BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

HyperLTL Model Checking

Ondrej Alexaj*

Abstract

HyperLTL Model Checking is an approach to verify a system against a given hyperproperty. In addition to expressing a property, HyperLTL allows us to evaluate the property against multiple executions of a system. Although the algorithmic approach based on automata has been established, it relies on standard ω-automata operations. The aim of this work is to outperform the complete state-of-the-art HyperLTL model checker by employing more efficient partial automata operations, particularly the complementation and inclusion. To achieve this, a novel modular-based complementation state-of-the-art tool has been leveraged, resulting in significant performance gains on the majority of testing instances. However, the results indicate the necessity for the development of a more effective inclusion checking algorithm. Finally, our approach to inclusion checking is also compared with the state-of-the-art tool. As a commonly used automata operation, our inclusion check could potentially contribute not only to the improvement of HyperLTL model checking but also to the advancement of other areas of verification.

*xalexa09@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

In the domain of hardware and software systems, formal verification is the process of proving or disproving the correctness of a system with respect to a given property. This is achieved through the use of formal methods that that provide a mathematical basis for specifying properties and modeling system behavior. Formal languages, automata theory and logics are some of the most important formal methods used in verification tasks [1, 2].

Model checking [1] is an automated verification method that systematically checks whether the property holds in the modelled system or not. The main advantage of this approach is the ability to provide a counterexample in case the property does not hold.

Hyperproperties were defined by Clarkson and Schneider in 2008 [3] as a set of trace properties. They point out properties that cannot be formulated as proper- ties of a single execution. In contrast to the properties of a single execution trace, which is satisfied by a trace, a hyper- property is satisfied by a set of traces. Robustness [4], path planning [5], generalized non-interference [6] etc. are examples of such properties.

HyperLTL [7] is an extension of linear temporal logic (LTL) serving as a formal base for expressing hyperproperties. An approach oftentimes referred to as Automata-Based Verification (ABV) [8], has been established to perform model checking.

Although HyperLTL MC's ABV approach of HyperLTL MC is decidable [8], it suffer from common ω-automata problems. The automata op- erations it consists of are the costliest ones. Namely, it includes complementation, automata product and inclusion checking.

In this work, we focus on optimizing the automata operations within the HyperLTL model checking by leveraging the more efficient complementation tool together with the implementation of our own inclusion check with the goal of generating as few states as possible.

In addition to being able to outperform the existing tool for HyperLTL MC, we believe that the techniques presented in this work can find their usecases in other formal verification tasks as well.

2. HyperLTL Model Checking scheme

In this section we explain the scheme from Figure 1.

Let $\varphi = Q_1\pi_1 Q_2\pi_2 \ldots Q_n\pi_n : \varphi^*$, with φ^* denoting the quantifier-free sub-formula of φ and $Q_i \in \{\exists, \forall\}$

for all $1 \leq i \leq n$. Firstly, a (nondeterministic) Büchi automaton A_{φ^*} equivalent to the LTL body φ^* is constructed. This is accomplished by the standard Tableau construction that creates automaton accepting exactly ω-words satisfying the φ^* [9]. This automaton's alphabet is $\Sigma_{\varphi^*} = (2^{AP})^n$, one set of atomic propositions for each trace quantifier. The next step is to inductively eliminate trace quantifiers. Suppose the following sub-formula of φ, $\psi = Q_k \pi_k : \varphi_k$. We can safely make an assumption that automaton $A_k = (Q, \Sigma, \delta, q_{in}, F)$ for φ_k is already constructed (automaton A_{φ^*} being the base case). Since Q_k is the k-th quantifier, the alphabet of the automaton A_k is $\Sigma = (2^{AP})^k$. Now, if $Q_k = \exists$, we can perform *existential projection*, which is intuitively the product of A_k and the Kripke structure $\mathcal{K} = (S, s_0, \delta, AP, L)$, necessary to associate the specification with the behavior. Formally, we construct an automaton $A_{k-1} = (Q \times S, \Sigma', \delta', (q_{in}, s_0), F \times S)$ where $\Sigma' = (2^{AP})^{k-1}$ and:

$$\begin{aligned} \delta'((q,s),(l_1,\ldots,l_{k-1})) = \\ = \{(r,s') \mid (s,s') \in \delta_{\mathcal{K}} \text{ and} \\ r \in \delta(q,(l_1,\ldots,l_{k-1},L(s)))\} \end{aligned} \quad (1)$$

where $(l_1,\ldots,l_{k-1})$ and $(l_1,\ldots,l_{k-1},L(s))$ are letters of automaton A_{k-1} and A_k respectively. An intuitive explanation of this definition is that we read along both the automaton and Kripke structure, choosing only transitions that are acceptable with respect to the current state of the system (Kripke structure). We, however, omitted the case where $Q_k = \forall$. This is transformed to the previous scenario using the law of double negation, i.e. $\neg\neg\forall\pi_k\varphi_k = \forall\pi_k\varphi_k$, which we can rewrite as $\neg\neg\forall\pi_k\varphi_k = \neg\exists\pi_k\neg\varphi_k$. Here the negations raise the need for the complementation procedure of Büchi automata.

After each quantifier is eliminated in the way described above, we end up with automaton over single-letter alphabet $\Sigma = (2^{AP})^0$. Now we just have to perform emptiness check over this automaton, meaning that $\mathcal{K} \models \varphi$ if and only if language of the automaton is non-empty. If the outermost quantifiers are universal, it is possible to perform an inclusion check between the Büchi automaton constructed from the Kripke structure and the Büchi automaton that has been inductively constructed.

3. Kofola vs AutoHyper

In the Figure 2 we can see a comparison of the execution times it takes to solve HyperLTL model checking instances[1], where Kofola is not optimized for inclusion. Specifically, in the Figure 2a, Boolean programs have been verified against the generalized non-interference property. Programs that plan robot movements on plans of sizes 10×10, 20×20, 40×40 and 60×60 are shown in Figure 2b. In these cases we suffered the most from inclusion. Verification of Symbolic programs is shown in Figure 2c. A summary comparison can be seen in the Figure 2d.

4. Language inclusion

We can reformulate the language inclusion problem of two ω-automata as follows:

$$L(A) \stackrel{?}{\subseteq} L(B) \iff L(A) \cap \overline{L(B)} \stackrel{?}{=} \emptyset,$$

consisting of common automata operations: complementation, product and emptiness check. We utilize several types of simulations to reduce generated state space as much as possible, as we believe that this is a way to go in optimizing language inclusion. With these simulations we are able to state *subsumptions* (e.g. implications (1) and (2)) that can decide inclusion and avoid constructing the entire counterexample, Figure 3.

5. Inclusion checker Kofola vs Spot

Although we outperform Spot in the generated state space (Figure 4 and Table 1), we have not yet been able to compete in the execution time, due to Spot being a highly optimized tool and Kofola being slower at the expense of modularity

6. Conclusions

First, we would like to highlight our success against *AutoHyper* on the test instances. In terms of language inclusion, we are already able to reduce the generated state space, but several other optimizations are in progress and will be integrated into Kofola in no time.

Acknowledgements

I would like to thank my supervisor Ing. Ondřej Lengál, Ph.D. for his guidance, help and patience during my work on this work.

[1] https://github.com/ondrik/automata-benchmarks/tree/master/omega/autohyper

References

[1] Miguel E. Andrés. Quantitative analysis of information leakage in probabilistic and nondeterministic systems, 2011.

[2] Bernd Finkbeiner. Automata, games, and verification, 2015.

[3] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, 2008.

[4] Pedro R. D'Argenio, Gilles Barthe, Sebastian Biewer, Bernd Finkbeiner, and Holger Hermanns. Is your software on dope? In Hongseok Yang, editor, *Programming Languages and Systems*, pages 83–110, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[5] Tzu-Han Hsu, César Sánchez, and Borzoo Bonakdarpour. Bounded model checking for hyperproperties. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 94–112, Cham, 2021. Springer International Publishing.

[6] D. McCullough. Noninterference and the composability of security properties. In *Proceedings. 1988 IEEE Symposium on Security and Privacy*, pages 177–186, 1988.

[7] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust*, pages 265–284, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[8] Bernd Finkbeiner. Logics and algorithms for hyperproperties. *ACM SIGLOG News*, 10(2):4–23, jul 2023.

[9] Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. pages 253–271, 09 1999.