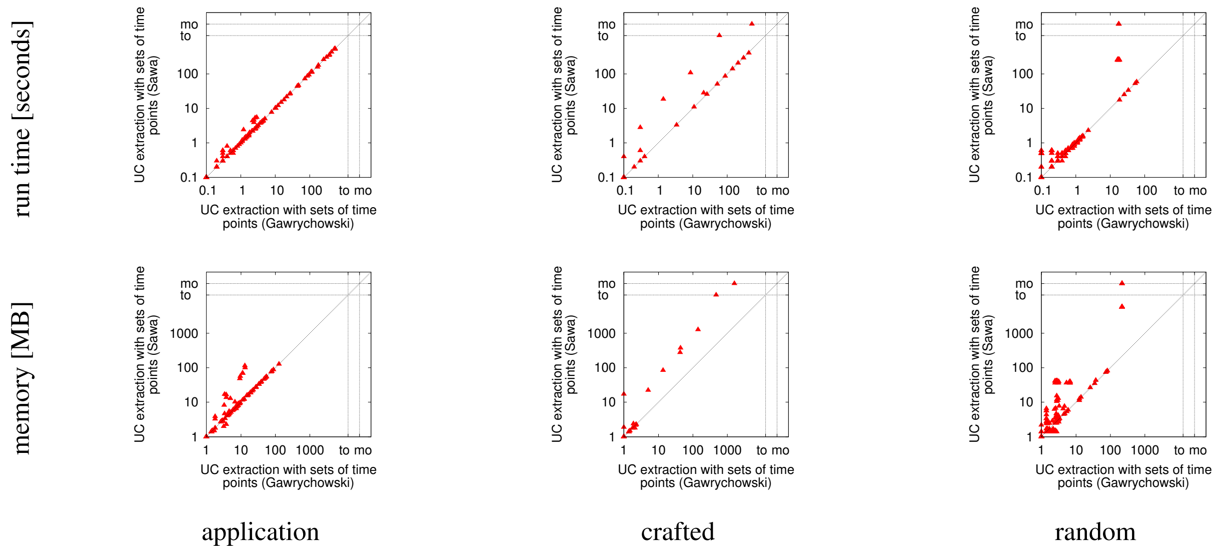

Figure C.10: Comparison of using Sawa's algorithm (y-axis) versus Gawrychowski's algorithm (x-axis) for extracting UCs with sets of time points in terms of run time (in seconds) and memory (in MB) separated by categories **application**, **crafted**, and **random**.