Journal of Artificial Intelligence Research 80 (2024) 517-558 Submitted 08/2023; published 06/2024

Computing Unsatisfiable Cores for $\mathrm{LTL_f}$ Specifications

Marco Roveri MARCO.ROVERI@UNITN.IT
University of Trento,
Via Sommarive, 9, 38123 Trento, Italy

Claudio Di Ciccio C.DICICCIO@UU.NL
Utrecht University,
Princetonplein 5, 3584 CC Utrecht, Netherlands

Chiara Di Francescomarino C.DIFRANCESCOMARINO@UNITN.IT
University of Trento,
Via Sommarive, 9, 38123 Trento, Italy

Chiara Ghidini CHIARA.GHIDINI@UNIBZ.IT
Free University of Bozen-Bolzano,
Piazza Domenicani 3, 39100 Bolzano, Italy, and
Fondazione Bruno Kessler,
Via Sommarive, 18, 38123 Trento, Italy

Abstract

Linear-time temporal logic on finite traces ($\mathrm{LTL_f}$) is rapidly becoming a de-facto standard to produce specifications in many application domains (including planning, business process management, run-time monitoring, and reactive synthesis). Several studies have challenged the satisfiability problem thus far. In this paper, we focus instead on unsatisfiable $\mathrm{LTL_f}$ specifications, with the objective of extracting the subset of formulae that cause inconsistencies within them, i.e., the unsatisfiable cores. We provide four algorithms to this end, which leverage the adaptation of a range of state-of-the-art algorithms to $\mathrm{LTL_f}$ satisfiability checking. We implement those algorithms extending the respective implementations and carry out an experimental evaluation on a set of reference benchmarks, restricting to the unsatisfiable specifications. The results put in evidence that the different algorithms and tools exhibit complementary features determining their efficiency and efficacy. Indeed, our findings suggest exploring different strategies and algorithmic solutions for the extraction of unsatisfiable cores from $\mathrm{LTL_f}$ specifications, thus confirming the challenging and multi-faceted nature of this problem.

1. Introduction

A growing body of literature evidences the adoption of linear-time temporal logic on finite traces ($\mathrm{LTL_f}$) (De Giacomo & Vardi, 2013) to produce systems specifications (De Giacomo, De Masellis, & Montali, 2014). Its widespread use spans across several application domains, including business process management (BPM) for declarative process modelling (De Giacomo, De Masellis, Grasso, et al., 2014; Montali et al., 2010) and mining (Cecconi et al., 2018; Di Ciccio & Montali, 2022; Räim et al., 2014), run-time monitoring and verification (Bauer et al., 2010; De Giacomo, De Masellis, Grasso, et al., 2014; De Giacomo et al., 2020), and AI planning (Calvanese et al., 2002; Camacho et al., 2018; Camacho & McIlraith, 2019; Sohrabi et al., 2011).

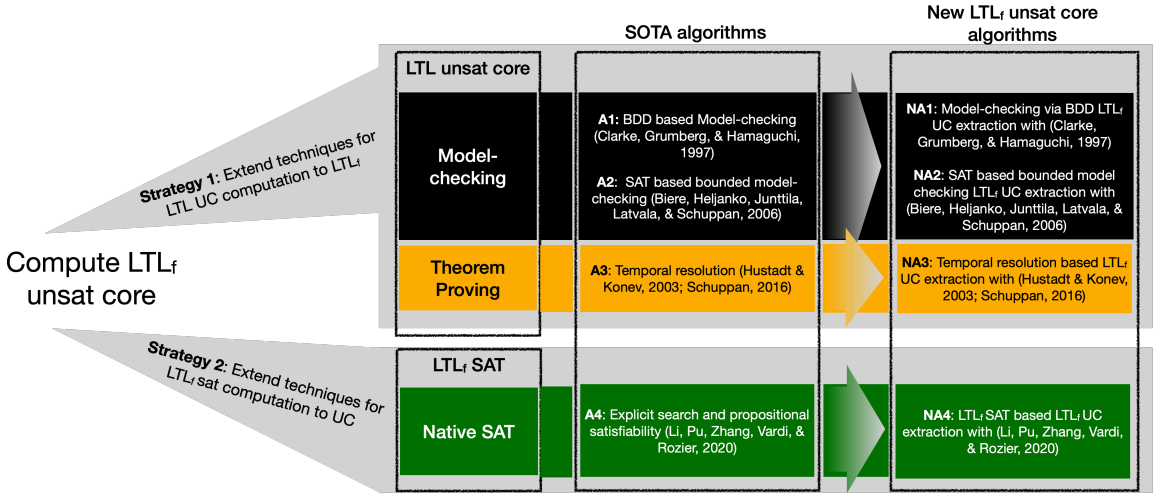


Figure 1: The algorithms.

When it comes to verification techniques and tool support for LTL_f, several studies approach the LTL_f satisfiability problem via reduction to LTL (Pnueli, 1977) satisfiability on infinite traces (De Giacomo, De Masellis, & Montali, 2014), or via specific propositional satisfiability approaches (Fionda & Greco, 2018; Li et al., 2020). However, no efforts have been devoted thus far to the identification of the formulas that lead to unsatisfiability in LTL_f specifications, with the consequence that no support has been offered for modellers and system designers to single out the causes of possible inconsistencies.

In this paper, we tackle the challenge of extracting unsatisfiable cores (UCs) from LTL_f specifications. Investigating this problem is interesting both from practical and theoretical viewpoints.

On the practical side, if unsatisfiability signals that a specification is defective, the identification of unsatisfiable cores provides the users with the opportunity to isolate the source of inconsistency and debug the relevant code fragment. Notice that determining a reason for unsatisfiability without automated support may be unfeasible for a number of reasons that range from the sheer size of the formula to the lack of time and skills of the user (Schuppan, 2012, 2018).

On the theoretical side, dealing with the extraction of UCs in LTL_f specifications is far from trivial. Indeed, there is no default *strategy* to move from the support provided for LTL to the one that has to be provided for LTL_f. We can identify two clear alternative strategies to address this problem: the first one extends algorithms for the extraction of UCs in LTL to the case of LTL_f (hereafter Strategy 1 or S1); the second one exploits algorithms that directly compute satisfiability in LTL_f to provide support for the extraction of UCs (hereafter Strategy 2 or S2). These two different strategies are emphasised in the two grey streams in Fig. 1. When looking at the *algorithms* realising the two strategies, the starting point for S1 would be state-of-the-art (SOTA) algorithms for the extraction of UCs in LTL (to be extended for LTL_f), while the starting point for S2 would be state-of-the-art algorithms for the computation of satisfiability in LTL_f (to be extended to the computation of UCs). If we look at these two classes of state-of-the-art algorithms, we

can notice a substantial imbalance. Several state-of-the-art algorithms exist for S1, in particular based on the reduction of SAT to model-checking and on theorem proving. On the contrary, the number of algorithms that could enable the implementation of S2 is still rather limited and reduces to a reference work based on an explicit search complemented with several propositional satisfiability checks. Since recent works show that often a single universal best algorithm does not exist, and systems exhibit behaviours that complement each other (Li et al., 2020, 2019), choosing a single strategy and a single algorithm from which to start is less than obvious.

In this work, we provide algorithms for the computation of UCs for $\mathrm{LTL_f}$ by exploiting both strategies and, whenever possible, different reference algorithms within each strategy. For Strategy 1, we consider three LTL satisfiability checking algorithms as the starting points: A1, based on Binary Decision Diagrams (BDDs) as described in the work of Clarke et al. (1997); A2, based on propositional satisfiability and introduced by Biere et al. (2006); A3, a theorem proving algorithm based on temporal resolution first presented by Hustadt and Konev (2003) and Schuppan (2016). For Strategy 2, we resort to the reference work of Li et al. (2020), based on explicit search and propositional satisfiability (hereinafter, A4). Figure 1 lists these algorithms inside the "SOTA algorithm" box. We believe that leveraging reference state-of-the-art approaches provides a rich starting point for the investigation of the problem and the provision of effective tools for the extraction of UCs in $\mathrm{LTL_f}$ specifications.

Our contributions thus consist of the following:

1. Four algorithms NA1, ..., NA4 that allow for the computation of an unsatisfiable core through the adaptation of algorithms A1, ..., A4, covering both Strategy 1 and Strategy 2 (Section 4). These algorithms are listed in the "New $\mathrm{LTL_f}$ unsat core algorithms" box in Fig. 1. Note that the algorithms based on propositional satisfiability (that is, NA2 and NA4) aim at extracting a UC, which may not necessarily be the minimum one. Instead NA1 and NA3 already allow for the extraction of a minimum unsatisfiable core.

2. An implementation of the proposed four algorithms NA1, ..., NA4 (Section 5.1). Three implementations extend existing tools for the corresponding original algorithms; instead, the implementation of NA3, based on temporal resolution, resorts to a preprocessing of the formula to reduce the input to the language restrictions of the original tool.

3. An experimental evaluation on a large set of reference benchmarks taken from (Li et al., 2020), restricted to the unsatisfiable ones (Sections 5.2 and 5.3). The results show an overall better time efficiency of algorithm NA4, based on Strategy 2. However, the cardinality of the UC extracted by the fastest approach is the smallest one in only about half of the cases. The experimental findings show that the proposed approaches are complementary on different specifications: depending on the varying number of propositional variables, number of conjuncts and degree of nesting of the temporal operators in the benchmarks, it is not rare that some of the implemented techniques achieve a noticeable performance when the other ones terminate with no result and vice-versa. The complementary behaviour of the different algorithms provides a further evidence of the challenge of providing an algorithmic support for the extraction

of UCs from LTL_f specifications and the adequacy of exploring different strategies and algorithmic solutions for this problem.

Since popular usages of LTL_f leverage past temporal operators (see e.g., the Declare language (van der Aalst et al., 2009)), we also provide a way to handle LTL_f with past temporal operators (see Definition 2 and all the respective technical parts). This results in the same expressive power as the one of the pure future version, though allowing for exponentially more succinct specifications (Gabbay, 1987; Laroussinie et al., 2002) and more natural encodings of LTL_f based modelling languages that make use of these operators. To this aim, we leverage algorithms already supporting LTL with past temporal operators, or a reduction to LTL_f with only future temporal operators to use existing approaches for LTL_f satisfiability checking.

The remainder of the paper is structured as follows. Section 2 and Section 3 illustrate background concepts of relevance to our work and some enablers for the extension of SOTA approaches towards the extraction of LTL_f unsatisfiable cores, respectively. Section 4 introduces the four algorithms NA1, ..., NA4, while Section 5 reports about the algorithms' implementation and experimental evaluation. Finally, related works, as well as conclusions and future works are described in Section 6 and Section 7, respectively.

2. Background

We outline here the main concepts upon which the remainder of the paper is built upon.

2.1 LTL_f Syntax and Semantics

Given a finite set of propositional variables $\mathcal{AP}$, we provide the following definitions. A *state* s over propositional variables in $\mathcal{AP}$ is a complete assignment of a Boolean value $\top$ or $\bot$ to the variables in $\mathcal{AP}$. For a set $P \subseteq \mathcal{AP}$, we denote with $s|_P$ the projection (restriction) of the complete assignments in s to consider only the propositional variables in P.

Definition 1. *We say that variable $x \in \mathcal{AP}$ holds in a state s iff x is assigned the truth value $\top$ in s, $x = \top$, and we denote this as $s \models_\text{p} x$ (where the 'p' subscript indicates that s is a model of x in a propositional sense).*

A *finite trace* over propositional variables in $\mathcal{AP}$ is a sequence $\pi = s_0, s_1, \ldots, s_{n-1}$ of states. The length of a trace $\pi = s_0, s_1, \ldots, s_{n-1}$, denoted $\mathsf{len}(\pi)$, is n. We denote with $\pi[i]$, with $0 \leq i < \mathsf{len}(\pi)$, the i-th state s_i. Given a finite trace π as above, we denote with $\pi[i, j]$ the segment of trace π ranging from i to j with $0 \leq i \leq j < \mathsf{len}(\pi)$, i.e., $\pi[i, j] = s_i, \ldots, s_j$. Given two (not necessarily distinct) traces $\pi' = s'_0, s'_1, \ldots, s'_{n'-1}$ and $\pi'' = s''_0, s''_1, \ldots, s''_{n''-1}$, we denote with the juxtaposition $\pi'\pi''$ the concatenation of the sequences, i.e., $\pi'\pi'' = s'_0, s'_1, \ldots, s'_{n'-1}, s''_0, s''_1, \ldots, s''_{n''-1}$.

An *infinite trace* over propositional variables in $\mathcal{AP}$ is a sequence $\pi = s_0, s_1, ...$ of states such that $\pi \in (2^{\mathcal{AP}})^\omega$. Given two finite traces π_1 and π_2, we indicate with $\pi_1\pi_2^\omega$ the infinite *lasso-shaped* trace with prefix π_1 and trace π_2 repeated indefinitely (intuitively, to indicate that π_2 is repeated within an infinite loop).

An *LTL_f formula* φ is built over the propositional variables in $\mathcal{AP}$ by using the classical Boolean connectives "$\wedge$", "$\vee$", and "$\neg$", complemented with the future temporal operators

"**X**" (next), "**N**" (weak next), "**G**" (always/globally), "**F**" (eventually/finally), "**U**" (until) and "**R**" (release), and with the past temporal operators "**Y**" (yesterday), "**Z**" (weak yesterday), "**H**" (historically), "**O**" (once), "**S**" (since), and "**T**" (trigger). The **N** operator is similar to **X** and solely differs in the way the final state is dealt with: in the last state, $\mathbf{X}\,\varphi$ is false, while $\mathbf{N}\,\varphi$ is true. Similarly, the **Z** operator is analogous to **Y** as in the sole initial state a difference occurs: in s_0, $\mathbf{Y}\,\varphi$ is false, whereas $\mathbf{Z}\,\varphi$ is true. (See Definition 2 for the semantics of all the $\mathrm{LTL_f}$ operators.)

The grammar for building $\mathrm{LTL_f}$ formulas is:

$$\begin{aligned}\varphi ::= {} & x \mid (\varphi_1 \wedge \varphi_2) \mid (\varphi_1 \vee \varphi_2) \mid \neg\varphi_1 \mid \\ & \textit{Future temporal operators} \\ & (\mathbf{X}\,\varphi_1) \mid (\mathbf{N}\,\varphi_1) \mid (\mathbf{F}\,\varphi_1) \mid (\mathbf{G}\,\varphi_1) \mid (\varphi_1\,\mathbf{U}\,\varphi_2) \mid (\varphi_1\,\mathbf{R}\,\varphi_2), \\ & \textit{Past temporal operators} \\ & (\mathbf{Y}\,\varphi_1) \mid (\mathbf{Z}\,\varphi_1) \mid (\mathbf{H}\,\varphi_1) \mid (\mathbf{O}\,\varphi_1) \mid (\varphi_1\,\mathbf{S}\,\varphi_2) \mid (\varphi_1\,\mathbf{T}\,\varphi_2),\end{aligned}$$

where $x \in \mathcal{AP}$ is a propositional variable, φ_1 and φ_2 are $\mathrm{LTL_f}$ formulas. Classical implication $\rightarrow$ and equivalence $\leftrightarrow$ connectives can be obtained in standard ways in terms of the $\wedge, \vee, \neg$ connectives. In the following, we use round parentheses as auxiliary symbols to clarify or alter the precedence of evaluation. Otherwise, we might omit them for the sake of readability.

Definition 2 ($\mathrm{LTL_f}$ Satisfiability)**.** *Given a* finite *trace π, the LTL$_f$ formula φ is true in π at state $\pi[i]$ s.t. $0 \leq i \leq \mathsf{len}(\pi) - 1$, denoted with $\pi, i \models \varphi$, iff:*

- $\pi, i \models x$ *iff* $\pi[i] \models_{\mathrm{p}} x$ *for* $x \in \mathcal{AP}$*;*
- $\pi, i \models \varphi_1 \wedge \varphi_2$ *iff* $\pi, i \models \varphi_1$ *and* $\pi, i \models \varphi_2$*;*
- $\pi, i \models \varphi_1 \vee \varphi_2$ *iff* $\pi, i \models \varphi_1$ *or* $\pi, i \models \varphi_2$*;*

Future temporal operators:

- $\pi, i \models \mathbf{X}\,\varphi$ *iff* $i < \mathsf{len}(\pi) - 1$ *and* $\pi, i + 1 \models \varphi$*;*
- $\pi, i \models \mathbf{N}\,\varphi$ *iff* $i < \mathsf{len}(\pi) - 1$ *and* $\pi, i + 1 \models \varphi$*, or* $i = \mathsf{len}(\pi) - 1$*;*
- $\pi, i \models \mathbf{F}\,\varphi$ *iff for some* j *with* $i \leq j < \mathsf{len}(\pi)$ *it holds that* $\pi, j \models \varphi$*;*
- $\pi, i \models \mathbf{G}\,\varphi$ *iff for every* j *with* $i \leq j < \mathsf{len}(\pi)$ *it holds that* $\pi, j \models \varphi$*;*
- $\pi, i \models \varphi_1\,\mathbf{U}\,\varphi_2$ *iff for some* j *with* $i \leq j < \mathsf{len}(\pi)$ *it holds that* $\pi, j \models \varphi_2$ *and for every* k *with* $i \leq k < j$ *it holds that* $\pi, k \models \varphi_1$*;*
- $\pi, i \models \varphi_1\,\mathbf{R}\,\varphi_2$ *iff for every* j *with* $i \leq j < \mathsf{len}(\pi)$ *it holds that* $\pi, j \models \varphi_2$*, or for some* j *with* $i \leq j < \mathsf{len}(\pi)$ *it holds that* $\pi, j \models \varphi_1$ *and for every* k *with* $i \leq k \leq j$ *it holds that* $\pi, k \models \varphi_2$*;*

Past temporal operators:

- $\pi, i \models \mathbf{Y}\,\varphi$ *iff* $1 \leq i$ *and* $\pi, i - 1 \models \varphi$*;*

- $\pi, i \models \mathbf{Z}\,\varphi$ *iff* $0 = i$ *or* $\pi, i-1 \models \varphi$*;*
- $\pi, i \models \mathbf{O}\,\varphi$ *iff for some* j *with* $0 \leq j \leq i$ *it holds that* $\pi, j \models \varphi$*;*
- $\pi, i \models \mathbf{H}\,\varphi$ *iff for every* j *with* $0 \leq j \leq i$ *it holds that* $\pi, j \models \varphi$*;*
- $\pi, i \models \varphi_1\,\mathbf{S}\,\varphi_2$ *iff for some* k *with* $0 \leq k \leq i$ *it holds that* $\pi, k \models \varphi_2$ *and for every* j *with* $k < j \leq i$ *it holds that* $\pi, j \models \varphi_1$*;*
- $\pi, i \models \varphi_1\,\mathbf{T}\,\varphi_2$ *iff for every* k *with* $0 \leq k \leq i$ *it holds that* $\pi, k \models \varphi_2$ *or for some* j *with* $k < j \leq i$ *it holds that* $\pi, j \models \varphi_1$*.*

We say that the finite trace π *is a* model of φ *(denoted with* $\pi \models \varphi$*) whenever* $\pi, 0 \models \varphi$*, and that* φ *is* satisfiable *whenever there exists a* π *such that* $\pi, 0 \models \varphi$*.*

Remark 2.1. *The following equivalences hold:*

- $(\mathbf{Z}\,\varphi) \leftrightarrow \neg(\mathbf{Y}\,\neg\varphi)$*,*
- $(\mathbf{O}\,\varphi) \leftrightarrow (\top\,\mathbf{S}\,\varphi)$*,*
- $(\mathbf{H}\,\varphi) \leftrightarrow \neg(\top\,\mathbf{S}\,\neg\varphi)$*,*
- $(\varphi_1\,\mathbf{T}\,\varphi_2) \leftrightarrow \neg(\neg\varphi_1\,\mathbf{S}\,\neg\varphi_2)$*, and*
- $(\varphi_1\,\mathbf{R}\,\varphi_2) \leftrightarrow \neg(\neg\varphi_1\,\mathbf{U}\,\neg\varphi_2)$*.*

In the remainder of this paper, we leverage the above equivalences whenever needed to simplify the presentation and the proofs.

The language of an $\mathrm{LTL_f}$ formula φ over $\mathcal{AP}$ is defined as $\mathcal{L}(\varphi) = \{\pi \mid \pi, 0 \models \varphi\}$. Thus, the satisfiability problem for an $\mathrm{LTL_f}$ formula φ can be reduced to checking that $\mathcal{L}(\varphi) \neq \emptyset$.

Let us consider, e.g., the formula $\varphi = \mathbf{G}(a \rightarrow \mathbf{N}\,b)$. Let $\pi^1 = s_0^1, s_1^1, s_2^1, s_3^1$ be a trace of length 4 such that $s_0^1 = \{a = \bot, b = \top\}$, $s_1^1 = \{a = \top, b = \bot\}$, $s_2^1 = \{a = \top, b = \top\}$, $s_3^1 = \{a = \top, b = \top\}$. Trace π^1 satisfies φ. On the contrary, trace $\pi^2 = s_0^2, s_1^2, s_2^2, s_3^2$, where $s_0^2 = \{a = \bot, b = \top\}$, $s_1^2 = \{a = \top, b = \bot\}$, $s_2^2 = \{a = \top, b = \top\}$, $s_3^2 = \{a = \top, b = \bot\}$, is not a model for φ since $s_2^2 \models_{\mathrm{p}} a$, but the next state s_3^2 is such that $s_3^2 \not\models_{\mathrm{p}} b$. The $\mathrm{LTL_f}$ formula $\phi' = \mathbf{G}(a \rightarrow \mathbf{X}\,b)$ is not satisfied by either of the traces. Indeed, the last state of π_1 is such that $s_3^1 \models_{\mathrm{p}} a$, but it is not followed by any state. As for π_2, $s_2^2 \models_{\mathrm{p}} a$ but in the next state $s_3^2 \not\models_{\mathrm{p}} b$; also, $s_3^1 \models_{\mathrm{p}} a$ but that is the last state, so no next state exists.

Formulas that contain both past and future temporal operators in the same formula are widely used, as they allow for the expression of requirements or behavioural rules in a more concise and natural way (Cecconi et al., 2018; Fuxman et al., 2004; van Lamsweerde & Letier, 2000). For instance, as also discussed in (Cimatti et al., 2004), a requirement like *if a problem is diagnosed, then a failure must have previously occurred* can be naturally formalised as $\mathbf{G}(\text{problem} \rightarrow \mathbf{O}\,\text{failure})$. This formalisation can be interpreted more intuitively than the pure future counterpart $\neg(\neg\text{failure}\,\mathbf{U}\,\text{problem})$. Similarly, the requirement *grants are issued only upon requests* can be easily specified as $\mathbf{G}(\text{grant} \rightarrow \mathbf{Y}(\neg\text{grant}\,\mathbf{S}\,\text{request}))$, which is more compact than the pure-future formulation: $(\text{request}\,\mathbf{R}\,\neg\text{grant}) \wedge \mathbf{G}(\text{grant} \rightarrow (\text{request} \vee (\mathbf{X}(\text{request}\,\mathbf{R}\,\neg\text{grant}))))$.

Commonalities and Differences between LTL and $\mathrm{LTL_F}$

The syntax of $\mathrm{LTL_f}$ formulas is almost identical to the original LTL one. Semantics differ, instead, due to the finite length of traces in $\mathrm{LTL_f}$, as a last state occurs only in a finite trace. Only in $\mathrm{LTL_f}$, then, **N** and **X** are satisfied under different conditions. Thus, while introducing the semantics for LTL we report only the semantics for the **X** operator. In the following, we will consider the semantics for LTL and highlight the differences with $\mathrm{LTL_f}$ whenever necessary.

Definition 3 (LTL Satisfiability)**.** *Given an infinite trace π, the LTL formula φ is true in π at state $\pi[i]$ s.t. $i \geq 0$, denoted with $\pi, i \models_{\mathrm{LTL}} \varphi$, iff:*

- $\pi, i \models_{\mathrm{LTL}} x$ *iff* $\pi[i] \models_{\mathrm{p}} x$ *for* $x \in \mathcal{AP}$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1 \wedge \varphi_2$ *iff* $\pi, i \models_{\mathrm{LTL}} \varphi_1$ *and* $\pi, i \models_{\mathrm{LTL}} \varphi_2$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1 \vee \varphi_2$ *iff* $\pi, i \models_{\mathrm{LTL}} \varphi_1$ *or* $\pi, i \models_{\mathrm{LTL}} \varphi_2$*;*

Future temporal operators:

- $\pi, i \models_{\mathrm{LTL}} \mathbf{X}\,\varphi$ *iff* $\pi, i+1 \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \mathbf{F}\,\varphi$ *iff for some* j *with* $i \leq j$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \mathbf{G}\,\varphi$ *iff for every* j *with* $i \leq j$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1\,\mathbf{U}\,\varphi_2$ *iff for some* j *with* $i \leq j$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi_2$ *and for every* k *with* $i \leq k < j$ *it holds that* $\pi, k \models_{\mathrm{LTL}} \varphi_1$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1\,\mathbf{R}\,\varphi_2$ *iff for every* j *with* $i \leq j$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi_2$*, or for some* j *with* $i \leq j$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi_1$ *and for every* k *with* $i \leq k \leq j$ *it holds that* $\pi, k \models_{\mathrm{LTL}} \varphi_2$*;*

Past temporal operators:

- $\pi, i \models_{\mathrm{LTL}} \mathbf{Y}\,\varphi$ *iff* $1 \leq i$ *and* $\pi, i-1 \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \mathbf{Z}\,\varphi$ *iff* $0 = i$ *or* $\pi, i-1 \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \mathbf{O}\,\varphi$ *iff for some* j *with* $0 \leq j \leq i$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \mathbf{H}\,\varphi$ *iff for every* j *with* $0 \leq j \leq i$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1\,\mathbf{S}\,\varphi_2$ *iff for some* k *with* $0 \leq k \leq i$ *it holds that* $\pi, k \models_{\mathrm{LTL}} \varphi_2$ *and for every* j *with* $k < j \leq i$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi_1$*;*
- $\pi, i \models_{\mathrm{LTL}} \varphi_1\,\mathbf{T}\,\varphi_2$ *iff for every* k *with* $0 \leq k \leq i$ *it holds that* $\pi, k \models_{\mathrm{LTL}} \varphi_2$ *or for some* j *with* $k < j \leq i$ *it holds that* $\pi, j \models_{\mathrm{LTL}} \varphi_1$*;*

We say that the infinite trace π is a model of φ *(denoted with $\pi \models_{\mathrm{LTL}} \varphi$) whenever $\pi, 0 \models_{\mathrm{LTL}} \varphi$, and that the LTL property φ* is satisfiable *whenever there exists a π such that $\pi, 0 \models_{\mathrm{LTL}} \varphi$.*

When clear from the context, for an LTL property φ, we abuse notation and use $\pi, i \models \varphi$ in place of $\pi, i \models_{\text{LTL}} \varphi$.

As noticed in (De Giacomo, De Masellis, & Montali, 2014), the evaluation of an LTL formula on an infinite trace may lead to an opposite outcome to the evaluation of an identical expression in LTL_f on finite traces. For example, $\mathbf{F}\, a \wedge \mathbf{G}(a \rightarrow \mathbf{F}\, b) \wedge \mathbf{G}(b \rightarrow \mathbf{F}\, a) \wedge \mathbf{G}\, \neg(a \wedge b)$ is satisfiable in LTL and unsatisfiable in LTL_f. A satisfying infinite trace π in LTL is $\pi = s_0, (s_1, s_2)^\omega$ such that $s_0 = \{a = \top, b = \bot\}$, $s_1 = \{a = \bot, b = \top\}$, $s_2 = \{a = \top, b = \bot\}$. However, in LTL_f $\mathbf{F}\, a \wedge \mathbf{G}(a \rightarrow \mathbf{F}\, b) \wedge \mathbf{G}(b \rightarrow \mathbf{F}\, a)$ implies that eventually, both a and b shall be true at the same time, and this is in contradiction with $\mathbf{G}\, \neg(a \wedge b)$ which requires that both a and b are never true simultaneously.

2.1.1 Unsatisfiable Cores

A set of LTL_f formulas $\Gamma = \{\varphi_1, ..., \varphi_N\}$ is a *specification*. Formulas $\varphi_1, ..., \varphi_N$ are considered in implicit conjunction. Therefore, we adopt with a slight abuse of notation both the set-based and the conjunctive expressions for specifications, i.e., $\Gamma = \bigwedge_{i=1..N} \varphi_i$.

Definition 4 (Unsatisfiable core)**.** *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be an LTL_f unsatisfiable specification. $\Phi \subseteq \Gamma$ is an* unsatisfiable core *of Γ iff Φ is unsatisfiable. A* minimal unsatisfiable core *Φ is such that $\Phi_i = \Phi \setminus \{\varphi_i\}$ for every $\varphi_i \in \Phi$ is satisfiable. A* minimum unsatisfiable core *is a minimal unsatisfiable core with the smallest possible cardinality.*

Remark 4.1. *Definition 4 can be seamlessly applied to LTL by considering Def. 3 in place of Def. 2 for satisfiability.*

Consider, e.g., the specification $\Gamma = \{\varphi_1, \ldots, \varphi_6\}$ of LTL_f formulas where $\varphi_1 = \mathbf{F}(a \vee b \vee c)$, $\varphi_2 = \mathbf{G}(a \rightarrow \mathbf{X}\, \mathbf{F}\, b)$, $\varphi_3 = \mathbf{G}(b \rightarrow \mathbf{X}\, \mathbf{F}\, c)$, $\varphi_4 = \mathbf{G}(c \rightarrow \mathbf{X}\, \mathbf{F}\, a)$, $\varphi_5 = \mathbf{G}(b \rightarrow \mathbf{X}\, \mathbf{F}\, a)$, $\varphi_6 = \mathbf{G}(c \rightarrow \mathbf{Y}\, b)$. Intuitively, the specification Γ is unsatisfiable because of circular dependencies that require a to be eventually followed by b, b by c and a, and c by a. Since φ_1 requires that at least one among a, b or c is eventually satisfied in the trace, only an infinite trace could satisfy Γ as a whole. Γ is a trivial unsatisfiable core, then. The specification $\{\varphi_1, \varphi_2, \varphi_3, \varphi_4\} \subseteq \Gamma$ is a minimal unsatisfiable core (since the removal of any of $\varphi_1, \varphi_2, \varphi_3, \varphi_4$ breaks the circular dependency). The specification $\{\varphi_1, \varphi_2, \varphi_5\} \subseteq \Gamma$ is not only minimal but also a minimum unsatisfiable core as it bears the lowest cardinality.

2.2 Checking Satisfiability of an LTL_f Formula

Checking the satisfiability of an LTL_f formula φ can be reduced to checking language emptiness of a nondeterministic finite state automaton (De Giacomo, De Masellis, & Montali, 2014). Alternative approaches for LTL_f formulas without past temporal operators (De Giacomo, De Masellis, & Montali, 2014; De Giacomo & Vardi, 2013; Fionda & Greco, 2018) address this problem by checking the satisfiability of an equi-satisfiable LTL formula over infinite traces (see Def. 3) leveraging on existing well-established techniques (see, e.g., Biere et al. 2006; Clarke et al. 1997). These approaches proceed as follows: (i) they introduce a new fresh propositional variable $\mathsf{end} \notin \mathcal{AP}$ used to denote the trace has ended; (ii) they require that end eventually holds (i.e., $\mathbf{F}\, \mathsf{end}$); (iii) they require that once end becomes true, it stays true forever (i.e., $\mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\, \mathsf{end})$); (iv) they translate the LTL_f formula φ into

an LTL formula by means of a rewriting function $\mathsf{f2l}(\varphi)$ that is defined recursively on the structure of the $\mathrm{LTL_f}$ formula φ as follows:

$$\begin{aligned}
\mathsf{f2l}(x) &\mapsto x \text{ for } x \in \mathcal{AP} \\
\mathsf{f2l}(\neg\varphi) &\mapsto \neg\mathsf{f2l}(\varphi) \\
\mathsf{f2l}(\varphi_1 \wedge \varphi_2) &\mapsto \mathsf{f2l}(\varphi_1) \wedge \mathsf{f2l}(\varphi_2) \\
\mathsf{f2l}(\varphi_1 \vee \varphi_2) &\mapsto \mathsf{f2l}(\varphi_1) \vee \mathsf{f2l}(\varphi_2) \\
\mathsf{f2l}(\mathbf{X}\,\varphi) &\mapsto \mathbf{X}(\mathsf{f2l}(\varphi) \wedge \neg\mathsf{end}) \\
\mathsf{f2l}(\mathbf{N}\,\varphi) &\mapsto \mathbf{X}(\mathsf{f2l}(\varphi) \vee \mathsf{end}) \\
\mathsf{f2l}(\mathbf{F}\,\varphi) &\mapsto \mathbf{F}(\mathsf{f2l}(\varphi) \wedge \neg\mathsf{end}) \\
\mathsf{f2l}(\mathbf{G}\,\varphi) &\mapsto \mathbf{G}(\mathsf{f2l}(\varphi) \vee \mathsf{end}) \\
\mathsf{f2l}(\varphi_1\,\mathbf{U}\,\varphi_2) &\mapsto \mathsf{f2l}(\varphi_1)\,\mathbf{U}(\mathsf{f2l}(\varphi_2) \wedge \neg\mathsf{end}) \\
\mathsf{f2l}(\varphi_1\,\mathbf{R}\,\varphi_2) &\mapsto (\mathsf{f2l}(\varphi_1) \wedge \neg\mathsf{end})\,\mathbf{R}(\mathsf{f2l}(\varphi_2) \vee \mathsf{end})
\end{aligned}$$

Theorem 1 (De Giacomo, De Masellis, and Montali 2014)**.** *Any $\mathrm{LTL_f}$ formula without past temporal operators φ is satisfiable iff the LTL formula*

$$\mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\varphi) \tag{1}$$

is satisfiable.

Hereafter, we denote with $\mathsf{LTLf2LTL}(\varphi)$ the LTL formula obtained by applying equation (1) in Theorem 1 to the $\mathrm{LTL_f}$ formula φ, i.e., $\mathsf{LTLf2LTL}(\varphi) := \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\varphi)$.The resulting LTL formula can then be checked for satisfiability with any state-of-the-art LTL satisfiability checker as discussed by De Giacomo, De Masellis, and Montali 2014; Li et al. 2020.

Finally, in SAT-based frameworks for $\mathrm{LTL_f}$ satisfiability checking like the one proposed by Li et al. 2020, propositional SAT solving techniques are used to construct a transition system T_φ for a given $\mathrm{LTL_f}$ formula φ, and $\mathrm{LTL_f}$ satisfiability checking reduces to a path search problem over the constructed transition system.[1]

Theorem 2 (Li et al. 2020)**.** *Let φ be an $\mathrm{LTL_f}$ formula without past temporal operators. φ is satisfiable iff there is a final state in T_φ.*

A *final state* for T_φ is any state satisfying the Boolean formula $\mathsf{end} \wedge (\mathsf{xnf}(\varphi))^p$, where

(i) end is a new propositional variable such that $\mathsf{end} \notin \mathcal{AP}$ to identify the last state of satisfying traces (similarly to De Giacomo, De Masellis, and Montali 2014);

(ii) $\mathsf{xnf}(\varphi)$ is the *neXt Normal Form* of φ, an equi-satisfiable formula to φ such that there are no Until/Release sub-formulas in the propositional atoms[2] of $\mathsf{xnf}(\varphi)$, built linearly from φ; and

1. In the rest of this subsection we use the notions and notations introduced in Li et al. (2020).
2. Following Def. 3 of Li et al. (2020), the propositional atoms of an $\mathrm{LTL_f}$ formula φ are the propositional variables, and the Next/Until/Release formulas. Intuitively, Li et al. (2020) considers all temporal sub-formulas of φ as propositional atoms.

(iii) $(\mathsf{xnf}(\varphi))^p$ is a propositional formula[3] over the propositional atoms of $\mathsf{xnf}(\varphi)$.

This approach uses a conflict driven algorithm, leveraging on propositional unsatisfiable cores, to perform the explicit path-search.

Next, we report some useful definitions, and we refer to Li et al. (2020) for the full details of this approach.

Definition 5 (Conflict Sequence, Li et al. 2020)**.** *Given an LTL_f formula φ, a* conflict sequence $\mathcal{C}$ *for the transition system T_φ is a finite sequence of sets of states such that:*

- *The initial state $s_0 = \{\varphi\}$ is in $\mathcal{C}[i]$ for $0 \leq i < |\mathcal{C}|$;*
- *Every state in $\mathcal{C}[0]$ is not a final state;*
- *For every state $s \in \mathcal{C}[i+1]$ such that $0 \leq i < |\mathcal{C}| - 1$, all the one-transition next states of s are included in $\mathcal{C}[i]$.*

We call each $\mathcal{C}[i]$ a frame*, and i is the* frame level*.*

For a given conflict sequence $\mathcal{C}$, the set $\bigcap_{0 \leq j < i} \mathcal{C}[j]$ (for $0 \leq i < |\mathcal{C}|$) represents a set of states that cannot reach a final state of T_φ in up to i steps.

Theorem 3 (Li et al. 2020)**.** *The LTL_f formula φ is unsatisfiable iff there are a conflict sequence $\mathcal{C}$ and an $i \geq 0$ such that $\bigcap_{0 \leq j < i} \mathcal{C}[j] \subseteq \mathcal{C}[i+1]$.*

We refer the reader to Li et al. (2020) for further details about the construction of T_φ, for the SAT-based algorithm to check for the existence of a final state in T_φ, and for the correctness and termination of that algorithm.

2.2.1 Symbolic Approaches to Check Language Emptiness for LTL

A standard symbolic approach to check language emptiness for a given LTL formula φ was proposed by Clarke et al. (1997) in the context of model checking with fairness constraints. It proceeds as follows: (i) first, it builds a Symbolic Non-Deterministic Büchi automaton for φ; (ii) then, it computes the set of fair states according to this automaton; finally, (iii) it intersects it with the set of initial states. The resulting set, denoted with $[\![\varphi]\!]$, is a propositional formula whose models represent all states that are the initial state of some infinite trace that accepts φ.

More precisely, let $\mathcal{M}_\varphi$ be a symbolic fair transition system over a set of Boolean variables $\mathcal{AP}_\varphi$ that encodes the formula φ, as discussed for instance in the work of Clarke et al. (1997). In this setting, $\mathcal{AP}_\varphi = \mathcal{AP} \cup \mathcal{AP}_{B(\varphi)}$ contains all the propositional variables $\mathcal{AP}$ and the Boolean variables $\mathcal{AP}_{B(\varphi)}$ (such that $\mathcal{AP}_{B(\varphi)} \cap \mathcal{AP} = \emptyset$) needed to encode a symbolic fair transition system representing the Büchi automaton for φ.[4] We denote with $[\![\varphi]\!]$ the set of states of this symbolic fair transition system such that the following assumptions hold:

3. Intuitively, for an LTL_f formula φ, the $(\cdot)^p$ is a function that traverses the $\mathsf{xnf}(\varphi)$ formula and replaces each propositional atom of $\mathsf{xnf}(\varphi)$ with a corresponding fresh propositional variable. We refer the reader to (Li et al., 2020) for further details.
4. We refer the reader to the work of Clarke et al. (1997) for (i) the formal definition of symbolic fair transition system and (ii) the details on a construction of a symbolic fair transition system $\mathcal{M}_\varphi$ for a given LTL formula φ.

(AssN1) All states in $[\![\varphi]\!]$ are the starting point of some trace accepting φ;

(AssN2) All words accepted by φ are accepted by some trace starting from $[\![\varphi]\!]$.

Notice that this approach is suitable both for BDD-based and for SAT-based approaches to LTL satisfiability.

2.2.2 Temporal Resolution Approaches for LTL Satisfiability

LTL satisfiability can also be addressed with temporal resolution (Fisher, 1991; Fisher et al., 2001). Temporal resolution extends classical propositional resolution with specific inference rules for each temporal operator. Temporal resolution has been implemented in solvers like TRP++ (Hustadt & Konev, 2003) showing effectiveness in analysing unsatisfiable LTL formulas (Schuppan & Darmawan, 2011). We refer the reader to the work of Fisher (1991); Fisher et al. (2001); Hustadt and Konev (2003) for further details. We remark that Schuppan (2016) showed how the temporal resolution proof graph constructed to prove unsatisfiability of an LTL formula without past temporal operators could be used to compute a minimal unsatisfiable core for the respective LTL formula.

3. Enablers

This section presents three enablers that allow for the extension of existing algorithms in the scientific literature towards the extraction of LTL$_f$ unsatisfiable cores. We will resort to these enablers for the design and realisation of four new algorithms, as described in the next section. In particular, here we illustrate: (i) the extension of the translation function $\mathsf{f2l}(\varphi)$ presented in Section 2 to handle LTL$_f$ past temporal operators; (ii) a translation that allows for the transformation of any LTL$_f$ formula with past temporal operators in an equi-satisfiable one with only future temporal operators; (iii) the use of an activation variable associated to each LTL$_f$ formula in Γ to extract unsatisfiable cores from existing frameworks for LTL/LTL$_f$ satisfiability. The first result enables the use of any framework for LTL satisfiability checking that supports both past and future temporal operators. The second result enables the use of any framework for LTL/LTL$_f$ satisfiability checking that supports only future temporal operators. Finally, the third result enables the computation of unsatisfiable cores of Γ leveraging existing LTL/LTL$_f$ satisfiability frameworks. Next, we describe the three enablers in detail.

3.1 Extending f2l to Handle Past Temporal Operators

We remark that the semantics for past temporal operators over finite traces coincides with the respective semantics on infinite traces, as it refers to the prefix of the trace. Therefore, given an LTL$_f$ formula φ, we can extend the $\mathsf{f2l}(\varphi)$ encoding to handle LTL$_f$ past temporal operators as follows:

$$\begin{array}{llll} \mathsf{f2l}(\mathbf{Y}\,\varphi) \mapsto \mathbf{Y}(\mathsf{f2l}(\varphi)); & \mathsf{f2l}(\mathbf{Z}\,\varphi) \mapsto \mathbf{Z}(\mathsf{f2l}(\varphi)); \\ \mathsf{f2l}(\mathbf{O}\,\varphi) \mapsto \mathbf{O}(\mathsf{f2l}(\varphi)); & \mathsf{f2l}(\mathbf{H}\,\varphi) \mapsto \mathbf{H}(\mathsf{f2l}(\varphi)); \end{array}$$

$$\begin{array}{l} \mathsf{f2l}(\varphi_1\,\mathbf{S}\,\varphi_2) \mapsto \mathsf{f2l}(\varphi_1)\,\mathbf{S}\,\mathsf{f2l}(\varphi_2); \\ \mathsf{f2l}(\varphi_1\,\mathbf{T}\,\varphi_2) \mapsto \mathsf{f2l}(\varphi_1)\,\mathbf{T}\,\mathsf{f2l}(\varphi_2). \end{array}$$

Basically, the encoding of a past operator is propagated recursively to the sub-formulas without modifications on the past operator itself. Together with Theorem 1, this extension allows us to prove the following corollary.

Corollary 1. *Any LTL$_f$ formula φ is satisfiable iff the following LTL formula is satisfiable:*

$$\mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\varphi). \tag{2}$$

This corollary enables the use of any framework for LTL satisfiability checking that supports both past and future temporal operators.

3.2 Removing Past Temporal Operators

Given an LTL$_\mathrm{f}$ formula φ with past operators, we can build an equi-satisfiable LTL$_\mathrm{f}$ formula over only future operators using the function $\mathsf{p2f}(\varphi, \emptyset) = \langle \varphi', \Upsilon \rangle$ that takes an LTL$_\mathrm{f}$ formula φ with past operators, and builds a new LTL$_\mathrm{f}$ formula φ' and a set of LTL$_\mathrm{f}$ formulas without past operators Υ as follows:

$$\begin{aligned}
\mathsf{p2f}(x, \Upsilon) \mapsto & \langle x, \Upsilon \rangle \text{ for } x \in \mathcal{AP} \\
\mathsf{p2f}(\odot\varphi, \Upsilon) \mapsto & \langle \odot\varphi', \Upsilon' \rangle \text{ where } \langle \varphi', \Upsilon' \rangle = \mathsf{p2f}(\varphi, \Upsilon) \\
& \text{and } \odot \in \{\neg, \mathbf{X}, \mathbf{N}, \mathbf{F}, \mathbf{G}\} \\
\mathsf{p2f}(\varphi_1 \oplus \varphi_2, \Upsilon) \mapsto & \langle \varphi_1' \oplus \varphi_2', \Upsilon' \rangle \text{ where } \langle \varphi_1', \Upsilon_1 \rangle = \mathsf{p2f}(\varphi_1, \Upsilon), \\
& \langle \varphi_2', \Upsilon_2 \rangle = \mathsf{p2f}(\varphi_2, \Upsilon), \Upsilon' = \Upsilon_1 \cup \Upsilon_2, \text{ and} \\
& \oplus \in \{\wedge, \vee, \mathbf{U}, \mathbf{R}\} \\
\mathsf{p2f}(\mathbf{Z}\,\varphi, \Upsilon) \mapsto & \mathsf{p2f}(\neg\,\mathbf{Y}\,\neg\varphi, \Upsilon) \\
\mathsf{p2f}(\mathbf{Y}\,\varphi, \Upsilon) \mapsto & \langle v_{\mathbf{Y}\,\varphi}, \Upsilon'' \rangle \text{ where } v_{\mathbf{Y}\,\varphi} \notin \mathcal{AP} \text{ is a fresh distinct propositional variable,} \\
& \langle \varphi', \Upsilon' \rangle = \mathsf{p2f}(\varphi, \Upsilon), \\
& \Upsilon'' = \Upsilon' \cup \{\neg v_{\mathbf{Y}\,\varphi}, \mathbf{G}((\mathbf{X}\,v_{\mathbf{Y}\,\varphi}) \leftrightarrow \varphi')\} \\
\mathsf{p2f}(\varphi_1\,\mathbf{S}\,\varphi_2, \Upsilon) \mapsto & \langle \varphi_2' \vee (\varphi_1' \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2}), \Upsilon' \rangle \text{ where} \\
& v_{\varphi_1\,\mathbf{S}\,\varphi_2} \notin \mathcal{AP} \text{ is a fresh distinct propositional variable,} \\
& \langle \varphi_1', \Upsilon_1 \rangle = \mathsf{p2f}(\varphi_1, \Upsilon), \langle \varphi_2', \Upsilon_2 \rangle = \mathsf{p2f}(\varphi_2, \Upsilon_1), \\
& \Upsilon' = \Upsilon_1 \cup \Upsilon_2 \cup \{\neg v_{\varphi_1\,\mathbf{S}\,\varphi_2}\} \cup \\
& \qquad \{\mathbf{G}((\mathbf{X}\,v_{\varphi_1\,\mathbf{S}\,\varphi_2}) \leftrightarrow (\varphi_2 \vee (\varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2})))\} \\
\mathsf{p2f}(\varphi_1\,\mathbf{T}\,\varphi_2, \Upsilon) \mapsto & \mathsf{p2f}(\neg(\neg\varphi_1\,\mathbf{S}\,\neg\varphi_2), \Upsilon) \\
\mathsf{p2f}(\mathbf{O}\,\varphi, \Upsilon) \mapsto & \mathsf{p2f}(\top\,\mathbf{S}\,\varphi, \Upsilon) \\
\mathsf{p2f}(\mathbf{H}\,\varphi, \Upsilon) \mapsto & \mathsf{p2f}(\neg(\top\,\mathbf{S}\,\neg\varphi), \Upsilon)
\end{aligned}$$

Intuitively, $\mathsf{p2f}(\varphi, \Upsilon)$ recursively replaces each sub-formula φ_i of φ with a top-level past temporal operator with a new distinct fresh propostional variable $v_{\varphi_i} \notin \mathcal{AP}$, and accumulates formulas capturing the semantics of the substituted past temporal sub-formulas in Υ. We remark that the same approach can be equivalently applied to LTL formulas too. In light of this translation, the following theorem follows.

Theorem 4. *Let φ be an LTL$_f$ (or LTL) formula over $\mathcal{AP}$. φ is satisfiable if and only if $\varphi' \wedge \bigwedge_{\rho \in \Upsilon} \rho$ is satisfiable, where $\langle \varphi', \Upsilon \rangle = \mathsf{p2f}(\varphi, \emptyset)$.*

Proof. The proof is by cases on the structure of the formula. We consider only the $\mathbf{Y}$ and $\mathbf{S}$ past temporal operators since $\mathsf{p2f}$ either preserves the formula or rewrites it leveraging the equivalences in Remark 2.1.

- $\mathbf{Y}\,\varphi$.
  $\Longrightarrow$ Let us assume that there exists a trace π such that $\pi, i \models \mathbf{Y}\,\varphi$ for some $i \geq 1$ (i.e., such that $\pi, i-1 \models \varphi$). We can construct a new trace π' extending π to consider a new fresh variable $v_{\mathbf{Y}\,\varphi} \notin \mathcal{AP}$. This new trace π' is such that $\pi'[0] \models_{\mathrm{p}} \neg v_{\mathbf{Y}\,\varphi}$ and for every $i \geq 1$, $\pi'[i] \models_{\mathrm{p}} v_{\mathbf{Y}\,\varphi}$ iff $\pi', i-1 \models \varphi$. Thus, the new trace is such that $\pi' \models \neg v_{\mathbf{Y}\,\varphi} \wedge \mathbf{G}((\mathbf{X}\, v_{\mathbf{Y}\,\varphi}) \leftrightarrow \varphi)$. Moreover, at the time point $i \geq 1$, $\pi'[i] \models_{\mathrm{p}} v_{\mathbf{Y}\,\varphi}$ holds by construction, and thus $\pi', i \models v_{\mathbf{Y}\,\varphi}$.

  $\Longleftarrow$ Let us assume that there exists a trace π such that $\pi \models \neg v_{\mathbf{Y}\,\varphi} \wedge \mathbf{G}((\mathbf{X}\, v_{\mathbf{Y}\,\varphi}) \leftrightarrow \varphi)$ and there exists an $i \geq 1$ such that $\pi, i \models v_{\mathbf{Y}\,\varphi}$. This trace will be such that $\pi[0] \models_{\mathrm{p}} \neg v_{\mathbf{Y}\,\varphi}$ and, for every $i \geq 1$, $\pi[i] \models_{\mathrm{p}} v_{\mathbf{Y}\,\varphi}$ iff $\pi, i-1 \models \varphi$. Thus, it finally holds that $\pi, i \models \mathbf{Y}\,\varphi$.

- $\varphi_1\,\mathbf{S}\,\varphi_2$
  $\Longrightarrow$ Let us assume that there exists a trace π such that $\pi, i \models \varphi_1\,\mathbf{S}\,\varphi_2$ for some $i \geq 0$. This trace is such that there exists a k with $0 \leq k \leq i$ such that $\pi, k \models \varphi_2$ and for every j with $k < j \leq i$ it holds that $\pi, j \models \varphi_1$. We can build a new trace π' extending the trace π to consider a new fresh variable $v_{\varphi_1\,\mathbf{S}\,\varphi_2} \notin \mathcal{AP}$. This new trace π' is such that $\pi'[0] \models_{\mathrm{p}} \neg v_{\varphi_1\,\mathbf{S}\,\varphi_2}$, and, for every $i \geq 1$, $\pi'[i] \models_{\mathrm{p}} v_{\varphi_1\,\mathbf{S}\,\varphi_2}$ iff $\pi', i-1 \models \varphi_2$ or $\pi', i-1 \models \varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2}$. Thus, this new trace is such that $\pi' \models \neg v_{\varphi_1\,\mathbf{S}\,\varphi_2} \wedge \mathbf{G}((\mathbf{X}\, v_{\varphi_1\,\mathbf{S}\,\varphi_2}) \leftrightarrow (\varphi_2 \vee (\varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2})))$, and at time point $i \geq 0$ it holds that $\pi', i \models \varphi_2$ or $\pi', i \models \varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2}$ by construction. Thus, it also holds that $\pi', i \models \varphi_2 \vee (\varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2})$.

  $\Longleftarrow$ Let us assume there is a trace π such that $\pi \models \neg v_{\varphi_1\,\mathbf{S}\,\varphi_2} \wedge \mathbf{G}((\mathbf{X}\, v_{\varphi_1\,\mathbf{S}\,\varphi_2}) \leftrightarrow (\varphi_2 \vee (\varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2})))$ and there exists a time point i such that $\pi, i \models (\varphi_2 \vee (\varphi_1 \wedge v_{\varphi_1\,\mathbf{S}\,\varphi_2}))$. This trace will be such that there exists a k with $0 \leq k \leq i$ such that $\pi, k \models \varphi_2$ and for every j with $k < j \leq i$ it holds that $\pi, j \models \varphi_1$, thus $\pi, i \models \varphi_1\,\mathbf{S}\,\varphi_2$.

□

This result enables the use of any framework for LTL/LTL$_f$ satisfiability checking that does not support past temporal operators.

3.3 Activation Variables

To compute the unsatisfiable core for a given specification $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL$_f$ formulas over $\mathcal{AP}$, we proceed as follows. For each LTL$_f$ formula $\varphi_i \in \Gamma$ we introduce a distinct *activation variable* a_i, i.e., a fresh propositional variable $a_i \in A$, where $A \cap \mathcal{AP} = \emptyset$. Thereupon, we define the LTL$_f$ formula $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$ over $\mathcal{AP} \cup A$. We make the following observation: the satisfiability of Γ is conditioned by the activation variables A, and we have the following theorems.

Theorem 5. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables such that $A \cap \mathcal{AP} = \emptyset$, and $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$ an LTL$_f$ formula defined over $\mathcal{AP} \cup A$. It holds that Γ is unsatisfiable if and only if $\Psi \wedge \bigwedge_{a_i \in A} a_i$ is unsatisfiable.*

Proof. $\Longrightarrow$ Let us assume that Γ is unsatisfiable. This entails that $\bigwedge_{\varphi_i \in \Gamma} \varphi_i$ is unsatisfiable. Let us now consider $\Psi \wedge \bigwedge_{a_i \in A} a_i$, with Ψ and $A \ni a_i$ defined as above, and let us assume it is satisfiable. This means that there exists a trace π such that Ψ is satisfied in the initial state $\pi[0]$ and every $a_i \in A$ is set to true. As a consequence, $a_i \rightarrow \varphi_i$ is satisfied by π for every $i = 1..N$. Therefore, also every $\varphi_i \in \Gamma$ (hence the conjunction of all φ_i formulas, and the set Γ), are satisfiable. However, this contradicts the initial hypothesis.
$\Longleftarrow$ Let us assume that $\Psi \wedge \bigwedge_{a_i \in A} a_i$ is unsatisfiable. This entails that for all subsets $A' \subseteq A$ such that each $a'_i \in A'$ is true and all the other variables in $A \setminus A'$ are false, the conjunction $\bigwedge_{a'_i \in A'} a'_i \rightarrow \varphi_i$ is unsatisfiable. Since every $a'_i \in A'$ is true, this entails that $\bigwedge_{i:a'_i \in A'} \varphi_i$ is unsatisfiable. Let us consider Γ, and assume it is satisfiable. This means that there exists a trace π such that $\pi, 0 \models \bigwedge_{\varphi_i \in \Gamma} \varphi_i$. Therefore, for all $\varphi_i \in \Gamma$, it holds that $\pi, 0 \models \varphi_i$, thus contradicting the above statement that $\bigwedge_{i:a'_i \in A'} \varphi_i$ is unsatisfiable, and hence the hypothesis that $\Psi \wedge \bigwedge_{a_i \in A} a_i$ is unsatisfiable. □

Theorem 6. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ be a set of propositional variables such that $A \cap \mathcal{AP} = \emptyset$, $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and* UC *be a subset of A (i.e., $\mathrm{UC} \subseteq A$). The set $\Phi_{\mathrm{UC}} = \{\varphi_i | a_i \in \mathrm{UC}\}$ is an unsatisfiable core for Γ iff the formula $\Psi \wedge \bigwedge_{a_i \in \mathrm{UC}} a_i$ is unsatisfiable.*

Proof. The proof is analogous to, and a direct consequence of, the proof of Theorem 5. □

This theorem allows us to obtain the unsatisfiable cores (UCs) of Γ by looking at the activation variables that will make Ψ unsatisfiable.

4. Extracting Unsatisfiable Cores for LTL$_\mathbf{f}$

We present here how algorithms A1, ..., A4 (see below for their definition) can be leveraged to define four new algorithms for the extraction of unsatisfiable cores for a given set of LTL$_\mathrm{f}$ formulas, following the two different strategies S1 and S2 highlighted in Fig. 1. Section 4.1 provides the results for Strategy S1 and contains three new algorithms that extend three state-of-the-art algorithms originally developed for LTL, either relying on LTL model checking or on temporal resolution; Section 4.2 instead provides the results for Strategy S2 and contains a new algorithm that extends a reference approach developed for LTL$_\mathrm{f}$ in a native manner.

4.1 Strategy S1: LTL$_\mathbf{f}$ Unsatisfiable Core Extraction via Reduction to LTL Unsatisfiable Core Extraction

This section provides details of how we extract LTL$_\mathrm{f}$ unsatisfiable cores via reduction to LTL satisfiability checking over infinite traces, and via LTL temporal resolution. The first two algorithms we present leverage two different state-of-the-art techniques for LTL satisfiability checking:

A1) an approach based on Binary Decision Diagrams (BDDs, Bryant 1992) as in the work of Clarke et al. (1997); and

A2) a SAT-based approach as presented by Biere et al. (2006).

The third algorithm is based on temporal resolution for LTL (A3, Hustadt and Konev 2003; Schuppan 2016), extended to support past temporal operators.

We leverage Theorem 6 to obtain the unsatisfiable cores (UCs) of $\Gamma = \{\varphi_1, ..., \varphi_N\}$ by looking at the activation variables $A = \{a_1, ..., a_N\}$, with $A \cap \mathcal{AP} = \emptyset$, which makes the formula $\bigwedge_{i=1..N}(a_i \rightarrow \varphi_i) \wedge \bigwedge_{a_i \in A} a_i$ unsatisfiable. In the following, we show how to obtain the UCs using different solving techniques.

4.1.1 Algorithm NA1: BDD-based LTL$_f$ Unsatisfiable Core Extraction

Given the set $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL$_f$ formulas, we build the formula Ψ as discussed in Theorem 5: $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$ for a set $A = \{a_1, ..., a_N\}$ of fresh variables such that $A \cap \mathcal{AP} = \emptyset$. Then, we consider the following LTL formula built leveraging Corollary 1:

$$\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi) \tag{3}$$

The set $[\![\Psi']\!]$ resulting from applying language emptiness algorithms on Ψ' (i.e., BddLtlSat (Ψ') in Algorithm NA1) can be symbolically represented as a propositional formula whose models encode all states that are the initial state of some infinite trace satisfying Ψ'. Notice that formula Ψ' contains the activation variables in A and the variables in $\mathcal{AP}$ alongside the variables in $\mathcal{AP}_{B(\Psi')}$. The latter is needed to encode the symbolic Büchi automaton for Ψ' (see Section 2.2.1 for details).

Theorem 7. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL$_f$ formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$. There exists a state $s \in [\![\Psi']\!]$ and a set $C \subseteq A$ such that $s \models_p \bigwedge_{a_i \in C} a_i$ if and only if $\bigwedge_{i, a_i \in C} \varphi_i$ is satisfiable.*

Proof. $\Longrightarrow$ Suppose there exists a state s in $[\![\Psi']\!]$ such that $s \models_p \bigwedge_{a_i \in C} a_i$. For (AssN1), there exists a trace π starting from s satisfying Ψ'. Since $s \models_p a_i$ for all $a_i \in C$, then trace π also satisfies $\bigwedge_{i, a_i \in C} \varphi_i$.
$\Longleftarrow$ Suppose that $\bigwedge_{i, a_i \in C} \varphi_i$ is satisfiable by some word w over the alphabet $2^{\mathcal{AP}}$. We extend w to w' such that w' satisfies $\bigwedge_{a_i \in C} a_i$. As a result, w' satisfies Ψ'. For (AssN2), there exists a trace π starting from $[\![\Psi']\!]$ that satisfies w'. Then, $\pi[0]$ satisfies $\bigwedge_{a_i \in C} a_i$. □

This theorem allows for the extraction from $[\![\Psi']\!]$ of all possible subsets of the implicants $\varphi_1, ..., \varphi_N$ that are consistent or inconsistent. In particular, given the set of states $[\![\Psi']\!]$ corresponding to the assignments of variables in A, $\mathcal{AP}$ and $\mathcal{AP}_{B(\Psi')}$, the set $\{s|_A \in 2^A \mid \bigwedge_{i, s \in [\![\Psi']\!], s \models_p a_i} \varphi_i$ is unsatisfiable$\}$ can be obtained from $[\![\Psi']\!]$ by quantifying existentially (projecting) the variables corresponding to $\mathcal{AP}$ and $\mathcal{AP}_{B(\Psi')}$ and negating (complementing) the result.

$$\mathrm{UCS}_\Gamma(A) = \neg(\exists \mathcal{AP}.\exists \mathcal{AP}_{B(\Psi')}.[\![\Psi']\!]) \tag{4}$$

$\mathrm{UCS}_\Gamma(A)$ is a propositional formula over variables in A where each satisfying assignment corresponds to an unsatisfiable core for Γ.

Algorithm NA1 BDD-based LTL_f UC extraction with the approach of Clarke et al. (1997)

Input: $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL_f formulas
Output: UC $\subseteq \Gamma$ or $\emptyset$

1: $A \leftarrow \{a_1, ..., a_N\}$ s.t. $A \cap \mathcal{AP} = \emptyset$
2: $\Psi \leftarrow \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$
3: $\Psi' \leftarrow \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$
4: $[\![\Psi']\!] \leftarrow \textsc{BddLtlSat}(\Psi')$
5: UCS $\leftarrow \neg(\exists\mathcal{AP}.\exists\mathcal{AP}_{B(\Psi')}.[\![\Psi']\!])$
6: **if** (UCS $= \emptyset$) **then return** $\emptyset$
7: UC $\leftarrow$ PickOne(UCS)
8: **return** UC

Corollary 2. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL$_f$ formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$, $\text{UCS}_\Gamma(A)$ and $SAT_\Gamma(A)$ propositional formulas over A defined as $\text{UCS}_\Gamma(A) = \neg(\exists\mathcal{AP}.\exists\mathcal{AP}_{B(\Psi')}.[\![\Psi']\!])$ and $SAT_\Gamma(A) = (\exists\mathcal{AP}.\exists\mathcal{AP}_{B(\Psi')}.[\![\Psi']\!])$. Every satisfying assignment for $\text{UCS}_\Gamma(A)$ is an* unsatisfiable *core for Γ: $\{s|_A \in 2^A \mid \bigwedge_{i,s \models_\text{p} a_i} \varphi_i$ is unsatisfiable$\}$. Every satisfying assignment for $SAT_\Gamma(A)$ is a satisfiable subset of Γ: $\{s|_A \in 2^A \mid \bigwedge_{i,s \models_\text{p} a_i} \varphi_i$ is satisfiable$\}$.*

Equation (4) can be easily implemented with BDDs through the respective existential quantification and negation BDD operations (Bryant, 1992; Cimatti et al., 2007).

Algorithm NA1 computes the set of all the unsatisfiable cores (UCS) for a set of LTL_f formulas Γ by leveraging the BDD-based approach discussed by Clarke et al. (1997). It takes as input a set $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL_f formulas over $\mathcal{AP}$. First, it builds a set $A = \{a_1, .., a_N\}$ of distinct activation variables such that $A \cap \mathcal{AP} = \emptyset$. Then it builds the LTL_f formula $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and converts it into the LTL formula $\Psi' \leftarrow \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$ leveraging Corollary 1. Finally, it employs the BddLtlSat algorithm (Clarke et al., 1997) to compute $[\![\Psi']\!]$. Algorithm NA1 returns the empty set $\emptyset$ if the formula is satisfiable, otherwise it returns a UC $\in$ UCS $\subseteq 2^A$ such that for every $s \in$ UCS the formula $\bigwedge_{i,s \models_\text{p} a_i} \varphi_i$ is unsatisfiable.

For a more thorough discussion on how the check for language emptiness is performed, see Section 2.2.1 and the work of Clarke et al. (1997).

Theorem 8. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL$_f$ formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$. Algorithm NA1 returns $\emptyset$ if the set of LTL$_f$ formulas Γ is satisfiable. Otherwise, it computes a set* UCS $\neq \emptyset$ *such that, for every* UC $\in$ UCS, *$\Phi_\text{UC} = \{\varphi_i \mid a_i \in \text{UC}\}$ is an unsatisfiable core for Γ, and then returns a* UC $\in$ UCS, *which corresponds to an unsatisfiable core for Γ.*

Proof. The proof is a direct consequence of Theorem 7 and Corollary 2. Indeed, $\textsc{BddLtlSat}(\Psi')$ computes $[\![\Psi']\!]$, i.e., the set of states that are a starting point of some trace satisfying Ψ'. If Γ is satisfiable, then any of its subsets is satisfiable as well. Therefore, any possible assignment to a_i will be such that Ψ' is satisfiable. As a consequence,

$SAT_\Gamma(A) \neq \emptyset$, and thus $\text{UCS}_\Gamma(A) = \emptyset$ as per Line 3 of Algorithm NA1. On the other hand, if Γ is unsatisfiable, Equation 4 extracts the formula over variables a_i such that every satisfying assignment for such formula corresponds to an unsatisfiable core for Ψ'. Notice that an unsatisfiable core for Ψ' is an unsatisfiable core for Γ, in turn. Therefore, the algorithm selects one of these assignments with the PickOne(UCS) operation on Line 4 of Algorithm NA1, thereby yielding an unsatisfiable core for Γ. □

4.1.2 Algorithm NA2: SAT-based LTL$_\text{f}$ Unsatisfiable Core Extraction

Determining language emptiness of an LTL formula can also be performed leveraging any off-the-shelf technique for SAT-based bounded model checking (BMC) equipped with completeness check (Biere et al., 2006; Claessen & Sörensson, 2012). We observe that all these approaches can be easily extended to extract an unsatisfiable core from a conjunction of temporal constraints by leveraging the ability of propositional SAT solvers to check the satisfiability of a propositional formula ψ under a set of assumptions specified in the form of literals $L \ni l_j$, ψ' turns out to be unsatisfiable, then the SAT solver can return a subset $\text{UC} \subseteq L$ such that $\psi \wedge \bigwedge_{l_j \in \text{UC}} l_j$ is unsatisfiable. SAT-based bounded model checking (Biere et al., 2003) encodes a finite trace of length k with a propositional formula over the set of variables representing the $\mathcal{AP}$ at each time step from 0 to k. To check for completeness, they typically encode the fact that the trace cannot be extended with states not yet visited (Biere et al., 2006). We remark that, in model checking, one considers both a transition system (i.e., a model) and a temporal logic formula. However, since we focus on satisfiability of LTL formulas only, we consider as a transition system the *universal model* (i.e., given a set of propositional variables $\mathcal{AP}$, the initial set of states is $2^{\mathcal{AP}}$, and the transitions relation is $2^{\mathcal{AP}} \times 2^{\mathcal{AP}}$). Notice that this operation corresponds to encoding symbolically both the initial set of states and the transition relation with $\top$.

The approach proceeds as illustrated in Algorithm NA2. Similarly to Algorithm NA1, Algorithm NA2 takes as input a set $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL$_\text{f}$ formulas over $\mathcal{AP}$. First, it builds a set $A = \{a_1, .., a_N\}$ of distinct activation variables such that $A \cap \mathcal{AP} = \emptyset$. Then it builds the LTL$_\text{f}$ formula $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and converts it into the LTL formula $\Psi' \leftarrow \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$ leveraging Corollary 1. It computes an unsatisfiable core for the set of LTL$_\text{f}$ formulas Γ leveraging the bounded model checking encoding defined by Biere et al. (2006). To this end, it uses a *completeness formula* $\text{EncC}(\phi, k)$, which is unsatisfiable iff the LTL formula ϕ is unsatisfiable, and a *witness formula* $\text{EncP}(\phi, k)$, which is satisfiable iff ϕ is satisfiable by a trace of length k.[5] For increasing values of k, we submit to the SAT solver a propositional encoding up to the considered k of a trace satisfying the formula Ψ' under the assumption that all the literals in A are true in the initial time step 0. In Algorithm NA2, we denote these literals with $A^{[0]} = \bigwedge_{a_i \in A} a_i^{[0]}$ and call SAT_Assume($\text{EncC}(\Psi', k)$, $A^{[0]}$), which checks the satisfiability of $\text{EncC}(\Psi', k)$ under the assumption that the literals in $A^{[0]}$ are true. When the call proves the formula unsatisfiable by returning *UNSAT*, it is straightforward to get one propositional unsatisfiable core from the SAT solver in terms of a subset of the variables in $A^{[0]}$, thus concluding the search. However, if the call returns *SAT*, we cannot conclude that the LTL formula is unsatisfiable.

5. We refer the reader to the work of Biere et al. (2006) for details on how the propositional formulas are constructed by$\text{EncC}(\phi, k)$ and $\text{EncP}(\phi, k)$.

Algorithm NA2 SAT BMC-based LTL_f UC extraction with the approach of Biere et al. (2006)

Input: $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL_f formulas
Output: UC $\subseteq \Gamma$ or $\emptyset$

1: $A \leftarrow \{a_1, ..., a_N\}$ s.t. $A \cap \mathcal{AP} = \emptyset$
2: $\Psi \leftarrow \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$
3: $\Psi' \leftarrow \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$
4: $k \leftarrow 0$
5: **while** (True) **do**
6: res, UC $\leftarrow$ SAT_Assume(EncC(Ψ', k), $A^{[0]}$)
7: **if** (res = *UNSAT*) **then return** UC
8: res $\leftarrow$ SAT(EncP$(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]}$)
9: **if** (res = *SAT*) **then return** $\emptyset$
10: $k \leftarrow k + 1$

In such a case, we check whether a lasso-shaped trace of length k exists that satisfies the propositional formula $\text{EncP}(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]}$. This is achieved with the new call $\text{SAT}(\text{EncP}(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]})$.

If *SAT* is returned, then we can conclude that the LTL formula is satisfiable and this information can be noted. Otherwise, we increase k and iterate. This approach is guaranteed to eventually terminate (Biere et al., 2006).

To sum up, Algorithm NA2 takes as input a set of LTL_f formulas $\Gamma = \{\varphi_1, .., \varphi_N\}$ and returns the empty set ($\emptyset$) if the specification is satisfiable, otherwise it returns a subset UC $\subseteq A$ such that $\bigwedge_{i, a_i \in \text{UC}} \varphi_i$ is unsatisfiable. Equipped with these notions, we prove the following.

Lemma 1. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL_f formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL_f formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$. Given l, k such that $0 \leq l \leq k$, we have that $\text{EncP}(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]}$ is satisfiable iff there exists a lasso-shaped trace $\pi[0, l-1]\pi[l, k]^\omega$ such that $\pi \models \bigwedge_{i, a_i \in A} \varphi_i$.*

Proof. $\Longrightarrow$ Let $\pi[0, k]$ be a finite trace corresponding to a satisfying assignment for $\text{EncP}(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]}$. From assumption (AssN1), there exists an l (with $0 \leq l \leq k$) such that $\pi[0, l-1]\pi[l, k]^\omega \models \Psi'$. With $\bigwedge_{a_i \in A} a_i^{[0]}$ and the construction of Ψ', we conclude that the projection of $\pi[0, l-1]\pi[l, k]^\omega$ onto $\mathcal{AP}$ satisfies $\bigwedge_{i, a_i \in A} \varphi_i$.
$\Longleftarrow$ Let us assume a lasso shaped witness $\pi[0, l-1]\pi[l, k]^\omega$ for $\bigwedge_{i, a_i \in A} \varphi_i$. Any extension to $\pi[0, l-1]\pi[l, k]^\omega$ such that $\bigwedge_{a_i \in A} a_i^{[0]}$ holds in the initial state, is a trace satisfying Ψ'. As a consequence, also $\text{EncP}(\Psi', k) \wedge \bigwedge_{a_i \in A} a_i^{[0]}$ is satisfiable. □

Lemma 2. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL_f formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL_f formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$, and UC $\subseteq A$. If $\text{EncC}(\Psi', k)$ is unsatisfiable under the assumption that $\bigwedge_{a_i \in \text{UC}} a_i^{[0]}$ holds true, then $\bigwedge_{i, a_i \in \text{UC}} \varphi_i$ is unsatisfiable.*

Proof. Let EncC(Ψ', k) be unsatisfiable under the assumption $\bigwedge_{a_i \in \text{UC}} a_i^{[0]}$. Let us assume there exists a witness π of $\bigwedge_{i,a_i \in \text{UC}} \varphi_i$. We can extend the π trace to a trace π' over variables $\mathcal{AP} \cup A$ such that $\bigwedge_{a_i \in \text{UC}} a_i^{[0]}$ is a trace satisfying Ψ', thus contradicting the assumption (AssN2). □

Theorem 9. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$. Algorithm NA2 returns $\emptyset$ if the set of LTL$_f$ formulas Γ is satisfiable, otherwise it returns a $\text{UC} \neq \emptyset$ such that the set $\Phi_{\text{UC}} = \{\varphi_i | a_i \in \text{UC}\}$ is an unsatisfiable core for Γ.*

Proof. The proof is a direct consequence of Lemma 1 and Lemma 2. □

Algorithm NA2 uses the encoding of Biere et al. (2006) for both EncC(Ψ', k) and EncP(Ψ', k). We remark that the schema can also be adapted to leverage other algorithms such as the ones based on k-liveness (Claessen & Sörensson, 2012) or liveness to safety (Biere et al., 2002), both relying on the IC3 algorithm (Bradley, 2011). What intuitively changes is the propositional encoding of the LTL formula and the calls to the SAT solver to reflect the IC3 algorithm. However, this is left as future work.

4.1.3 Algorithm NA3: Temporal Resolution based LTL$_f$ Unsatisfiable Core Extraction

We can extract the unsatisfiable core of a set of LTL$_f$ formulas $\Gamma = \{\varphi_1, ..., \varphi_N\}$ via LTL temporal resolution (TR, Hustadt and Konev 2003) by leveraging the results previously discussed in this paper and existing LTL temporal resolution engines equipped for temporal unsatisfiable core extraction (Schuppan, 2016). Algorithm NA3 computes an unsatisfiable core for a set of LTL$_f$ formulas $\Gamma = \{\varphi_1, ..., \varphi_N\}$ as follows. First, it creates a set A of fresh propositional variables $A = \{a_1, .., a_N\}$ such that $A \cap \mathcal{AP} = \emptyset$. Second, it builds the formula $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$. Third, it leverages Theorem 1 to convert the LTL$_f$ formula into an equi-satisfiable LTL formula $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$. Fourth, it applies Theorem 4 to remove the past temporal operators in Ψ', and enforces each activation variable $a_i \in A$ to hold, thus it builds the LTL formula $\psi \leftarrow \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi \wedge \bigwedge_{a_i \in A} a_i$. Finally, the resulting LTL formula ψ is given as input to any LTL temporal resolution solver suitable to extract a temporal unsatisfiable core. In particular, we rely on the TRP++ temporal resolution solver (Schuppan, 2016). If the LTL temporal resolution solver responds *UNSAT*, we get an unsatisfiable core of the original set of LTL$_f$ formulas by looking at the activation variables in the extracted temporal unsatisfiable core UC_ψ.

Algorithm NA3 takes as input the set $\Gamma = \{\varphi_1, .., \varphi_N\}$ and returns the empty set ($\emptyset$) if Γ is satisfiable. Otherwise, it returns a subset $\text{UC} \subseteq A$ such that $\bigwedge_{i,a_i \in \text{UC}} \varphi_i$ is unsatisfiable. It uses the TRP++ algorithm (Schuppan, 2016) to compute the unsatisfiable core UC_ψ, and then it extracts from it only the formulas corresponding to $a_i \in A$ (denoted in the algorithm with $\text{UC}_\psi|_A$). The TRP++(ψ) algorithm introduced by Schuppan (2016) first converts the LTL formula ψ into an equi-satisfiable set of Separated Normal Form (SNF, Fisher 1991) clauses C, and then it checks whether this set is satisfiable or not. In case of unsatisfiability, it computes an unsatisfiable core $C^{\text{uc}} \subseteq C$ and returns the set UC_ψ obtained from C^{uc} by

Algorithm NA3 TR LTL_f UC Extraction with the approach of Schuppan (2016)

Input: $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL_f formulas
Output: $\text{UC} \subseteq \Gamma$ or $\emptyset$

1: $A \leftarrow \{a_1, ..., a_N\}$ s.t. $A \cap \mathcal{AP} = \emptyset$
2: $\Psi \leftarrow \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$
3: $\Psi' \leftarrow \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$
4: $\langle \phi', \Upsilon \rangle \leftarrow \mathsf{p2f}(\Psi', \emptyset)$
5: $\psi \leftarrow \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi \wedge \bigwedge_{a_i \in A} a_i$
6: $\text{res}, \text{UC}_\psi \leftarrow \text{TRP++}(\psi)$
7: **if** ($\text{res} = \mathit{UNSAT}$) **then return** $\text{UC}_\psi|_A$
8: **return** $\emptyset$

applying a reconstruction with respect to the original set of LTL formulas. We refer the reader to the work of Schuppan (2016) for more details on the algorithm and the proof of correctness of the TRP++ algorithm.

Theorem 10. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL_f formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL_f formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, Ψ' an LTL formula defined as $\Psi' = \mathbf{F}\,\mathsf{end} \wedge \mathbf{G}(\mathsf{end} \rightarrow \mathbf{X}\,\mathsf{end}) \wedge \mathsf{f2l}(\Psi)$, and ψ an LTL formula defined as $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi \wedge \bigwedge_{a_i \in A} a_i$ where $\langle \phi', \Upsilon \rangle = \mathsf{p2f}(\Psi', \emptyset)$. Then Algorithm NA3 returns $\emptyset$ if the set of LTL_f formulas Γ is satisfiable, otherwise it returns a $\text{UC} \neq \emptyset$ such that the set $\Phi_{\text{UC}} = \{\varphi_i | a_i \in \text{UC}\}$ is an unsatisfiable core for Γ.*

Proof. If the set Γ is satisfiable, then the formula $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi \wedge \bigwedge_{a_i \in A} a_i$ where $\langle \phi', \Upsilon \rangle = \mathsf{p2f}(\Psi', \emptyset)$, is satisfiable since it leverages on transformations that preserve satisfiability (see Theorems 1, 4, and 6). Therefore, also $\text{TRP++}(\psi)$ returns as an answer that Γ is satisfiable, and the algorithm yields the empty set $\emptyset$ to indicate this.

On the other hand, if Γ is unsatisfiable, then also $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi \wedge \bigwedge_{a_i \in A} a_i$ is unsatisfiable. Therefore, $\text{TRP++}(\psi)$ returns *UNSAT* together with an unsatisfiable core for the formula ψ (denoted UC_ψ in the algorithm). We remark that, given the structure of ψ, each a_i will be then converted into an SNF clause $c_{a_i} = a_i$ which will thus be part of the set of SNF clauses C used internally by TRP++. Since this formula is unsatisfiable, the TRP++ algorithm will extract an unsatisfiable core $\text{UC}_\psi = C^{\text{uc}} \subseteq C$ such that C^{uc} is unsatisfiable. C^{uc}, among other clauses, will contain some c_{a_i} for $a_i \in A$ that correspond to the respective formulas in Γ (thanks also to Theorem 6). Therefore, the set $\text{UC}_\psi = C^{uc}$ restricted to the only variables $a_i \in A$ (denoted with $\text{UC}_\psi|_A$ in the algorithm) will represent an unsatisfiable core for Γ. □

4.2 Strategy 2: LTL_f Unsatisfiable Core Extraction via Native SAT

This section provides details on how we adapted the native SAT-based LTL_f satisfiability approach discussed by Li et al. (2020) to extract the unsatisfiable core. Since the original approach for LTL_f satisfiability checking was not supporting past temporal operators (Li et al., 2020), we rely on Theorem 4 to get rid of the past temporal operators, thus obtaining an equi-satisfiable LTL_f formula without them.

Algorithm NA4 SAT LTL$_f$ UC Extraction with the approach of Li et al. (2020)

Input: $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL$_f$ formulas
Output: UC $\subseteq \Gamma$ or $\emptyset$

1: $A \leftarrow \{a_1, ..., a_N\}$ s.t. $A \cap \mathcal{AP} = \emptyset$
2: $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$
3: $\langle \phi', \Upsilon \rangle \leftarrow \mathsf{p2f}(\Psi, \emptyset)$
4: $\psi \leftarrow \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi$
5: res $\leftarrow$ SATLTLF(ψ, A)
6: **if** (res = *UNSAT*) **then return** SAT$_{\text{SOLVER}}$.GET_UC (A)
7: **return** $\emptyset$

4.2.1 Algorithm NA4: LTL$_f$ SAT- based LTL$_f$ Unsatisfiable Core Extraction

The algorithm introduced by Li et al. (2020) can be extended to extract the unsatisfiable core of a set $\Gamma = \{\varphi_1, ..., \varphi_N\}$ of LTL$_f$ formulas over $\mathcal{AP}$ as follows (see Algorithm NA4). First, similarly to the other algorithms in this paper, we build a set $A = \{a_1, .., a_N\}$ of fresh propositional variables such that $A \cap \mathcal{AP} = \emptyset$. Second, we build the LTL$_f$ formula $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$. Then, we apply Theorem 4 to get an equi-satisfiable LTL$_f$ formula ψ without temporal past operators such that $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi$ where $\langle \phi', \Upsilon \rangle = \mathsf{p2f}(\Psi, \emptyset)$. The resulting LTL$_f$ formula (without past temporal operators) is then passed as input to the SATLTLF algorithm discussed in the work of Li et al. (2020). The SATLTLF algorithm in Algorithm NA4 is almost identical to the original one presented in (Li et al., 2020). We modify each internal call SAT_Assume by enforcing that each of those calls also assumes that the activation variables in A are all true. We refer the reader to the work of Li et al. (2020) for a thorough description of the algorithm (which is out of the scope of this paper). We remark that the only modifications performed to the Li et al. (2020) algorithm consist in changing each call to SAT_Assume to also enforce that the activation variables in A are all true. Intuitively, given an LTL$_f$ formula ϕ, the algorithm by Li et al. (2020) constructs a *conflict sequence* $\mathcal{C} = \mathcal{C}[0], ..., \mathcal{C}[k]$ (i.e., a sequence of states that cannot reach a final state of the transition system T_ϕ constructed from the formula ϕ given as input). This sequence is extracted from the unsatisfiable cores resulting from different propositional unsatisfiable queries performed within the algorithm itself. As per Theorem 3, the input LTL$_f$ formula is unsatisfiable iff there exists a conflict sequence $\mathcal{C}$ and an integer $i \geq 0$ such that $\bigcap_{0 \leq j \leq i} \mathcal{C}[j] \subseteq \mathcal{C}[i+1]$. When the SATLTLF algorithm returns *UNSAT*, we extract from the last element of the computed conflict sequence (i.e., $\mathcal{C}[i+1]$) the unsatisfiable core UC $\subseteq A$, and the set $\Gamma' = \{\varphi_i | a_i \in \text{UC}\}$ is an unsatisfiable core for Γ leveraging Theorem 3.

Theorem 11. *Let $\Gamma = \{\varphi_1, ..., \varphi_N\}$ be a set of LTL$_f$ formulas over $\mathcal{AP}$, $A = \{a_1, ..., a_N\}$ a set of propositional variables with $A \cap \mathcal{AP} = \emptyset$, Ψ an LTL$_f$ formula defined as $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, and ψ an LTL$_f$ formula defined as $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi$ where $\langle \phi', \Upsilon \rangle \leftarrow \mathsf{p2f}(\Psi, \emptyset)$. Then Algorithm NA4 returns $\emptyset$ if the set of LTL$_f$ formulas Γ is satisfiable, otherwise it returns a UC $\neq \emptyset$ such that the set $\Phi_{\text{UC}} = \{\varphi_i | a_i \in \text{UC}\}$ is an unsatisfiable core for Γ.*

Proof. If the set Γ is satisfiable, then the formula $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi$, where $\langle \phi', \Upsilon \rangle = \mathsf{p2f}(\Psi, \emptyset)$ and $\Psi = \bigwedge_{i=1..N}(a_i \rightarrow \varphi_i)$, is also satisfiable since it resorts to transformations that preserve satisfiability (see Theorems 4, 1 and 6). Thus, as SATLTLF is correct and complete (Li et

Table 1: The tools implementing the LTL_f unsatisfiable core extraction algorithms

Algorithm	See	Tool name	Based upon	Augmentation	Available at
NA1	Section 4.1.1	NuSMV-B	(Cimatti et al., 2002; Clarke et al., 1997)	Enhancement	`https://github.com/roveri-marco/ltlfuc`
NA2	Section 4.1.2	NuSMV-S	(Biere et al., 2006; Cimatti et al., 2002)	Enhancement	`https://github.com/roveri-marco/ltlfuc`
NA3	Section 4.1.3	trp++	(Hustadt & Konev, 2003; Schuppan, 2016)	Toolchain	`http://www.schuppan.de/viktor/trp++uc/`
NA4	Section 4.2	aaltaf	(Li et al., 2020)	New module	`https://github.com/roveri-marco/aaltaf-uc`

al., 2020), it declares that Γ is satisfiable. In turn, Algorithm NA4 returns the empty set $\emptyset$ to indicate this. On the other hand, if Γ is unsatisfiable, then also the formula $\psi = \phi' \wedge \bigwedge_{\varphi \in \Upsilon} \varphi$ is unsatisfiable. As per Theorem 3, the SATLTLF algorithm will build a conflict sequence $\mathcal{C}$, and $i \geq 0$ such that $\bigcap_{0 \leq j \leq i} \mathcal{C}[j] \subseteq \mathcal{C}[i+1]$, $\mathcal{C}[i+1]$ will represent the states from which it is not possible to reach a final state for T_ψ (i.e., there is no trace starting from these states that satisfies ψ, or, put in other words, all the traces from these states do not satisfy ψ). Thus, for all states $s \in \mathcal{C}[i+1]$ there exists a set $C \subseteq A$ such that $s \models_\text{p} \bigwedge_{a_i \in C} a_i$, the formula $\bigwedge_{i, a_i \in C} \varphi_i$ is unsatisfiable, and the set C corresponds to an unsatisfiable core extracted from $\mathcal{C}[i+1]$ by construction of the SATLTLF algorithm. We refer the reader to the work of Li et al. 2020 for further details in this regard. □

4.3 Discussion

We make the following observations. All the described approaches extract one unsatisfiable core, though not necessarily a minimum/minimal one. Algorithm NA1 could also be easily extended to get the minimal UC from the UCS set of all possible unsatisfiable cores for the given formula. For the SAT-based approaches, a minimum/minimal unsatisfiable core could be extracted by leveraging the ability of the SAT solver to get a minimum/minimal propositional unsatisfiable core. Similarly, the temporal resolution solver could be instrumented to get a minimum/minimal core. In all cases, it might be possible to get a minimum/minimal one with specialised solvers and/or with additional search. However, this is left for future work.

5. Experimental Evaluation

In this section, we provide details on the implementations of the proposed algorithms (Section 5.1), and then we describe the setup and the data sets used for the experimental evaluation (Section 5.2). We conclude with a report on the results alongside an examination thereof (Section 5.3).

5.1 Implementation of the Algorithms

Table 1 summarises our implementations of the four algorithms described in Section 4. We realise Algorithms NA1 and NA2 as extensions of the NuSMV model checker (Cimatti et al., 2002) exploiting the built-in support for past temporal operators, the $\mathsf{f2l}(\varphi)$ conversion, and Eq. (2). In particular, we enhanced (i) the BDD-based algorithm for LTL language emptiness (Clarke et al., 1997) and (ii) the SAT-based approaches (Biere et al., 2006). We shall henceforth refer to these tools as NuSMV-B and NuSMV-S, respectively. The

Table 2: The benchmarks used in our experiments. For the sake of readability, we compactly denote with `$A` the *LTL-as-LTLf* root, and with `$B` the *LTLf-specific* root.

		Clauses					Clauses		
Family	**Problems**	**Min.**	**Max.**	**Avg.**	**Family**	**Problems**	**Min.**	**Max.**	**Avg.**
`$A`/*acacia/demo-v3/demo-v3:cl*	11	9	49	29.00	`$A`/*anzu/genbuf/genbuf:cl*	20	58	461	231.50
`$A`/*alaska/lift/lift*	17	13	29	21.00	`$A`/*forobots*	38	6	6	6.00
`$A`/*alaska/lift/lift:b*	17	12	32	22.47	`$A`/*rozier/counter/counter*	19	6	24	15.00
`$A`/*alaska/lift/lift:b:f*	17	12	32	22.47	`$A`/*rozier/counter/counterCarry*	19	8	26	17.00
`$A`/*alaska/lift/lift:b:f:l*	17	14	50	32.47	`$A`/*rozier/counter/counterCarryLinear*	19	8	8	8.00
`$A`/*alaska/lift/lift:b:l*	17	14	50	32.47	`$A`/*rozier/counter/counterLinear*	18	6	6	6.00
`$A`/*alaska/lift/lift:f*	17	13	29	21.00	`$A`/*rozier/formulas/n*	30	1	4	1.33
`$A`/*alaska/lift/lift:f:l*	17	15	47	31.00	`$A`/*schuppan/O1formula*	27	4	1002	224.00
`$A`/*alaska/lift/lift:l*	17	15	47	31.00	`$A`/*schuppan/O2formula*	27	2	1000	222.00
`$A`/*anzu/amba/amba:c*	17	75	351	213.47	`$A`/*schuppan/phltl*	13	5	101	30.85
`$A`/*anzu/amba/amba:cl*	17	77	369	223.47	`$B`/*benchmarks:ltlf/LTLfRandomConjunction/C100*	500	118	154	131.50
`$A`/*anzu/genbuf/genbuf*	20	1	1	1.00	`$B`/*benchmarks:ltlf/LTLfRandomConjunction/V20*	425	13	146	81.54
`$A`/*anzu/genbuf/genbuf:c*	20	57	441	221.00	**Overall**	1377	1	1002	97.63

source code for the extended version of NuSMV with these implementations is available at `https://github.com/roveri-marco/ltlfuc`.

We create a toolchain for Algorithm NA3. First, our variant of AALTAF generates a file that is suitable for the TRP++ temporal resolution solver (Hustadt & Konev, 2003) using the $\mathsf{f2l}(\varphi)$ conversion as per Eq. (2), and $\mathsf{p2f}(\varphi, \emptyset)$. Then, the resulting file is submitted to TRP++. Finally, the generated UC is post-processed to extract the auxiliary variables A. For our experiments, we use the latest version of TRP++.[6]

Finally, we implement Algorithm NA4 within an extended version of the AALTAF tool (Li et al., 2020), with a novel dedicated module supporting past temporal operators through $\mathsf{p2f}(\varphi, \emptyset)$. The source code for our extended version of AALTAF is available at `https://github.com/roveri-marco/aaltaf-uc`.

5.2 The Experimental Setup

For the experimental evaluation, we considered all the unsatisfiable problems reported in the work of Li et al. (2020), for a total of 1377 problems. To select the specifications of interest to our analysis from the original testbed, we included only those for which at least one solver declared that the set was unsatisfiable and no other tool contradicted the result, as per the experimental data reported by Li et al. (2020). To compute the Γ set, we considered all the top-level conjuncts of each formula in the benchmark set. For every benchmark, we used the variant of the formula in the AALTAF format as an input. For the other tools except AALTAF, we implemented dedicated modules within AALTAF to convert the native input encodings into its accepted format.

We carried out the experimental evaluation considering the four implementations provided by NuSMV-B, NuSMV-S, our variant of AALTAF, and the TRP++ toolchain. We ran all experiments on an Ubuntu 18.04.5 LTS machine, 8-Core Intel® Xeon® at 2.2 GHz, equipped with 64 GB of RAM. We set a memory occupation limit of 4 GB, and a CPU usage

6. `http://www.schuppan.de/viktor/trp++uc/`.

limit of 60 min.[7] Additionally, we considered $k = 50$ as the maximum depth for NuSMV-S, and we ran NuSMV-B with the BDD dynamic variable reordering mode active (Felt et al., 1993) to dynamically reduce the size of the BDDs and thus save space over time.[8] Whenever the wall-clock timing reported by the implemented technique fell under the lowest sensitivity of the tool, we replaced the timing with the minimum non-zero timing reported overall (i.e., 3.78×10^{-4} s).

Finally, we categorised the benchmarking specifications into 25 families, according to their characteristics and provenance. Table 2 shows the number of specifications per family, along with the minimum, maximum and average number of clauses within it. In particular, the *LTLf-specific/benchmarks/LTLFRandomConjunction/V20* and *LTLf-specific/benchmarks/LTLFRandomConjunction/C100* benchmarks are conjunctions of formulas, each selected randomly from standard patterns. They are characterised by a *temporal depth* (i.e., the maximum nesting of temporal operators) of up to 3, with 20 propositional variables. The number of conjunctions ranges from 20 to 100 for the former, and from 10 to 100 for the latter. The *LTL-as-LTLf/rozier/counter/** benchmarks[9] are characterised by the fact that they have temporal formulas of different temporal depths (from 2 to 20) with a small number of propositional variables. These formulas are a conjunction of subformulas characterised by a top level **G** whose body contains a nested chain of 2 to 20 **X**'s. The benchmarks in *LTL-as-LTLf/schuppan/O1Formula* contain a large number of propositional variables (from 1 to 1000) with temporal formulas of small depth (2 to 3) and different operators. The benchmarks in *LTL-as-LTLf/schuppan/O2Formula* are big conjunctions of formulas in the form $\mathbf{G}\,\mathbf{F}\,a_i \leftrightarrow a_j$ with $a_i \neq a_j$.

All the material to reproduce the experiments reported hereinafter is available at `https://github.com/roveri-marco/ltlfuc/archive/refs/tags/jair-release-v1.0.zip`.

5.3 The Results

In the experimental evaluation, we consider the following evaluation metrics: (i) the result of the check (expecting all the tools to return unsatisfiability and extract an unsatisfiable core if no resource limit is reached); (ii) the search time to compute and return an unsatisfiable core; (iii) the size of the computed unsatisfiable core. We remark that none of the presented algorithms strives for finding a minimum unsatisfiable core. The approach based on TRP++ may be used to that end, and NuSMV-B could in principle be easily adapted to select from the intermediate computed set UCS a minimum unsatisfiable core (as discussed previously). Nevertheless, we pick the first returned UC for all tools so as to have a fair comparison among them.

7. These settings are motivated by similar choices performed in the experimental evaluations carried out in Li et al. (2020).
8. These settings are motivated by similar choices performed in the experimental evaluations carried out by Cimatti et al. (2007); Schuppan (2016).
9. For the sake of conciseness, we use the /* shorthand notation to compactly indicate all benchmark families under a given root. For example, *LTL-as-LTLf/rozier/counter/** is a collective identifier for the *LTL-as-LTLf/rozier/counter/counter*, *LTL-as-LTLf/rozier/counter/counterCarry*, *LTL-as-LTLf/rozier/counter/counterCarryLinear*, and *LTL-as-LTLf/rozier/counter/counterLinear* benchmark families.

The first result is that, as expected, all the tools reported consistent output when terminating without reaching a resource limit (being it memory, time or search-space depth). In other words, for all the considered benchmarks it was never the case that an algorithm declared the specification as satisfiable. This outcome is in line with the original findings of Li et al. (2020). We also checked that every computed core was unsatisfiable by feeding it into the aaltaf algorithm. However, we remark that individual algorithms could extract different unsatisfiable cores among the diverse possible ones.

In the following, we further investigate the performance of the algorithms in terms of *efficiency* and *efficacy*. We gauge the former taking into account the time the tools take to find a UC. We measure the latter considering the cardinality of the returned UC. In principle, indeed, a tool could just return the whole input set of constraints as unsatisfiable. Such an output would be very fast to compute, and the tool would be deemed as highly efficient. However, the outcome would be of scarce informativeness.

In the remainder of the section, we shall consider the sole cases in which the tools were able to return a UC within the given resource limits (thus excluding timeouts and unknown answers), unless explicitly stated otherwise. The presentation of the experimental results proceeds as follows. Section 5.3.1 shows how tools rank in terms of both criteria (i.e., execution time and UC cardinality). That section serves as a general introduction to the following, more detailed analysis. Section 5.3.2 focus on how algorithms cumulatively perform in terms of execution time. Section 5.3.3 delves deeper into this topic by illustrating pairwise comparisons of the tools' efficiency. Then, Section 5.3.4 categorizes the performance of tools based on the benchmark families, and Section 5.3.5 reports on our studies pertaining to the effect that the number of conjuncts in formulas have on execution time. Section 5.3.6 moves the focus away from time perfomance and shifts it onto the efficacy of tools, with a comparative evaluation of the cardinality of the UCs returned by the tools. Analogously to Section 5.3.4, the section also categorises the results with respect to the benchmark family of the input. Afterwards, Section 5.3.7 illustrates pairwise comparisons of the tools' efficacy, akin to our analysis reported in Section 5.3.3 for efficiency. To conclude, Section 5.3.8 offers a summary of the findings.

5.3.1 Time Performance and Cardinality of the UC

The Sankey chart in Fig. 2 compactly depicts two-staged rankings: speed of computation *and* cardinality of the computed UC. In the rankings here we allow for ties (for example, two tools can occupy the first position together). As it turns out, aaltaf is the fastest tool in the majority of tests. However, it returns the UCs that are smallest in size with about half of the test cases. Although trp++ usually occupies the second position in the time ranking, the cardinality of the returned UC usually is the lowest whenever it finds one. Also, notice that trp++ returns the smallest UC in 680 cases, that is basically as many times as aaltaf (with 681 cases). NuSMV-B and NuSMV-S manage to return an unsatisfiable core less often than the other two tools. However, especially NuSMV-B yields comparably small unsatisfiable cores whenever it succeeds in finding one under the imposed time constraints. We recall that we exclude from the plot also the points representing returned unknown answers, which motivates the low number of entries for

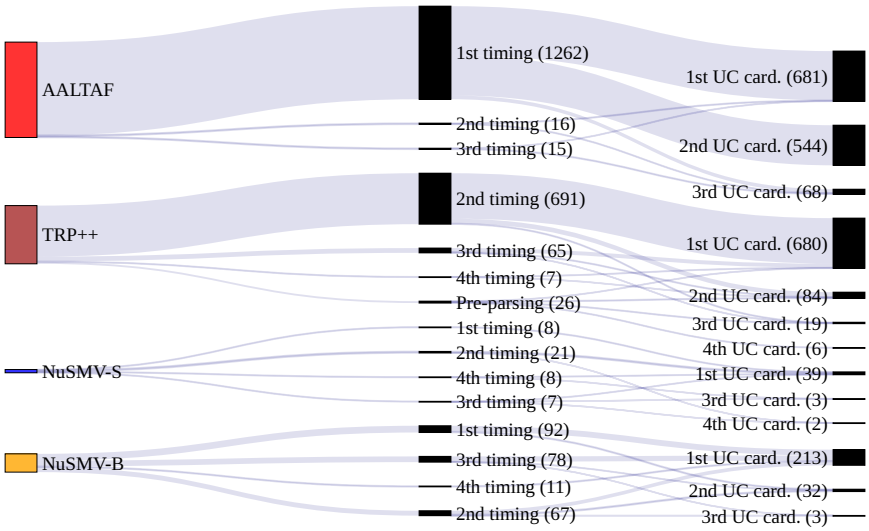


Figure 2: Sankey chart displaying the tool rankings based on the computation time and the cardinality of the returned UCs.

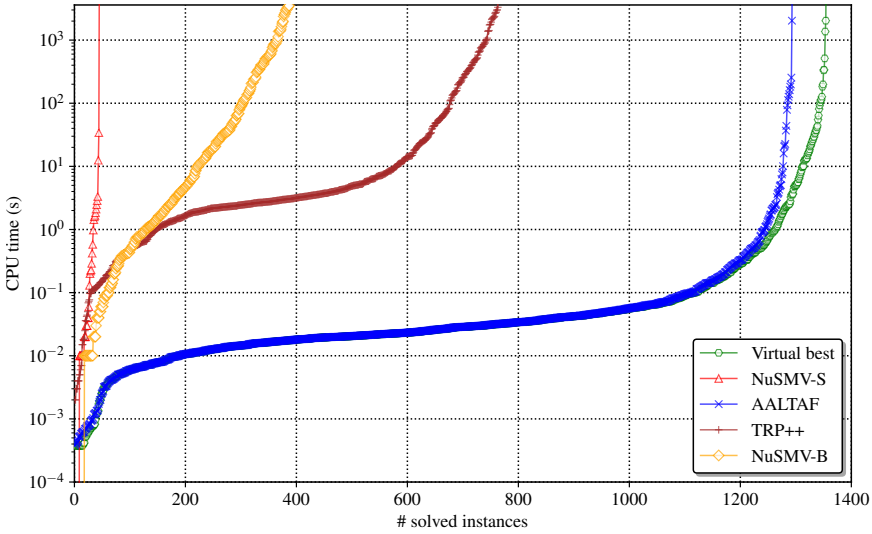


Figure 3: Cactus plot of the experimental evaluation. The cactus plot, typically used by the SAT and SMT community for benchmarking automated verification tools, reports the number of solved instances on the x-axis, and the cumulative computation time required by the tools on the y-axis

.

NuSMV-S overall. Next, we discuss the factors that led to the different performance levels in a more in-depth comparative assessment, starting with time.

5.3.2 Execution Time

Figure 3 shows on the x-axis the number of problems solved by each algorithm (within the 60 min timeout), and on the y-axis the time taken to solve them cumulatively. Alongside the aforementioned tools, the figure illustrates the performance of the *virtual best*, that is the minimum time required for each solved instance among the four implementations. As above, we exclude from the plot the points representing runs that do not return a UC. The overall minimum, maximum, average, and median timings to return a UC are 0.0004 s, 3478.96 s, 50.7472 s and 0.0812 s, respectively. The maximum, average, and median timings shrink to 2029.7347 s, 4.8534 s and 0.0274 s, respectively, for the virtual best. We observe that aaltaf outperforms the other implementations in the majority of cases, although the tail of the virtual-best curve on the right-hand side of both plots exhibits an influence from trp++ and NuSMV-B, thus witnessing that the proposed approaches are complementary.

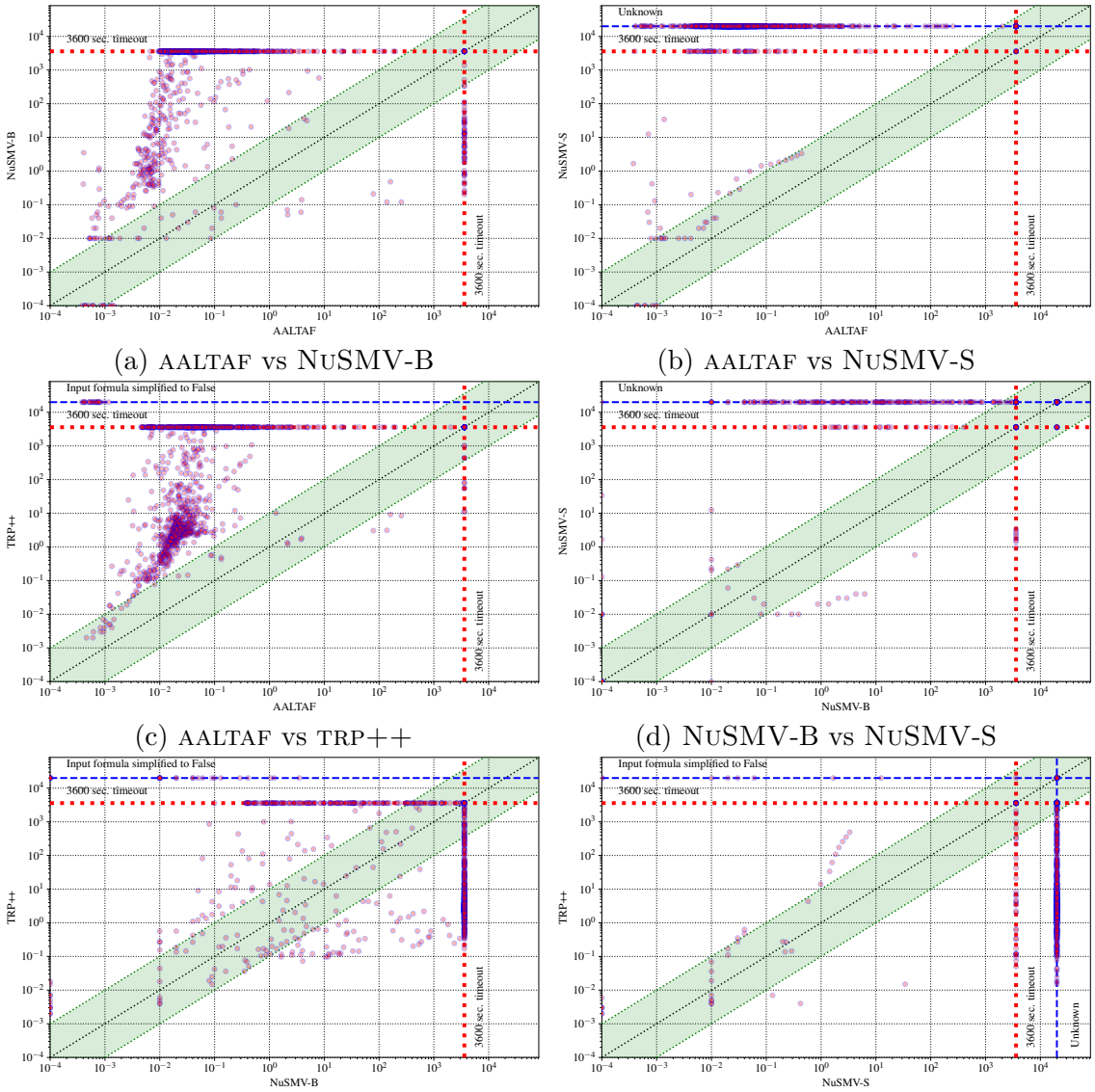
5.3.3 Pairwise Efficiency Comparison

Figure 4 illustrates pairwise comparisons of time efficiency of the considered tools. Here, we also include cases in which a certain answer is *not* returned. Our objective is to provide a visual clue of tests in which only one (or none) of the two tools in the plot was able to extract the UC. Figure 4(a) compares aaltaf with NuSMV-B, Fig. 4(b) compares aaltaf with NuSMV-S, Fig. 4(c) compares aaltaf with trp++, Fig. 4(d) compares NuSMV-B with NuSMV-S, Fig. 4(e) compares NuSMV-B with trp++, and finally Fig. 4(f) compares NuSMV-S with trp++. Figure 4(c), in particular, shows that aaltaf outperforms trp++ in terms of computation speed: most of the points, indeed, are located above the diagonal, thus indicating that aaltaf demands less time than trp++ to return the unsatisfiable cores. The plot also shows that trp++ exceeds the timeout in several cases (points on the red line marked with "3600 sec. timeout"). Furthermore, we remark that trp++ operates a pre-processing phase on the input specification prior to the actual identification of UCs. If it manages to reduce the given set of conjuncts to false at that stage, it stops the computation before returning any UCs and raises an alert. The points lying on the line marked with "Input formula simplified to False" indicate those cases (see Figs. 4(c), (e) and (f)). This simplification occurred 26 times in total.

NuSMV-S was able to conclude that the formula was unsatisfiable and return a UC in 44 cases, whereas it yielded an unknown answer (i.e., it reached $k = 50$ without being able to decide on unsatisfiability) in 1278 cases (see the line labelled with "Unknown" in Figs. 4(b), (d), and (f)). Finally, we report that aaltaf, trp++, NuSMV-B, and NuSMV-S reached the timeout in 84, 562, 990, and 55 cases, respectively.

5.3.4 Best Time Performance per Benchmark Family

Figure 5 focuses again on computation time, contrasting the overall best performance with the benchmark family the input data stem from. In particular, Fig 5(a) shows a pie chart with an overview of the number of tests in which a tool was the fastest, and Fig 5(b) depicts a stacked bar chart in which the results are grouped by benchmark family. Differently from the Sankey diagram in Fig. 2, we determine only one best performer among all tools, without *ex-aequo* leading positions. When solvers take the same time, we associate the best result to the tool that returned the UC that is smaller in cardinality.



(a) AALTAF vs NuSMV-B

(b) AALTAF vs NuSMV-S

(c) AALTAF vs TRP++

(d) NuSMV-B vs NuSMV-S

(e) NuSMV-B vs TRP++

(f) NuSMV-S vs TRP++

Figure 4: Scatter plots comparing search timings for each algorithm pair.

The pie chart in Fig. 5(a) confirms that AALTAF is the most time-efficient tool as evidenced by its 1261 fastest runs, followed by NuSMV-B (85), and NuSMV-S (8). In 23 cases, no tool was able to return an unsatisfiable core. As shown by Fig. 5(b), NuSMV-B turned out to find the UC in minimum time with the *LTL-as-LTLf/rozier/counter/** benchmark families. Figure 5(b) shows that the problems of the *LTL-as-LTLf/schuppan/O2Formula* benchmark family are the most challenging ones for all the implemented techniques. Indeed, 14 out of the 23 problems that were not solved by any tools belong to it. The analysis per benchmark family in Fig. 5(b) confirms the superiority in terms of time efficiency of AALTAF in most of the tests, with the exclusion of the aforementioned *LTL-*

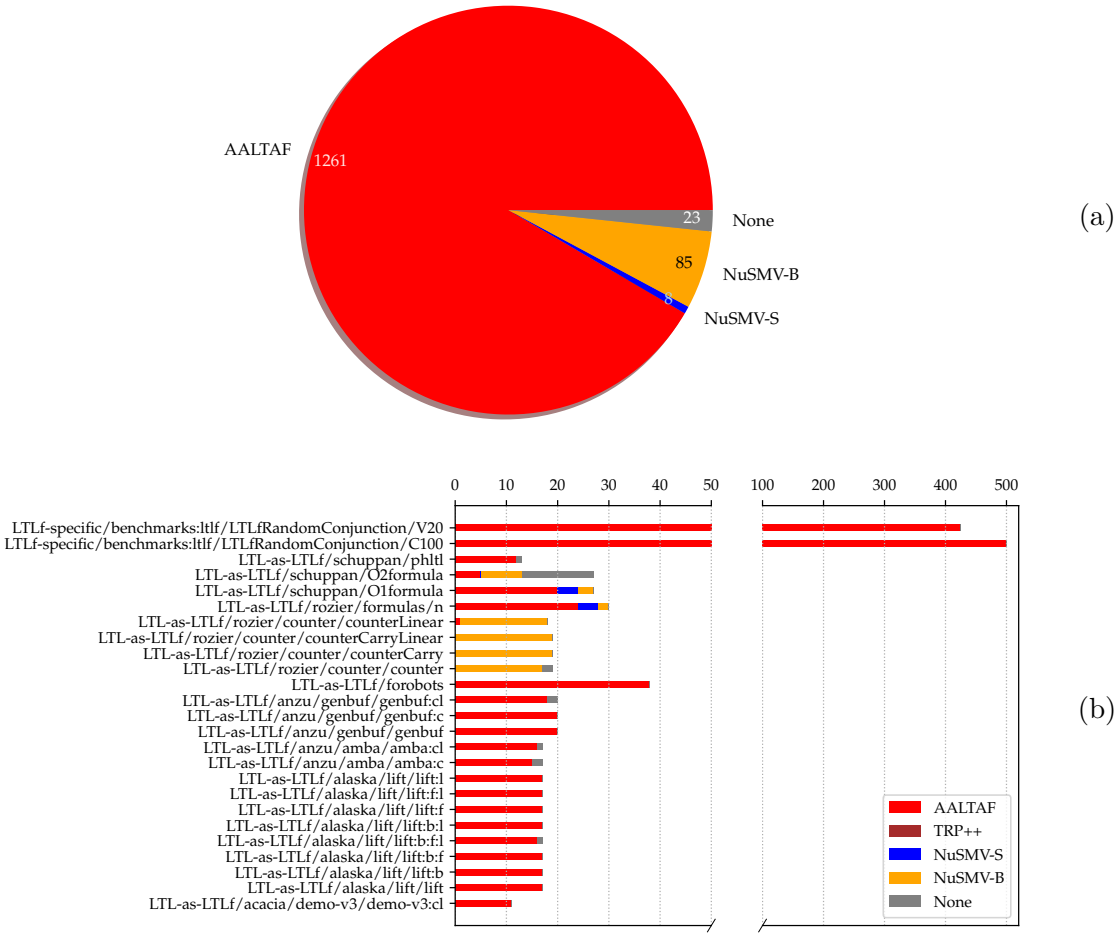


Figure 5: Number of tests in which the solver took the lowest computation time (a) for the entire set of benchmarks, and (b) per benchmark family.

as-LTLf/schuppan/O2Formula family, and the *LTL-as-LTLf/rozier/counter/** benchmarks, with which NuSMV-B is the best performer.

5.3.5 Best Time Performance per Number of Conjuncts

In order to further inspect the correlation between the time performance of the tools and the type of problems solved, we analysed the relationship between the number of conjuncts of the problems and the corresponding computation time. Figure 6 plots the number of conjuncts in Γ (i.e., its cardinality) against the computation time (in seconds) of each of the considered algorithms.

Figure 7 isolates the points stemming from three families in particular: *LTLf-specific/benchmarks/LTLFRandomConjunction/V20* (Fig. 7(a)), *LTLf-specific/benchmarks/LTLFRandomConjunction/C100* (Fig. 7(b)), and *LTL-as-LTLf/schuppan/O1Formula* (Fig. 7(c)). The plots show that a relationship exists between the number of LTL$_f$ clauses and the computation time for all the four tools: the

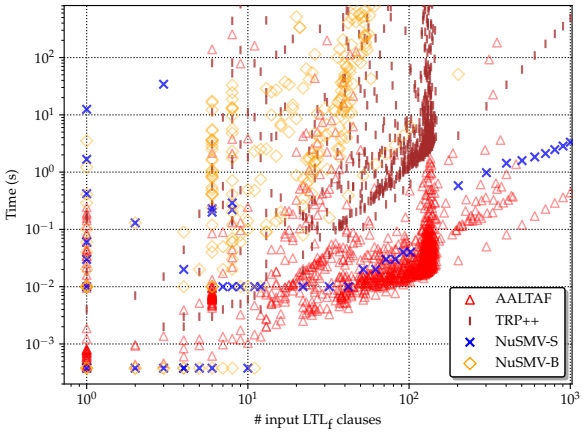


Figure 6: Number of conjuncts in Γ and respective computation time (in seconds) for each algorithm.

required overall time increases when the number of clauses increases. However, the number of clauses is not the only factor affecting the computation time. For instance, for the *LTLf-specific/benchmarks/LTLFRandomConjunction/C100* benchmark family (Fig. 7(b)), the computation time varies independently of the number of conjuncts, which ranges in a short interval (118 to 154 clauses, as per Table 2). Also, we can observe that neither NuSMV-S nor NuSMV-B could return a UC under the imposed experimental resource constraints with this benchmark family, while AALTAF appears to be faster than TRP++, following the general trend. Especially in Fig. 7(c) (and, to a lesser extent, in Fig. 7(a)) we can observe the different rapidity with which curves increase with the number of conjuncts: the steepest slope is associated with NuSMV-B, followed by TRP++ and NuSMV-S. NuSMV-S performs better than TRP++ with smaller sets of conjuncts, though. The most gradual upward trend belongs to the curve of AALTAF.

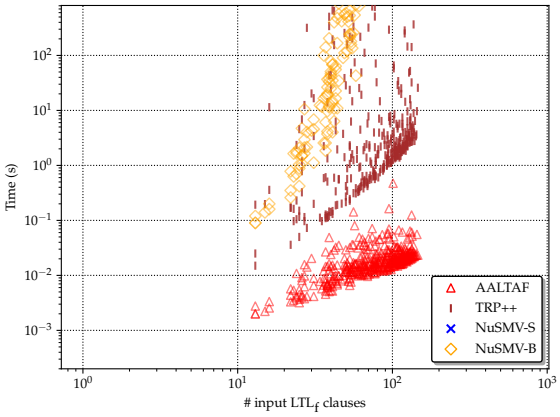
Next, we shift our focus from efficiency to efficacy, i.e., from execution time to the cardinality of the returned UCs.

5.3.6 Extraction of the Smallest UC per Benchmark Family

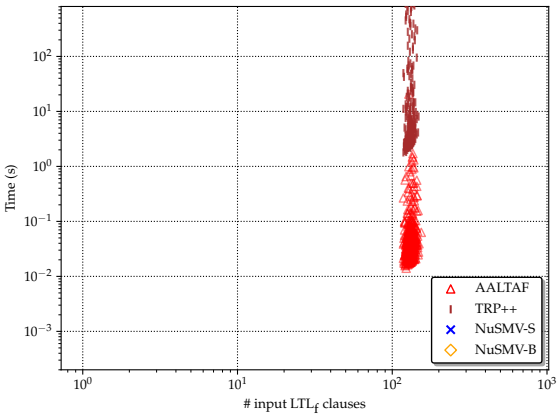
Figure 8 depicts the result of our efficacy analysis in the different benchmarks families. As shown in the pie chart of Fig. 8(a),AALTAF, TRP++, NuSMV-B, and NuSMV-S extract UCs that are the smallest in size[10] in 667, 556, 119 and 12 cases, respectively. As in Section 5.3.4, here we declare only one tool per test as the most effective. When multiple solvers return a UC of the same cardinality, we consider the one that took the lower computation time as prevailing. The overall minimum, maximum, average, and median cardinality of the smallest computed UCs were 1, 74, 6.513, and 4, respectively.

NuSMV-B computes the unsatisfiable core with the smallest size with the majority of test cases in the *LTL-as-LTLf/rozier/counter/** benchmarks. Notice that it is also the

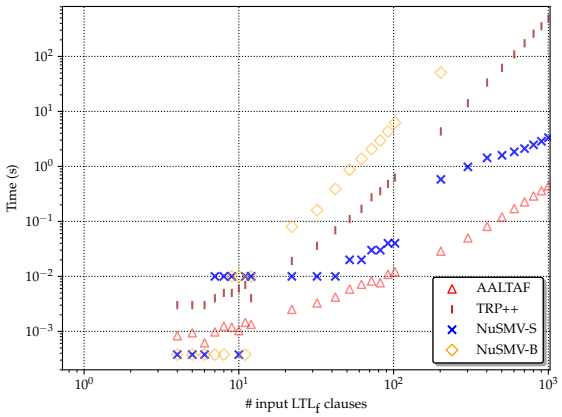
10. By "smallest" we mean the unsatisfiable core of smallest cardinality among the ones computed by the solvers. Notice that the smallest UC does not necessarily correspond to the minimum one, as discussed when presenting the algorithms.



(a)



(b)

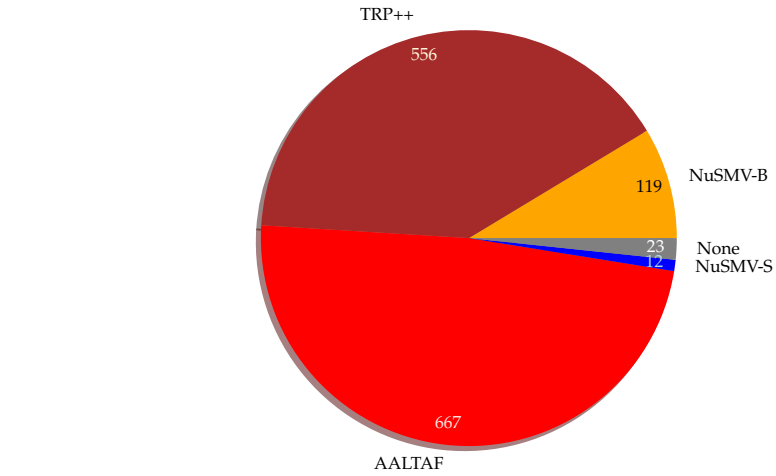


(c)

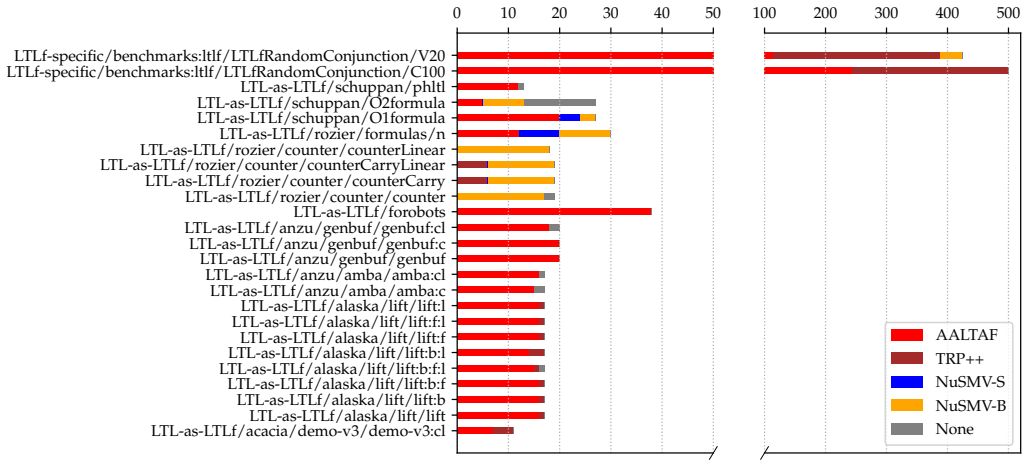
Figure 7: Number of conjuncts in Γ and respective computation time (in seconds) for each algorithm for three categories: (a) *LTLf-specific/benchmarks/LTLFRandomConjunction/V20*, (b) *LTLf-specific/benchmarks/LTLFRandomConjunction/C100*, and (c) *LTL-as-LTLf/schuppan/O1Formula*.

one that performs best most often in terms of search time (see Fig. 5(b)). Furthermore, NuSMV-B is able to obtain the smallest UC with most of the benchmarks within the *LTL-as-LTLf/schuppan/O2formula family*. On all other benchmarks, AALTAF outperforms the other algorithms, and with the *LTLf-specific/** benchmarks, TRP++ is the second best solver to find the smallest UCs after AALTAF.

These results suggest that NuSMV-B could be preferred on benchmarks with fewer propositional variables and larger temporal depth. However, the SAT-based approaches seem to work better on benchmarks with a higher number of propositional variables that are not always directly correlated with one another. In these cases, BDDs may suffer a blow-up in size due to the canonicity of the representation indeed, as BDD dynamic variable ordering could help though to a limited extent (Felt et al., 1993). Notice that none



(a)



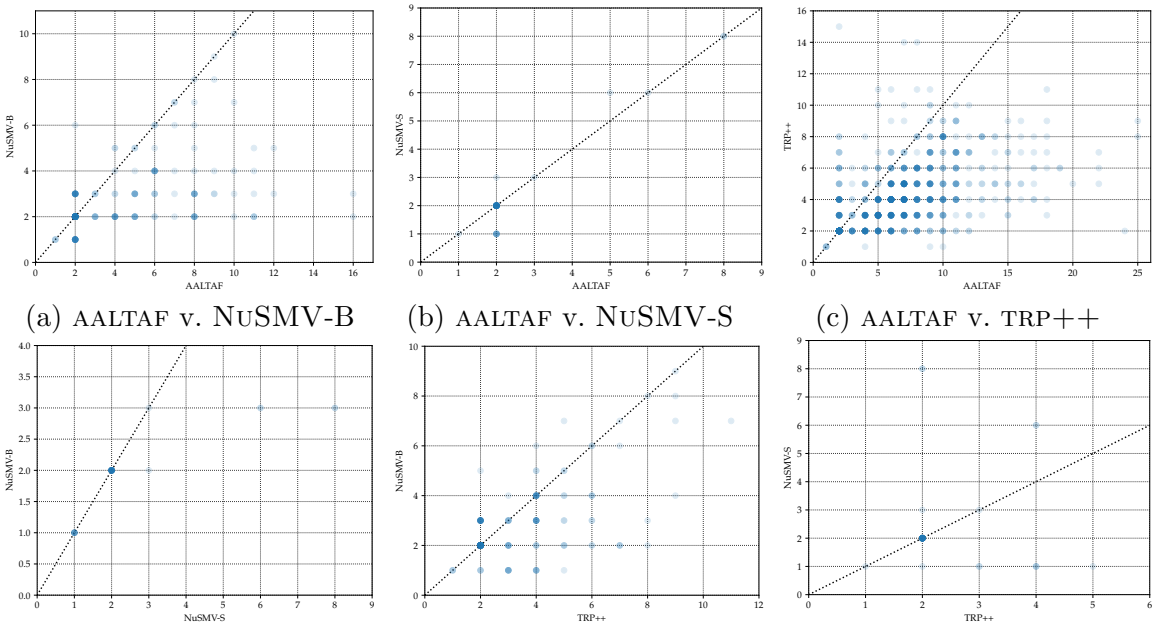
(b)

Figure 8: Number of tests in which the solver returned an unsatisfiable core of the smallest size (a) for the entire set of benchmarks, and (b) per benchmark family.

of the solvers was capable of dealing with most of the big conjunctions of formulas in the *LTL-as-LTLf/schuppan/O2Formula* family (the corresponding tallest stacked bar in Fig. 8 is labelled with "None", indeed).

5.3.7 Pairwise Efficacy Comparison

Figure 9 plots the pairwise comparison between different tools on the subset of the cases where both approaches were able to compute the UC. For instance, Fig. 9(a) compares the cardinality of the UCs returned by AALTAF with the cardinality of the UCs returned by NuSMV-B. The plot shows that the UCs returned by NuSMV-B have a lower cardinality than the ones returned by AALTAF: most of the points are indeed located below the diagonal line. The opacity of the points represents the number of cases for which the two algorithms returned UCs with the specific cardinality corresponding to the point's coordinates in the



(a) AALTAF v. NuSMV-B (b) AALTAF v. NuSMV-S (c) AALTAF v. TRP++

(d) NuSMV-S v. NuSMV-B (e) TRP++ v. NuSMV-B (f) TRP++ v. NuSMV-S

Figure 9: Scatter plots comparing the cardinality of computed UCs for each algorithm pair.

plot. Overall, we can observe that the UCs extracted by TRP++ and NuSMV-B (when these tools manage to return a certain answer) are often lower in size than the UCs returned by AALTAF.

5.3.8 Summary of the Findings

To conclude, we remark that these results (i) demonstrate an overall better performance of AALTAF in terms of time efficiency, (ii) show a tie between AALTAF and TRP++ with respect to the cardinality of the extracted UCs, and (iii) emphasise a complementarity of the proposed approaches depending on the characteristics of the specifications at hand.

Tables 3 and 4 summarise the above findings. We can observe that none of the algorithms outperforms all the others on every benchmark. For example, NuSMV-S and NuSMV-B end up in a timeout and return an unknown answer in considerably many cases, and a number of problems are solved by only one of them. However, NuSMV-B is capable of handling the *LTL-as-LTLf/rozier/counter/** benchmarks well. AALTAF does not always turn out to extract the smallest UC: in a number of cases, TRP++, NuSMV-B and NuSMV-S extract UCs of a lower cardinality, excelling in particular in those cases in which AALTAF ends in a timeout. A deeper investigation of the characteristics that lead to such behaviours paves the path for future research endeavours.

Table 3: Best results as per the cardinality of UCs and wall-clock timings obtained via AALTAF and TRP++. The top achievements with NuSMV-S and NuSMV-B are in Table 4 alongside the benchmarks for which no tools returned a UC.

		AALTAF		TRP++	
Family	Total	min.UC	min.time	min.UC	min.time
Overall	1377	667 (48.44 %)	1261 (91.58 %)	556 (40.38 %)	
LTL-as-LTLf/acacia/demo-v3/demo-v3:cl	11	7 (63.64 %)	11 (100.00 %)	4 (36.36 %)	
LTL-as-LTLf/alaska/lift/lift	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:b	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:b:f	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:b:f:l	17	15 (88.24 %)	16 (94.12 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:b:l	17	14 (82.35 %)	17 (100.00 %)	3 (17.65 %)	
LTL-as-LTLf/alaska/lift/lift:f	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:f:l	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/alaska/lift/lift:l	17	16 (94.12 %)	17 (100.00 %)	1 (5.88 %)	
LTL-as-LTLf/anzu/amba/amba:c	17	15 (88.24 %)	15 (88.24 %)		
LTL-as-LTLf/anzu/amba/amba:cl	17	16 (94.12 %)	16 (94.12 %)		
LTL-as-LTLf/anzu/genbuf/genbuf	20	20 (100.00 %)	20 (100.00 %)		
LTL-as-LTLf/anzu/genbuf/genbuf:c	20	20 (100.00 %)	20 (100.00 %)		
LTL-as-LTLf/anzu/genbuf/genbuf:cl	20	18 (90.00 %)	18 (90.00 %)		
LTL-as-LTLf/forobots	38	38 (100.00 %)	38 (100.00 %)		
LTL-as-LTLf/rozier/counter/counter	19				
LTL-as-LTLf/rozier/counter/counterCarry	19			6 (31.58 %)	
LTL-as-LTLf/rozier/counter/counterCarryLinear	19			6 (31.58 %)	
LTL-as-LTLf/rozier/counter/counterLinear	18		1 (5.56 %)		
LTL-as-LTLf/rozier/formulas/n	30	12 (40.00 %)	24 (80.00 %)		
LTL-as-LTLf/schuppan/O1formula	27	20 (74.07 %)	20 (74.07 %)		
LTL-as-LTLf/schuppan/O2formula	27	5 (18.52 %)	5 (18.52 %)		
LTL-as-LTLf/schuppan/phltl	13	12 (92.31 %)	12 (92.31 %)		
LTLf-specific/.../C100	500	244 (48.80 %)	500 (100.00 %)	256 (51.20 %)	
LTLf-specific/.../V20	425	114 (26.82 %)	425 (100.00 %)	274 (64.47 %)	

6. Related Work

To the best of our knowledge, this is the first research endeavour aimed at extracting unsatisfiable cores for LTL_f. In the following, we review the most relevant literature concerning LTL/LTL_f satisfiability, and LTL SAT-based UC extraction.

The LTL satisfiability problem has been addressed through tableau-based methods (e.g., Janssen, 1999), temporal resolution (e.g., Fisher et al., 2001), and reduction to model checking (e.g., Cimatti et al., 2007; Rozier & Vardi, 2010, 2011). In (Rozier & Vardi, 2010), a reduction of the LTL satisfiability problem to a model checking problem, and a comparison of different model checkers (explicit/symbolic) is carried out, resulting in better performance and quality for symbolic approaches. A thorough comparison of the main tools dealing with the LTL satisfiability problem is reported in (Schuppan & Darmawan, 2011). The paper considers also tableau and temporal resolution based solvers, revealing a complementary behaviour between some of the considered solvers.

The problem of checking the satisfiability of LTL_f properties has been the subject of several works (Fionda & Greco, 2018; Li et al., 2020, 2014). Li et al. (2014) leverage the

Table 4: Best results as per the cardinality of UCs and wall-clock timings obtained via NuSMV-S and NuSMV-B, alongside the benchmarks for which no tools returned a UC (marked with "None"). The top achievements with aaltaf and trp++ are in Table 3.

		NuSMV-S		NuSMV-B		
Family	Total	min.UC	min.time	min.UC	min.time	None
Overall	1377	12 (0.87%)	8 (0.58%)	119 (8.64%)	85 (6.17%)	23 (1.67%)
LTL-as-LTLf/acacia/demo-v3/demo-v3:cl	11					
LTL-as-LTLf/alaska/lift/lift	17					
LTL-as-LTLf/alaska/lift/lift:b	17					
LTL-as-LTLf/alaska/lift/lift:b:f	17					
LTL-as-LTLf/alaska/lift/lift:b:f:l	17					1 (5.88%)
LTL-as-LTLf/alaska/lift/lift:b:l	17					
LTL-as-LTLf/alaska/lift/lift:f	17					
LTL-as-LTLf/alaska/lift/lift:f:l	17					
LTL-as-LTLf/alaska/lift/lift:l	17					
LTL-as-LTLf/anzu/amba/amba:c	17					2 (11.76%)
LTL-as-LTLf/anzu/amba/amba:cl	17					1 (5.88%)
LTL-as-LTLf/anzu/genbuf/genbuf	20					
LTL-as-LTLf/anzu/genbuf/genbuf:c	20					
LTL-as-LTLf/anzu/genbuf/genbuf:cl	20					2 (10.00%)
LTL-as-LTLf/forobots	38					
LTL-as-LTLf/rozier/counter/counter	19			17 (89.47%)	17 (89.47%)	2 (10.53%)
LTL-as-LTLf/rozier/counter/counterCarry	19			13 (68.42%)	19 (100.00%)	
LTL-as-LTLf/rozier/counter/counterCarryLinear	19			13 (68.42%)	19 (100.00%)	
LTL-as-LTLf/rozier/counter/counterLinear	18			18 (100.00%)	17 (94.44%)	
LTL-as-LTLf/rozier/formulas/n	30	8 (26.67%)	4 (13.33%)	10 (33.33%)	2 (6.67%)	
LTL-as-LTLf/schuppan/O1formula	27	4 (14.81%)	4 (14.81%)	3 (11.11%)	3 (11.11%)	
LTL-as-LTLf/schuppan/O2formula	27			8 (29.63%)	8 (29.63%)	14 (51.85%)
LTL-as-LTLf/schuppan/phltl	13					1 (7.69%)
LTLf-specific/.../C100	500					
LTLf-specific/.../V20	425			37 (8.71%)		

finite semantics of traces for introducing a propositional SAT based algorithm for the LTL$_f$ satisfiability problem together with some heuristics to guide the search. The approach is implemented in the aalta-finite tool, which is shown to outperform other existing approaches based on the reduction to the LTL satisfiability problem. An extension of that work is presented in (Li et al., 2020). The new approach leverages a transition system (TS) for the input LTL$_f$ formula, thereby reducing satisfiability checking to a SAT-based path-search problem over this TS. Implemented in aalta-finite, it is shown to provide the best results in checking unsatisfiable formulas and comparable results for satisfiable ones. Fionda and Greco (2018) investigate the complexity of some fragments of LTL$_f$, and present a SAT-based algorithm that outperforms the aalta-finite version in (Li et al., 2014). Our algorithm NA4 is based upon the work of Li et al. (2020) as a state-of-the-art tool for checking the satisfiability of LTL$_f$ properties.

The UC extraction for LTL has also been the subject of several studies (Awad et al., 2011; Goré et al., 2013; Narizzano et al., 2018; Schuppan, 2016). Goré et al. (2013) present a BDD based approach that leverages a method to determine minimal UCs for SAT (Huang,

2005) to find minimal UCs in LTL. In the work of Awad et al. (2011), UCs are extracted by leveraging a tableau-based solver to obtain an initial subset of unsatisfiable LTL formulas and then applying a deletion-based minimisation to the subset. The approach, implemented in PROCMINE is part of a tool for the synthesis of business process templates. Schuppan (2016) propose a technique to extract fine-grained UCs by constructing and optimising resolution graphs for temporal resolution. Finally, Narizzano et al. (2018) presents a SAT-based encoding suitable for the unsatisfiable core extraction of LTL-based property specification patterns (Dwyer et al., 1999) extended with inequality statements on Boolean and numeric variables. Algorithm NA3, presented here, is built upon the work of Schuppan (2016) to compute UCs using temporal resolution.

In the context of process mining, Corea and Delfmann (2019); Di Ciccio et al. (2017) identify inconsistencies for specific LTL_f-based constraints contained in the Declare modelling language (van der Aalst et al., 2009). They rely on automata language and language inclusion techniques to identify the inconsistencies, and are specific to the precise structure of Declare. Thus, they cannot be generalised to address generic LTL_f-based specifications.

Finally, we remark that works on propositional UC extraction (e.g., Goldberg and Novikov 2003; Huang 2005; Marques-Silva and Janota 2014) could be used to improve the quality of the computed cores. We leave this investigation for future developments.

7. Conclusions and Future Work

In this paper, we have addressed the problem of LTL_f unsatisfiable core extraction, presenting four algorithms based on different state-of-the-art techniques for LTL and LTL_f satisfiability checking. We have implemented each of them based on existing tools, and we have carried out an experimental evaluation on a set of reference benchmarks for unsatisfiable temporal formulas. The results show a consistent output when terminating without reaching a resource limit and the feasibility of the proposed algorithms. The extensive evaluation evidences an overall better performance of AALTAF (and thus of algorithm NA4), both in terms of time efficiency and cardinality of the extracted UCs. Nonetheless, none of the algorithms outperforms all the others on every benchmark and in a number of cases, TRP++, NuSMV-B and NuSMV-S extract UCs of a lower cardinality, excelling in particular in those cases in which AALTAF ends in a timeout. Furthermore, the evaluation shows that in 28 cases, no tool was able to return an unsatisfiable core, and that the problems of the *LTL-as-LTLf/schuppan/O2Formula* benchmark family are the most challenging ones for all the implemented techniques. Indeed, 14 out of the 28 problems that were not solved by any tools belong to this benchmark family.

These results show the adequacy of exploring different strategies and algorithmic solutions for this problem, and – at the same time – provide a first extensive baseline for future algorithms for the extraction of LTL_f unsatisfiable cores.

For future work, we envisage the following research endeavours. Firstly, addressing the problem of extracting minimal UCs is in our plans. It is also our objective to extend the approach to other $\text{LTL}/\text{LTL}_\text{f}$ algorithms based on k-liveness (Claessen & Sörensson, 2012), liveness to safety (Biere et al., 2002), or tableau constructions (Geatti et al., 2021), Furthermore, we intend to extend the problems set with benchmarks from other domains (including AI planning, requirements engineering, and process management), also including

a study on effectiveness and efficacy in presence of specifications the satisfiability of which is not certain, as is rather common in these cases. Moreover, we want to correlate structural information (e.g., $\mathcal{AP}$ cardinality, temporal depth, number of operators) with solving algorithms, and aim to investigate the extension to the infinite state case exploiting SMT techniques (Barrett et al., 2009; Daniel et al., 2016).

Acknowledgments

M. Roveri and C. Ghidini are partly supported by the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU. M. Roveri is also partially supported by the project MUR PRIN 2020 - RIPER - Resilient AI-Based Self-Programming and Strategic Reasoning - CUP E63C22000400001, and by the European Union under Horizon Europe Programme - Grant Agreement 101070537 — CrossCon. The work of C. Di Ciccio and C. Ghidini received funding from the Italian Ministry of University and Research under the PRIN programme, grant B87G22000450001 (PINPOINT - exPlaInable kNowledge-aware PrOcess INTelligence).

References

Awad, A., Goré, R., Thomson, J., & Weidlich, M. (2011). An Iterative Approach for Business Process Template Synthesis from Compliance Rules. In H. Mouratidis & C. Rolland (Eds.), *Advanced Information Systems Engineering - 23rd International Conference, CAiSE 2011, London, UK, June 20-24, 2011. Proceedings* (Vol. 6741, pp. 406–421). Springer. Retrieved from https://doi.org/10.1007/978-3-642-21640-4_31 doi: 10.1007/978-3-642-21640-4_31

Barrett, C. W., Sebastiani, R., Seshia, S. A., & Tinelli, C. (2009). Satisfiability Modulo Theories. In A. Biere, M. Heule, H. van Maaren, & T. Walsh (Eds.), *Handbook of Satisfiability* (Vol. 185, pp. 825–885). IOS Press. Retrieved from https://doi.org/10.3233/978-1-58603-929-5-825 doi: 10.3233/978-1-58603-929-5-825

Bauer, A., Leucker, M., & Schallhart, C. (2010). Comparing LTL Semantics for Runtime Verification. *Journal of Logic and Computation*, *20*(3), 651-674. Retrieved from https://doi.org/10.1093/logcom/exn075 doi: 10.1093/logcom/exn075

Biere, A., Artho, C., & Schuppan, V. (2002). Liveness Checking as Safety Checking. *Electronic Notes in Theoretical Computer Science*, *66*(2), 160–177. doi: 10.1016/S1571-0661(04)80410-9

Biere, A., Cimatti, A., Clarke, E. M., Strichman, O., & Zhu, Y. (2003). Bounded Model Checking. *Advances in Computers*, *58*, 117–148. Retrieved from https://doi.org/10.1016/S0065-2458(03)58003-2 doi: 10.1016/S0065-2458(03)58003-2

Biere, A., Heljanko, K., Junttila, T. A., Latvala, T., & Schuppan, V. (2006). Linear Encodings of Bounded LTL Model Checking. *Logical Methods in Computer Science*, *2*(5). Retrieved from https://doi.org/10.2168/LMCS-2(5:5)2006 doi: 10.2168/LMCS-2(5:5)2006

Bradley, A. (2011). SAT-Based Model Checking without Unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation* (Vol. 6538, p. 70-87). Springer. Retrieved from https://doi.org/10.1007/978-3-642-18275-4_7

doi: 10.1007/978-3-642-18275-4_7

Bryant, R. E. (1992). Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Computing Surveys*, *24*(3), 293–318. Retrieved from https://doi.org/10.1145/136035.136043 doi: 10.1145/136035.136043

Calvanese, D., De Giacomo, G., & Vardi, M. Y. (2002). Reasoning about Actions and Planning in LTL Action Theories. In D. Fensel, F. Giunchiglia, D. L. McGuinness, & M. Williams (Eds.), *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002* (pp. 593–602). Morgan Kaufmann.

Camacho, A., Baier, J. A., Muise, C. J., & McIlraith, S. A. (2018). Finite LTL Synthesis as Planning. In M. de Weerdt, S. Koenig, G. Röger, & M. T. J. Spaan (Eds.), *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018, Delft, The Netherlands, June 24-29, 2018* (pp. 29–38). AAAI Press. Retrieved from https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17790

Camacho, A., & McIlraith, S. A. (2019). Strong Fully Observable Non-Deterministic Planning with LTL and LTLf Goals. In S. Kraus (Ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019* (pp. 5523–5531). ijcai.org. Retrieved from https://doi.org/10.24963/ijcai.2019/767 doi: 10.24963/ijcai.2019/767

Cecconi, A., Di Ciccio, C., De Giacomo, G., & Mendling, J. (2018). Interestingness of Traces in Declarative Process Mining: The Janus LTLp$_f$ Approach. In M. Weske, M. Montali, I. Weber, & J. vom Brocke (Eds.), *Business Process Management - 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9-14, 2018, Proceedings* (Vol. 11080, pp. 121–138). Springer. Retrieved from https://doi.org/10.1007/978-3-319-98648-7_8 doi: 10.1007/978-3-319-98648-7_8

Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., ... Tacchella, A. (2002). NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In E. Brinksma & K. G. Larsen (Eds.), *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings* (Vol. 2404, pp. 359–364). Springer. Retrieved from https://doi.org/10.1007/3-540-45657-0_29 doi: 10.1007/3-540-45657-0_29

Cimatti, A., Roveri, M., Schuppan, V., & Tonetta, S. (2007). Boolean Abstraction for Temporal Logic Satisfiability. In W. Damm & H. Hermanns (Eds.), *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings* (Vol. 4590, pp. 532–546). Springer. Retrieved from https://doi.org/10.1007/978-3-540-73368-3_53 doi: 10.1007/978-3-540-73368-3_53

Cimatti, A., Roveri, M., & Sheridan, D. (2004). Bounded Verification of Past LTL. In A. J. Hu & A. K. Martin (Eds.), *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings* (Vol. 3312, pp. 245–259). Springer. Retrieved from https://doi.org/10.1007/978-3-540-30494-4_18 doi: 10.1007/978-3-540-30494-4_18

Claessen, K., & Sörensson, N. (2012). A Liveness Checking Algorithm that Counts. In G. Cabodi & S. Singh (Eds.), *Formal Methods in Computer Aided Design* (p. 52-59). IEEE.

Clarke, E. M., Grumberg, O., & Hamaguchi, K. (1997). Another Look at LTL Model Checking. *Formal Methods Syst. Des.*, *10*(1), 47–71. Retrieved from https://doi.org/10.1023/A:1008615614281 doi: 10.1023/A:1008615614281

Corea, C., & Delfmann, P. (2019). Quasi-Inconsistency in Declarative Process Models. In T. T. Hildebrandt, B. F. van Dongen, M. Röglinger, & J. Mendling (Eds.), *Business Process Management Forum - BPM Forum 2019, Vienna, Austria, September 1-6, 2019, Proceedings* (Vol. 360, pp. 20–35). Springer. Retrieved from https://doi.org/10.1007/978-3-030-26643-1_2 doi: 10.1007/978-3-030-26643-1_2

Daniel, J., Cimatti, A., Griggio, A., Tonetta, S., & Mover, S. (2016). Infinite-State Liveness-to-Safety via Implicit Abstraction and Well-Founded Relations. In S. Chaudhuri & A. Farzan (Eds.), *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I* (Vol. 9779, pp. 271–291). Springer. Retrieved from https://doi.org/10.1007/978-3-319-41528-4_15 doi: 10.1007/978-3-319-41528-4_15

De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F. M., & Montali, M. (2014). Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces. In S. W. Sadiq, P. Soffer, & H. Völzer (Eds.), *Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings* (Vol. 8659, pp. 1–17). Springer. Retrieved from https://doi.org/10.1007/978-3-319-10172-9_1 doi: 10.1007/978-3-319-10172-9_1

De Giacomo, G., De Masellis, R., Maggi, F. M., & Montali, M. (2020). Monitoring Constraints and Metaconstraints with Temporal Logics on Finite Traces. *CoRR*, *abs/2004.01859*. Retrieved from https://arxiv.org/abs/2004.01859

De Giacomo, G., De Masellis, R., & Montali, M. (2014). Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness. In C. E. Brodley & P. Stone (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada* (pp. 1027–1033). AAAI Press. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8575

De Giacomo, G., & Vardi, M. Y. (2013). Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In F. Rossi (Ed.), *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013* (pp. 854–860). IJCAI/AAAI. Retrieved from http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997

Di Ciccio, C., Maggi, F. M., Montali, M., & Mendling, J. (2017). Resolving Inconsistencies and Redundancies in Declarative Process Models. *Information Systems*, *64*, 425–446. Retrieved from https://doi.org/10.1016/j.is.2016.09.005 doi: 10.1016/j.is.2016.09.005

Di Ciccio, C., & Montali, M. (2022). Declarative Process Specifications: Reasoning, Discovery, Monitoring. In W. M. P. van der Aalst & J. Carmona (Eds.), *Process mining handbook* (Vol. 448, pp. 108–152). Springer. Retrieved from https://doi.org/10.1007/978-3-031-08848-3_4 (Open access) doi: 10.1007/978-3-031-08848-3_4

Dwyer, M. B., Avrunin, G. S., & Corbett, J. C. (1999). Patterns in Property Specifications for Finite-State Verification. In B. W. Boehm, D. Garlan, & J. Kramer (Eds.), *Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, May 16-22, 1999* (pp. 411–420). ACM. Retrieved from

https://doi.org/10.1145/302405.302672 doi: 10.1145/302405.302672

Felt, E., York, G., Brayton, R. K., & Sangiovanni-Vincentelli, A. L. (1993). Dynamic Variable Reordering for BDD Minimization. In *European Design Automation Conference* (pp. 130–135). IEEE Computer Society.

Fionda, V., & Greco, G. (2018). LTL on Finite and Process Traces: Complexity Results and a Practical Reasoner. *Journal of Artificial Intelligence Research*, *63*, 557–623. Retrieved from https://doi.org/10.1613/jair.1.11256 doi: 10.1613/jair.1.11256

Fisher, M. (1991). A Resolution Method for Temporal Logic. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991* (pp. 99–104). Morgan Kaufmann. Retrieved from http://ijcai.org/Proceedings/91-1/Papers/017.pdf

Fisher, M., Dixon, C., & Peim, M. (2001). Clausal Temporal Resolution. *ACM Transactions on Computational Logic*, *2*(1), 12–56. Retrieved from https://doi.org/10.1145/371282.371311 doi: 10.1145/371282.371311

Fuxman, A., Liu, L., Mylopoulos, J., Roveri, M., & Traverso, P. (2004). Specifying and Analyzing Early Requirements in Tropos. *IEEE International Conference on Requirements Engineering*, *9*(2), 132–150.

Gabbay, D. M. (1987). The Declarative Past and Imperative Future: Executable Temporal Logic for Interactive Systems. In B. Banieqbal, H. Barringer, & A. Pnueli (Eds.), *Temporal Logic in Specification, Altrincham, UK, April 8-10, 1987, Proceedings* (Vol. 398, pp. 409–448). Springer. Retrieved from https://doi.org/10.1007/3-540-51803-7_36 doi: 10.1007/3-540-51803-7_36

Geatti, L., Gigante, N., Montanari, A., & Reynolds, M. (2021). One-pass and Tree-shaped Tableau Systems for TPTL and $\mathrm{TPTL_b}$+Past. *Information and Computation*, *278*, 104599. Retrieved from https://doi.org/10.1016/j.ic.2020.104599 doi: 10.1016/j.ic.2020.104599

Goldberg, E. I., & Novikov, Y. (2003). Verification of Proofs of Unsatisfiability for CNF Formulas. In *2003 Design, Automation and Test in Europe Conference and Exposition (DATE 2003), 3-7 March 2003, Munich, Germany* (pp. 10886–10891). IEEE Computer Society. Retrieved from http://doi.ieeecomputersociety.org/10.1109/DATE.2003.10008 doi: 10.1109/DATE.2003.10008

Goré, R., Huang, J., Sergeant, T., & Thomson, J. (2013). *Finding Minimal Unsatisfiable Subsets in Linear Temporal Logic using BDDs.* https://www.timsergeant.com/files/pltlmup/gore_huang_sergeant_thomson_mus_pltl.pdf. (Accessed: 30-08-2021)

Huang, J. (2005). MUP: a Minimal Unsatisfiability Prover. In T. Tang (Ed.), *Proceedings of the 2005 Conference on Asia South Pacific Design Automation, ASP-DAC 2005, Shanghai, China, January 18-21, 2005* (pp. 432–437). ACM Press. Retrieved from https://doi.org/10.1145/1120725.1120907 doi: 10.1145/1120725.1120907

Hustadt, U., & Konev, B. (2003). TRP++2.0: A Temporal Resolution Prover. In F. Baader (Ed.), *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings* (Vol. 2741, pp. 274–278). Springer. Retrieved from https://doi.org/10.1007/978-3-540-45085-6_21 doi: 10.1007/978-3-540-45085-6_21

Janssen, G. (1999). *Logics for Digital Circuit Verification : Theory, Algorithms, and*

Applications (Doctoral dissertation, Electrical Engineering). doi: 10.6100/IR520460

Laroussinie, F., Markey, N., & Schnoebelen, P. (2002). Temporal Logic with Forgettable Past. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings* (pp. 383–392). IEEE Computer Society. Retrieved from https://doi.org/10.1109/LICS.2002.1029846 doi: 10.1109/LICS.2002.1029846

Li, J., Pu, G., Zhang, Y., Vardi, M. Y., & Rozier, K. Y. (2020). SAT-based Explicit LTLf Satisfiability Checking. *Artificial Intelligence*, *289*, 103369. Retrieved from https://doi.org/10.1016/j.artint.2020.103369 doi: 10.1016/j.artint.2020.103369

Li, J., Zhang, L., Pu, G., Vardi, M. Y., & He, J. (2014). LTLf Satisfiability Checking. In T. Schaub, G. Friedrich, & B. O'Sullivan (Eds.), *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)* (Vol. 263, pp. 513–518). IOS Press. Retrieved from https://doi.org/10.3233/978-1-61499-419-0-513 doi: 10.3233/978-1-61499-419-0-513

Li, J., Zhu, S., Pu, G., Zhang, L., & Vardi, M. Y. (2019). SAT-based explicit LTL reasoning and its application to satisfiability checking. *Formal Methods in System Design*, *54*(2), 164–190. Retrieved from https://doi.org/10.1007/s10703-018-00326-5 doi: 10.1007/s10703-018-00326-5

Marques-Silva, J., & Janota, M. (2014). Computing Minimal Sets on Propositional Formulae I: Problems & Reductions. *CoRR*, *abs/1402.3011*. Retrieved from http://arxiv.org/abs/1402.3011

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., & Storari, S. (2010). Declarative Specification and Verification of Service Choreographiess. *ACM Transactions on the Web*, *4*(1), 3:1–3:62. Retrieved from https://doi.org/10.1145/1658373.1658376 doi: 10.1145/1658373.1658376

Narizzano, M., Pulina, L., Tacchella, A., & Vuotto, S. (2018). Consistency of Property Specification Patterns with Boolean and Constrained Numerical Signals. In A. Dutle, C. A. Muñoz, & A. Narkawicz (Eds.), *NASA Formal Methods - 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings* (Vol. 10811, pp. 383–398). Springer. Retrieved from https://doi.org/10.1007/978-3-319-77935-5_26 doi: 10.1007/978-3-319-77935-5_26

Pnueli, A. (1977). The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977* (pp. 46–57). IEEE Computer Society. Retrieved from https://doi.org/10.1109/SFCS.1977.32 doi: 10.1109/SFCS.1977.32

Räim, M., Di Ciccio, C., Maggi, F. M., Mecella, M., & Mendling, J. (2014). Log-Based Understanding of Business Processes through Temporal Logic Query Checking. In R. Meersman et al. (Eds.), *On the Move to Meaningful Internet Systems: OTM 2014 Conferences - Confederated International Conferences: CoopIS, and ODBASE 2014, Amantea, Italy, October 27-31, 2014, Proceedings* (Vol. 8841, pp. 75–92). Springer. Retrieved from https://doi.org/10.1007/978-3-662-45563-0_5 doi: 10.1007/978-3-662-45563-0_5

Rozier, K. Y., & Vardi, M. Y. (2010). LTL Satisfiability Checking. *International Journal of Software Tools for Technology Transfer*, *12*(2), 123–137. Retrieved from https://

doi.org/10.1007/s10009-010-0140-3 doi: 10.1007/s10009-010-0140-3

Rozier, K. Y., & Vardi, M. Y. (2011). A Multi-encoding Approach for LTL Symbolic Satisfiability Checking. In M. J. Butler & W. Schulte (Eds.), *FM 2011: Formal Methods - 17th International Symposium on Formal Methods, Limerick, Ireland, June 20-24, 2011. Proceedings* (Vol. 6664, pp. 417–431). Springer. Retrieved from https://doi.org/10.1007/978-3-642-21437-0_31 doi: 10.1007/978-3-642-21437-0_31

Schuppan, V. (2012). Towards a Notion of Unsatisfiable and Unrealizable Cores for LTL. *Science of Computer Programming*, *77*(7-8), 908–939. Retrieved from https://doi.org/10.1016/j.scico.2010.11.004 doi: 10.1016/j.scico.2010.11.004

Schuppan, V. (2016). Extracting Unsatisfiable Cores for LTL via Temporal Resolution. *Acta Informatica*, *53*(3), 247–299. Retrieved from https://doi.org/10.1007/s00236-015-0242-1 doi: 10.1007/s00236-015-0242-1

Schuppan, V. (2018). Enhanced Unsatisfiable Cores for QBF: Weakening Universal to Existential Quantifiers. In L. H. Tsoukalas, É. Grégoire, & M. Alamaniotis (Eds.), *IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, 5-7 November 2018, Volos, Greece* (pp. 81–89). IEEE. Retrieved from https://doi.org/10.1109/ICTAI.2018.00023 doi: 10.1109/ICTAI.2018.00023

Schuppan, V., & Darmawan, L. (2011). Evaluating LTL Satisfiability Solvers. In T. Bultan & P. Hsiung (Eds.), *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings* (Vol. 6996, pp. 397–413). Springer. Retrieved from https://doi.org/10.1007/978-3-642-24372-1_28 doi: 10.1007/978-3-642-24372-1_28

Sohrabi, S., Baier, J. A., & McIlraith, S. A. (2011). Preferred Explanations: Theory and Generation via Planning. In W. Burgard & D. Roth (Eds.), *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011.* AAAI Press. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3568

van der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative Workflows: Balancing between Flexibility and Support. *Computer Science - Research and Development*, *23*(2), 99–113. Retrieved from https://doi.org/10.1007/s00450-009-0057-9 doi: 10.1007/S00450-009-0057-9

van Lamsweerde, A., & Letier, E. (2000). Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, *26*(10), 978–1005. Retrieved from https://doi.org/10.1109/32.879820 doi: 10.1109/32.879820