Decompositions in Compositional Translation of LTLf to DFA

Yash Kankariya[1], **Suguman Bansal**[1]

[1]Georgia Institute of Technology
ykankariya@gatech.edu, suguman@gatech.edu

Abstract

Prior compositional methods in LTLf to DFA conversion have focused on improving the composition phase. In this work, we examine improvements to the decomposition phase that result in overall improvements in LTLf to DFA translation. Our work is based on reducing the structure of the underlying Abstract Syntax Tree (AST) of a formula such that the new AST results in fewer composition operations.

1 Introduction

Linear Temporal Logic over Finite Traces (LTLf) is a specification language that expresses rich and complex temporal behaviors over a finite time horizon (De Giacomo and Vardi 2013). LTLf finds broad-ranging applications, including robotics navigation and manipulation, automated synthesis, and reinforcement learning.

A critical component of all these applications is the conversion of LTLf to their equivalent *Deterministic Finite-state Automata* (DFA). For instance, consider LTLf synthesis, the automated construction of a reactive system that continuously interacts with an uncontrollable external environment from a high-level description of its desired behavior expressed in LTLf (De Giacomo and Vardi 2015). Here, the conversion of LTLf to DFA is necessary to establish a two-player adversarial game between the controllable system and the uncontrollable environment. The LTLf-to-DFA conversion has been shown to result in a double-exponential blow-up in the worst case, highlighting its inherent complexity. Despite the theoretical complexity, mastering the efficiency of LTLf to DFA conversion holds the key to advancing applications in these diverse domains.

Prior works on scalable and efficient LTLf-to-DFA conversions have devised *compositional methods* based on the syntax of the LTLf formula. These approaches decompose the formula into its *Abstract Syntax Tree* (AST) and employ a bottom-up approach to obtain the final DFA. To elaborate, first the DFAs are constructed at the node of the tree. Then the DFAs are composed using standard language-theoretic and automata-theoretic operations along the AST, till the final DFA is constructed at the root of the AST. In order to prevent state-space explosion, DFA minimization is performed aggressively at the nodes of the AST. This is the approach adopted in three state-of-the-art tools for LTLf-to-DFA conversion. The first is based on the tool MONA (Henriksen et al. 1995). Here, the algorithm constructs a binary AST and performs DFA minimization at every node of the tree. This was addressed in the tool Lisa (Bansal et al. 2020) where instead of a full AST, the algorithm decomposed the original formula at the outermost conjunction only, thus creating a k-ary AST of single depth. Thus, the roots are subformulas as opposed to atomic propositions. The k-ary structure offers flexibility in developing heuristics to choose the order in which component DFAs are composed. Finally, the current state-of-the-art Lydia extends Lisa by constructing complete k-ary trees till the roots that are occupied by propositions (De Giacomo and Favorito 2021).

All these prior approaches have focused on improving the composition phase. In this work, we focus on the decomposition phase with the goal of reducing the number of compositions for each formula. We observe that the decomposition of a formula into its AST may result in the duplication of semantically or syntactically equivalent formulas at various nodes. If not assembled together, this may result in repeated construction of identical DFAs. A naive approach to overcome this is to store all subformulas and their corresponding DFAs in an easy-access database. Before the construction of a DFA at a node, the database is accessed to ensure no duplication. With the use of BDD-based representations of the LTLf formula, one can prevent duplication of semantically equivalent subformulas as well.

In our work, we reduce the number of compositions even further. In this work, we re-arrange the AST to obtain a semantically equivalent graph with fewer number of edges, hence fewer compositions. For instance, consider the formula $\phi = \bigwedge_{i=1}^{k}(\psi \vee \phi_i)$. The AST obtained from syntax-driven decomposition is illustrated in Figure 1. It would require $2k-1$ compositions; k from each $(\psi \vee \phi_i)$ and $k-1$ from the outer conjunction. Despite BDD-based representations to identify that ψ is common to all, the number of compositions will remain at $2k-1$. Instead, fewer compositions would be required if the formula were represented as $\phi = \psi \vee \bigwedge_{i=1}^{k}\phi_i$. This requires k compositions only; $k-1$ from the conjunction and one more from the disjunction. The corresponding AST for the new formula is illustrated in Figure 1. Fewer compositions would improve the

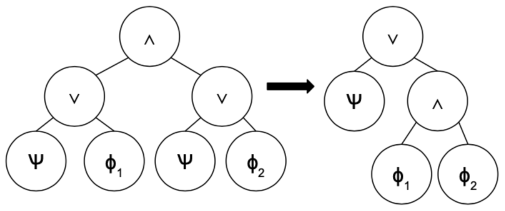

Figure 1: AST Optimization for $k = 2$

performance of LTLf-to-DFA translators. In our work, we present an approach that converts the original AST into an equivalent AST with fewer edges. The algorithm combines bottom-up traversal of the graph with syntactic-equivalence rules. This is used as a preprocessing step in the decomposition phase in the compositional approach to translate the formula into automata. We show that this results in significant performance improvement across the board i.e. runtime performance, number of benchmarks solved, and memory usage. Note that while we deploy the improvements to the decomposition phase in LTLf-to-DFA conversions, our approach could have broader applicability in other logic to automata translations.

2 Experimental Analysis

We have implemented our algorithm in C++ called Swift. We compare its performance against the current state-of-the-art tools Lisa and Lydia. Benchmarks were taken from the LTLf synthesis track of SYNTCOMP 2023. All experiments were performed on a system with dual-core 4GB RAM.

We compare the tools on runtime, number of benchmarks solved, and memory usage. Swift improves performance on all three accounts.

Runtime Figure 2 presents the performance on the nim benchmarks from the SYNTCOMP suite. The cactus plot demonstrates a substantial improvement in runtime performance compared to Lisa and Lydia. The nim-benchmarks are considered the hardest in this suite as the intermediate subformulas are known to generate large DFAs (i.e. nodes of the AST are large DFAs) while the final DFA is small. Hence having fewer DFAs to compute helps improve the performance over this class of benchmark. The improvement in performance can be attributed to our postprocessing resulting in a considerable reduction of operations.

Number of benchmarks and Memory Usage Figure 3 presents the performance on single- and double-counter benchmarks from the SYNTCOMP suite. These formulas have very few repeated subformulas. As seen in Figure 3, we saw a substantial improvement in runtime compared to Lisa-Explicit while obtaining comparable results to Lydia. However, our Swift solved more benchmarks than Lydiadue to reduced memory usage. These could be attributed to the smaller AST structure resulting in less storage.

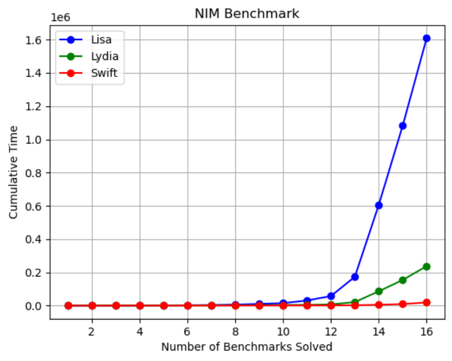

Figure 2: Runtime Cactus Plot of Nim Benchmarks

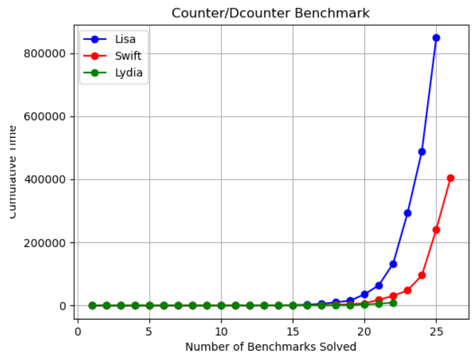

Figure 3: Runtime Cactus Plot of Counter Benchmarks

3 Future Work

In the future, we will incorporate further algorithmic improvements, including an on-the-fly construction of the reduced AST and further leverage the similarity and symmetry between formulas. We will also attempt more compact representations of the AST. Finally, we will examine the theoretical implications of our algorithm i.e. does it generate the minimal AST for a given formula.

References

Bansal, S.; Li, Y.; Tabajara, L.; and Vardi, M. 2020. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *Proc. of AAAI*.

De Giacomo, G.; and Favorito, M. 2021. Compositional approach to translate LTLf/LDLf into deterministic finite automata. In *Proc. of ICAPS*.

De Giacomo, G.; and Vardi, M. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 1558–1564. AAAI Press.

De Giacomo, G.; and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 854–860. AAAI Press.

Henriksen, J. G.; Jensen, J.; Jørgensen, M.; Klarlund, N.; Paige, R.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic second-order logic in practice. In *TACAS*, 89–110. Springer.