Computing Unsatisfiable Cores for LTL_f Specifications[*]

(Discussion/Short Paper)

Marco Roveri[1,*], Claudio Di Ciccio[2], Chiara Di Francescomarino[3] and Chiara Ghidini[3]

[1]*University of Trento, Via Sommarive 9, Trento, Italy*

[2]*Sapienza University of Rome, Viale Regina Elena 295, 00161 Rome, Italy*

[3]*FBK IRST, Via Sommarive 18, Trento, Italy*

1. Introduction

A growing body of literature evidences the adoption of linear-time temporal logic on finite traces (LTL$_f$) [2] to produce systems specifications [3]. Its widespread use spans across several application domains, including business process management (BPM) for declarative process modeling, run-time monitoring and verification, and AI planning.

When it comes to verification techniques and tool support for LTL$_f$, several studies approach the LTL$_f$ satisfiability problem via reduction to LTL [4] satisfiability on infinite traces [3], or via specific propositional satisfiability approaches [5, 6]. However, hardly any efforts have been devoted thus far to the identification of the formulas that lead to unsatisfiability in LTL$_f$ specifications, with the consequence that very little support has been offered for modelers and system designers to single out the causes of possible inconsistencies.

In [1] 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 leads them to a consequent debugging. Notice that determining a reason for unsatisfiability without automated support may reveal 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 [7]. On the theoretical side, we remark that dealing with the extraction of UCs in LTL$_f$ specifications is far from trivial. Indeed, there is neither a default pathway to move from the support provided for LTL to the one that has to be provided for LTL$_f$ nor a default algorithm upon which this transition could be based. Concerning the *pathway*, there are two clear alternatives to address this problem: the first one extends techniques for the extraction of UCs in LTL to the case of LTL$_f$; the second

✉ marco.roveri@unitn.it (M. Roveri); claudio.diciccio@uniroma1.it (C. Di Ciccio); dfmchiara@fbk.eu
(C. Di Francescomarino); ghidini@fbk.eu (C. Ghidini)

🆔 0000-0001-9483-3940 (M. Roveri); 0000-0001-5570-0475 (C. Di Ciccio); 0000-0002-0264-9394
(C. Di Francescomarino); 0000-0003-1563-4965 (C. Ghidini)

one exploits algorithms that directly compute satisfiability in LTL$_f$ to provide support for the extraction of UCs. Concerning the specific *algorithms* from which to start, the two approaches present different scenarios. In the first pathway, it is easy to observe that several techniques for the extraction of UCs in LTL exist, and could be extended to the case of LTL$_f$. Since recent works show that a single universal best algorithm does not exist and often the systems exhibit behaviors that complement each other [8, 6], choosing a single algorithm from which to start is less than obvious. In the second pathway, instead, the number of works on satisfiability in LTL$_f$ is still rather limited.

We explore both the above pathways in [1]. For the LTL pathway, in particular, we consider algorithms belonging to two reference approaches: one based on model-checking, and the other one based on theorem proving. For the LTL$_f$ pathway, we consider a reference state-of-the-art specific reduction to propositional satisfiability. 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 LTL$_f$ specifications. Our comparative evaluation shows a complementary behavior of the different algorithms.

Our contribution consists of the following:

1. Four algorithms that allow for the computation of an unsatisfiable core through the adaptation of the main reference state-of-the-art approaches for LTL and LTL$_f$ satisfiability checking. For the LTL pathway, we consider two satisfiability checking algorithms: one based on Binary Decision Diagrams (BDDs) [9], and the other based on propositional satisfiability [10] alongside a theorem proving algorithm based on temporal resolution [11]. For the native LTL$_f$ pathway, we consider the reference work in [6] based on explicit search and propositional satisfiability. Notice that the techniques based on propositional satisfiability (that is, based on [10] and [6]) aim at extracting a UC, which may not necessarily be the minimum one. The BDD and temporal-resolution based algorithms already allow for the extraction of a minimum unsatisfiable core.

2. An implementation of the proposed four algorithms. Three implementations extend existing tools for the corresponding original algorithms; the implementation of the algorithm based on temporal resolution, instead, resorts to a pre-processing 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 [6], restricted to the unsatisfiable ones. The results show an overall better time efficiency of the algorithm based on the native LTL$_f$ pathway [6]. 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 exhibit a complementarity of the proposed approaches 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 results and vice-versa.

Since popular usages of LTL$_f$ leverage past temporal operators (see e.g., the DECLARE language [12]), we also provide a way to handle LTL$_f$ with past temporal operators. This results in the same expressive power as that of pure future version, though allowing for exponentially more succinct specifications [13] and more natural encodings of LTL$_f$ based modeling languages that make use of these operators. This objective is pursued by leveraging algorithms already supporting LTL with past temporal operators, or through a reduction to LTL$_f$ with only future temporal operators to use existing approaches for LTL$_f$ satisfiability checking.
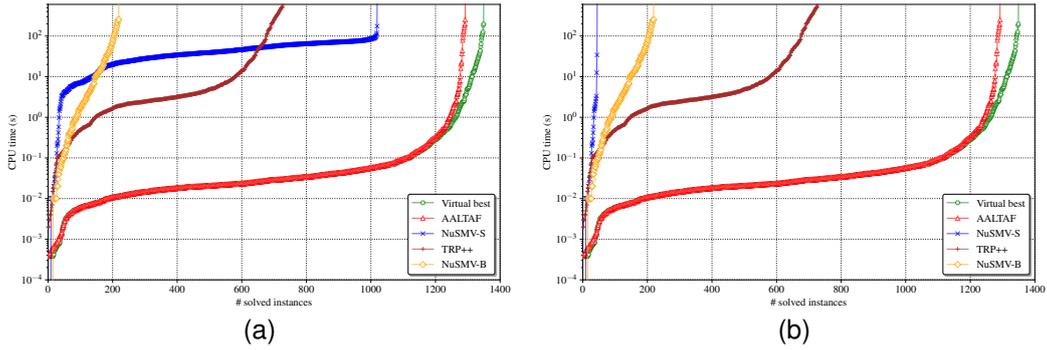
Figure 1: Cactus plots for the experimental evaluation.

2. Experimental Evaluation

The reader can refer to [1] for the complete description of the proposed algorithms and their experimental evaluation. Here we report a sketch of the main findings.

Implementation of the Algorithms. We implemented the algorithms based on BDD [9] and propositional satisfiability [10] as extensions of the NUSMV model checker [14] exploiting the built-in support for past temporal operators. In particular, we enhanced (i) the BDD-based algorithm for LTL language emptiness [9] and (ii) the SAT-based approaches [10]. We shall henceforth refer to these tools as NUSMV-B and NUSMV-S, respectively.[1] We implemented the algorithm based on native LTL_f through propositional SAT within an extended version of the AALTAF tool [6], with a novel dedicated extension to support past temporal operators.[2] We implemented the algorithm based on temporal resolution [11] satisfiability as a toolchain. First, it calls our variant of AALTAF to generate a file that is suitable for the TRP++ temporal resolution solver [11] using the dedicated extension to support past temporal operators. Then, the resulting file is submitted to TRP++, and finally the generated UC is post-processed to extract the auxiliary variables.[3]

The results. 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 in [6]. However, we remark that individual algorithms could extract different unsatisfiable cores among the diverse possible ones.

Concerning the computation time, Fig. 1(a) shows the number of problems solved by each algorithm within the 10 min timeout on the abscissae and the cumulative time taken to solve them on the ordinates. 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. Figure 1(b) is similar to Fig. 1(a), although it excludes the timings for the NUSMV-S runs that reached $k = 50$ albeit being unable to prove unsatisfiability (*unknown* answer) and thus construct an unsatisfiable core. In this figure, then, the virtual best considers only

---

[1]The extended version of NUSMV with these implementations is available at https://github.com/roveri-marco/ltlfuc.

[2]The source code for our extended version of AALTAF is available at https://github.com/roveri-marco/aaltaf-uc.

[3]For the experiments, we used the latest version of TRP++ taken from http://www.schuppan.de/viktor/trp++uc/.

the cases where the tool computed an unsatisfiable core. Both plots show that AALTAF outperforms the other tools 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 complementarity of the proposed approaches. A thorough investigation of the comparative assessment is reported in [1].

The overall minimum, maximum, average, and median best timings to return an UC are $0.0004\,\mathrm{s}$, $198.5054\,\mathrm{s}$, $1.4931\,\mathrm{s}$ and $0.0282\,\mathrm{s}$, respectively. The material to reproduce the experiments is available at https://github.com/roveri-marco/ltlfuc/archive/refs/tags/release-v0.zip.

References

[1] M. Roveri, C. Di Ciccio, C. Di Francescomarino, C. Ghidini, Computing unsatisfiable cores for LTL$_f$ specifications, arXiv:2203.04834v1, 2022.

[2] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013, pp. 854–860.

[3] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on LTL on finite traces: Insensitivity to infiniteness, in: AAAI, AAAI Press, 2014, pp. 1027–1033.

[4] A. Pnueli, The temporal logic of programs, in: FOCS, IEEE, 1977, pp. 46–57.

[5] V. Fionda, G. Greco, LTL on finite and process traces: Complexity results and a practical reasoner, J. Artif. Intell. Res. 63 (2018) 557–623.

[6] J. Li, G. Pu, Y. Zhang, M. Y. Vardi, K. Y. Rozier, SAT-based explicit LTLf satisfiability checking, Artif. Intell. 289 (2020) 103369.

[7] V. Schuppan, Towards a notion of unsatisfiable and unrealizable cores for LTL, Sci. Comput. Program. 77 (2012) 908–939.

[8] J. Li, S. Zhu, G. Pu, L. Zhang, M. Y. Vardi, SAT-based explicit LTL reasoning and its application to satisfiability checking, Formal Methods Syst. Des. 54 (2019) 164–190.

[9] E. M. Clarke, O. Grumberg, K. Hamaguchi, Another look at LTL model checking, Formal Methods Syst. Des. 10 (1997) 47–71.

[10] A. Biere, K. Heljanko, T. A. Junttila, T. Latvala, V. Schuppan, Linear encodings of bounded LTL model checking, Log. Methods Comput. Sci. 2 (2006).

[11] V. Schuppan, Extracting unsatisfiable cores for LTL via temporal resolution, Acta Informatica 53 (2016) 247–299.

[12] W. M. P. van der Aalst, M. Pesic, H. Schonenberg, Declarative workflows: Balancing between flexibility and support, Comput. Sci. Res. Dev. 23 (2009) 99–113.

[13] D. M. Gabbay, The declarative past and imperative future: Executable temporal logic for interactive systems, in: Temporal Logic in Specification, Springer, 1987, pp. 409–448.

[14] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella, NuSMV 2: An OpenSource Tool for Symbolic Model Checking, in: CAV, Springer, 2002, pp. 359–364.