On the Power of Automata Minimization in Reactive Synthesis

Shufang Zhu
Sapienza Università di Roma
Roma, Italy
zhu@diag.uniroma1.it

Lucas M. Tabajara
Rice University
Houston, USA
lucasmt@rice.edu

Geguang Pu*
East China Normal University
Shanghai, China
ggpu@sei.ecnu.edu.cn

Moshe Y. Vardi
Rice University
Houston, USA
vardi@cs.rice.edu

Temporal logic is often used to describe temporal properties in AI applications. The most popular language for doing so is Linear Temporal Logic (LTL). Recently, LTL on finite traces, LTL$_f$, has been investigated in several contexts. In order to reason about LTL$_f$, formulas are typically compiled into deterministic finite automata (DFA), as the intermediate semantic representation. Moreover, due to the fact that DFAs have canonical representation, efficient minimization algorithms can be applied to maximally reduce DFA size, helping to speed up subsequent computations. Here, we present a thorough investigation on two classical minimization algorithms, namely, the Hopcroft and Brzozowski algorithms. More specifically, we show how to apply these algorithms to semi-symbolic (explicit states, symbolic transition functions) automata representation. We then compare the two algorithms in the context of an LTL$_f$-synthesis framework, starting from LTL$_f$ formulas. While earlier studies on comparing the two algorithms starting from randomly-generated automata concluded that neither algorithm dominates, our results suggest that starting from LTL$_f$ formulas, Hopcroft's algorithm is the best choice in the context of reactive synthesis. Deeper analysis explains why the supposed advantage of Brzozowski's algorithm does not materialize in practice.

1 Introduction

In many situations in Formal Methods and AI, we are interested in expressing properties over a sequence of successive states. Temporal logic, especially Linear Temporal Logic (LTL) has been thoroughly investigated for doing so [48]. Recently, a variant of LTL on finite traces, namely LTL$_f$, has been investigated [27]. LTL$_f$ found application in numerous AI contexts, as it is suitable for expressing properties over an unbounded but finite sequence of successive states. When reasoning about actions and planning, LTL$_f$ has been employed as a specification mechanism for finite-horizon temporally extended goals [18, 26]. As a specification language, we can use LTL$_f$ to specify desired properties in machine learning [17, 25, 56], program synthesis [28, 60, 16], Business Process Management (BPM) [46, 35, 20], Markov Decision Processes (MDPs) with non-Markovian rewards [11], MDPs policy synthesis [55], also non-Markovian planning and decision problems [10]. A general survey of applications of LTL$_f$ in AI and CS can be found in [27, 24].

In many applications, the common technique for reasoning about LTL$_f$ is compiling formulas into deterministic finite automata (DFA), cf. [60]. Unfortunately, the DFA size can be, in the worst case, doubly-exponential to the length of the formula [42]. Indeed, LTL$_f$-to-DFA compilation has been shown

*Corresponding author

to be the bottleneck of LTL$_f$ synthesis [60]. On the positive side, DFAs can be fully minimized thus helping to speed up subsequent computations [39, 13, 28]. Furthermore, there is evidence that the doubly-exponential blow-up of LTL$_f$-to-DFA compilation, does not tend to occur in practice [53]. This means that applications that require compiling temporal knowledge in LTL$_f$ to DFAs can still be implemented efficiently for many instances, as long as a good minimization algorithm is used. The natural question to ask, then, is: what is the best way in practice of constructing a minimal DFA from an LTL$_f$ formula?

An empirical evaluation of DFA-minimization algorithms can be found in [54]. That work compares two classical algorithms for constructing a minimal DFA from an NFA (nondeterministic finite automaton). The first is Hopcroft's algorithm [39], which first determinizes the NFA into a (not necessarily minimal) DFA, and then partitions the state space into equivalence classes. These equivalence classes then correspond to the states of the minimal DFA. The second is Brzozowski's algorithm [13], which reverses the automaton twice, determinizing and removing unreachable states after each reversal. This sequence of operations guarantees that the resulting DFA is minimal. The conclusion reached by [54] is that neither algorithm dominates across the board, and the best algorithm to use for a given NFA depends in part on the NFA's transition density.

A few aspects make the evaluation in [54] unsatisfactory for our purposes. First, the algorithms were compared considering an NFA as a starting point, while we are interested in obtaining minimal DFAs from LTL$_f$ formulas. This difference in initial representation may require certain steps of the algorithms to be implemented in a different way that affects their complexity. Second, the evaluation was performed on NFAs generated using a random model, which might not be representative of automata compiled from LTL$_f$ formulas. Third, automata generated from formulas tend to be semi-symbolic, having their transitions represented symbolically by data structures such as Binary Decision Diagrams (BDDs) [38], a commonly being used representation method that is more compact than the classical explicit representation. This semi-symbolic representation can also affect how certain operations are implemented and therefore the performance of the algorithms. Moreover, studies from [54] started with random automata, with the lack of looking into practical applications. We would like to look deeper into the context of applications. A number of applications require the step of automata minimization. Examples are monitoring [53], reactive synthesis [59], and Business Process Management (BPM) [46, 35, 20].

In this work we re-examine the comparison between the Hopcroft and Brzozowski algorithms, this time starting from LTL$_f$ formulas. In particular, we focus on the context of *reactive synthesis*, the algorithms of which usually make use of automata-theoretic techniques to automatically convert system properties described in high-level specification into a reactive system satisfying these properties, cf. [49]. In the standard approach for solving this problem for LTL$_f$, the LTL$_f$ specification is first compiled into a DFA, and then a system satisfying the specification is synthesized by solving a reachability game over this DFA [28]. The game-solving step is performed over a fully-symbolic representation of the DFA that also encodes the state space symbolically, and which is thus exponentially smaller than the explicit representation [60]. It has also been shown that synthesis techniques employing minimized DFA often shows dominating performance than the ones without [52, 5].

In our evaluation, Hopcroft's algorithm is represented by the tool MONA [38], a sophisticated platform for constructing DFAs from temporal logical specifications, commonly being used for minimal DFA construction in LTL$_f$ synthesis. It should be noted that, though MONA is able to handle full *second-order logic* (MSO), which subsumes *first-order logic* (FOL), it has been shown that expressing LTL$_f$ in FOL gives better performance [58]. MONA constructs a DFA in a bottom-up fashion, first constructing small DFAs for subformulas and then progressively combining them while applying Hopcroft minimization after each step. Since there is no preexisting tool that makes use of Brzozowski's algorithm, we show here how it can be effectively simulated within the existing framework of LTL$_f$ synthesis. This is done by using MONA to construct a minimal DFA for the *reverse* language of the LTL$_f$ specification, which can then be reversed, determinized and pruned of unreachable states to obtain the minimal DFA for the original language. The possible advantage of Brzozowski's algorithm is that the DFA for the reverse language of an LTL$_f$ formula is guaranteed to be at most exponential, rather than doubly exponential, in the size of the formula [19, 27].

We present two approaches for performing the final determinization step in Brzozowski's algorithm: an explicit approach, using routines implemented in the SPOT automata library [29], and a symbolic approach, which converts the reversed automaton directly into a fully symbolic DFA. The benefit of the fully symbolic approach is that it avoids constructing the semi-symbolic DFA, which may be exponentially larger than its fully-symbolic representation. On the other hand, this also means that the DFA is converted to the fully-symbolic representation before removing the unreachable states, which can lead to this representation being more complex than necessary. After the fully symbolic representation is constructed, computing the reachable states serves only to reduce the search space during synthesis, but does not reduce the size of the representation. While symbolic determinization has been discussed in prior works [2, 45], the role that symbolic determinization can play in the framework of LTL$_f$ [60, 5] and the impact it may have on algorithmic performance has not been investigated yet.

We compare Hopcroft's Algorithm and the two versions of Brzozowski's Algorithm on a number of LTL$_f$-synthesis benchmarks by evaluating not only the performance of the DFA construction, but also how the resulting fully-symbolic DFA affects the end-to-end synthesis performance. We find that, despite the minimal DFA for the reverse language having an exponentially smaller theoretical upper bound compared to the minimal DFA for the original language, in practice it is often a similar size or even larger. As a consequence, this observation suggests that Brzozowski's algorithm does not benefit in practice from performing determinization symbolically, as the fully-symbolic representation becomes much less efficient and the reachability computation does not compensate for the overhead. Finally, we observe that Hopcroft's algorithm dominates significantly in the wide majority of the cases. We thus conclude that unlike in [54], where the evaluation indicated that both minimization algorithms perform well in different cases, in the context of synthesis Hopcroft's algorithm is likely preferable. Moreover, the discrepancy between theory and practice, which leads to the disappearing of the exponential blow-up from the DFA of the reverse language to the DFA of the original LTL$_f$ formula, suggests the Hopcroft approach as a promising option also in other scenarios that require obtaining minimal DFA from LTL$_f$ [1].

2 Preliminaries

2.1 LTL$_f$ and Pure-Past LTL$_f$

Linear Temporal Logic over finite traces, called LTL$_f$ [27] extends propositional logic with finite-horizon temporal connectives. In particular, LTL$_f$ can be considered as a variant of Linear Temporal Logic (LTL) [48]. The key feature of LTL$_f$ is that it is interpreted over finite traces, rather than infinite traces as in LTL. Given a set of propositions $\mathcal{P}$, the syntax of LTL$_f$ is identical to LTL, and defined as:

$$\phi ::= \top \mid \bot \mid p \in \mathcal{P} \mid (\neg\phi) \mid (\phi_1 \wedge \phi_2) \mid (X\phi) \mid (\phi_1 U \phi_2).$$

$\top$ and $\bot$ represent *true* and *false* respectively. X (Next) and U (Until) are temporal connectives. Other temporal connectives can be expressed in terms of those. A *trace* $\rho = \rho[0], \rho[1], \ldots$ is a sequence of propositional assignments (sets), in which $\rho[m] \in 2^{\mathcal{P}}$ ($m \geq 0$) is the m-th assignment of ρ, and $|\rho|$

[1]Some proofs are moved to the appendix due to the lack of space. See full version on arXiv.

represents the length of ρ. Intuitively, $\rho[m]$ is interpreted as the set of propositions that are *true* at instant m. A trace ρ is *infinite* if $|\rho| = \infty$, denoted as $\rho \in (2^{\mathcal{P}})^{\omega}$, otherwise ρ is *finite*, denoted as $\rho \in (2^{\mathcal{P}})^{*}$. We assume standard temporal semantics from [27], and we write $\rho \models \phi$, if finite trace ρ satisfies ϕ at instant 0. We define the language of a formula ϕ as the set of traces satisfying ϕ, denoted as $\mathcal{L}(\phi)$.

We now introduce PLTL$_f$, which is the *pure-past* version of LTL$_f$ that considers the past instead of the future [58, 24]. PLTL$_f$ is defined as follows:

$$\theta ::= \top \mid \bot \mid p \in \mathcal{P} \mid (\neg\theta) \mid (\theta_1 \wedge \theta_2) \mid (Y\theta) \mid (\theta_1 S \theta_2).$$

We assume standard temporal semantics from [58, 24]. PLTL$_f$ has a natural interpretation on finite traces: the formula is satisfied if it holds in the last instant of a trace. Consider finite trace ρ, we say that $\rho \models \theta$, if $\rho, k-1 \models \theta$, where $k = |\rho|$. We define the language $\mathcal{L}(\theta)$ as the set of finite traces satisfying θ, that is, $\mathcal{L}(\theta) = \{\rho \mid \rho \models \theta\}$.

Consider LTL$_f$ formula ϕ, we can reverse it by replacing each temporal connective in ϕ with the corresponding *past* connective from PLTL$_f$ thus getting ϕ^R. X and U are replaced by Y and S, respectively. For a finite trace ρ, we define $\rho^R = \rho[|\rho|-1], \rho[|\rho|-2], \ldots, \rho[1], \rho[0]$ to be the reverse of ρ. We define the reverse of $\mathcal{L}$ as the set of reversed traces in $\mathcal{L}$, denoted as $\mathcal{L}^R$; formally, $\mathcal{L}^R = \{\rho^R \mid \rho \in \mathcal{L}\}$. The following theorem shows that PLTL$_f$ formula ϕ^R accepts exactly the reverse language of ϕ.

Theorem 1. [58] *Let $\mathcal{L}(\phi)$ be the language of LTL$_f$ formula ϕ and $\mathcal{L}^R(\phi)$ be the reverse language, then $\mathcal{L}^R(\phi) = \mathcal{L}(\phi^R)$.*

2.2 Automata Representations

An LTL$_f$ formula can be compiled into an automaton over finite words that accepts a trace if and only if that trace satisfies the formula. Here we introduce a few different automata representations. The difference between the representations here and the standard textbook representation of finite-state automata is that the alphabet is defined in terms of truth assignments to propositions.

Definition 1 (Nondeterministic Finite Automata)**.** *An NFA is represented as a tuple $\mathcal{N} = (\mathcal{P}, \mathcal{S}, \mathcal{S}_0, \eta, Acc)$, where • $\mathcal{P}$ is a finite set of propositions; • $\mathcal{S}$ is a finite set of states; • $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states; • $\eta : \mathcal{S} \times 2^{\mathcal{P}} \to 2^{\mathcal{S}}$ is the transition function such that given current state $s \in \mathcal{S}$ and an assignment $\sigma \in 2^{\mathcal{P}}$, η returns a set of successor states; • $Acc \subseteq \mathcal{S}$ is the set of accepting states.*

If there is only one initial state s_0 and η returns a unique successor for each state $s \in \mathcal{S}$ and assignment $\sigma \in 2^{\mathcal{P}}$, then we say that $\mathcal{N}$ is a deterministic finite automaton (DFA). In this case, η can be written in the form of $\eta : \mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$. The set of traces accepted by $\mathcal{N}$ is called the *language* of $\mathcal{N}$ and denoted by $\mathcal{L}(\mathcal{N})$. Moreover, we also introduce here a so-called codeterministic finite automaton (co-DFA) [47]. $\mathcal{N}$ is called a co-DFA if for each state $s \in \mathcal{S}$ and transition condition $\sigma \in 2^{\mathcal{P}}$, there is a unique predecessor d such that $\eta(d, \sigma) = s$. Intuitively, an NFA with a unique accepting state $Acc = \{s_{acc}\}$ is considered as a co-DFA if reversing all the transitions and switching $\mathcal{S}_0$ with Acc gives us a DFA.

A question that remains is how to represent the transition function η efficiently. It could be represented by a table mapping states and assignments in $2^{\mathcal{P}}$ to the set of successor states, but this table would necessarily be exponential in the number of propositions. In practice, from a given state it is usually the case that multiple assignments can lead to a same successor state. These assignments can then be represented collectively by a single Boolean formula λ. For a given state, the number of such formulas is usually much smaller than the number of assignments. Therefore, the transition function can alternatively be represented by a relation $H : \mathcal{S} \times \Lambda \times \mathcal{S}$, where Λ is a set of propositional formulas over

$\mathcal{P}$. We then have $(s,\lambda,d) \in H$ for a formula λ, iff $d \in \eta(s,\sigma)$ for every $\sigma \in 2^{\mathcal{P}}$ that satisfies λ. Intuitively, the tuples of H can be thought of as edges in the graph representation of the automaton, labeled by the propositional formulas that match the transitions. It should be noted that MONA [38] adopts this representation, representing propositional formulas as Binary Decision Diagrams (BDDs) [12].

We call the above a *semi-symbolic* automaton representation, as transitions are represented symbolically by propositional formulas but the states are still represented explicitly. In contrast, we now present a *fully-symbolic* (*symbolic* for short) representation, in which both states and transitions are represented symbolically. In the fully-symbolic representation, states are encoded using a set of *state variables* $\mathcal{Z}$, where a state corresponds to an assignment of $\mathcal{Z}$.

Definition 2 (Symbolic Deterministic Finite Automaton)**.** *A symbolic DFA of a corresponding explicit DFA $\mathcal{A} = (\mathcal{P},\mathcal{S},s_0,\eta,Acc)$, in which η is in the form of $\eta : \mathcal{S} \times 2^{\mathcal{P}} \to \mathcal{S}$, is represented as a tuple $\mathcal{D} = (\mathcal{P},\mathcal{Z},I,\delta,f)$, where*

- *$\mathcal{P}$ is the set of propositions as in $\mathcal{A}$;*
- *$\mathcal{Z}$ is a set of state variables with $|\mathcal{Z}| = \lceil \log|\mathcal{S}| \rceil$, and every state s in the explicit DFA corresponds to an assignment $Z \in 2^{\mathcal{Z}}$ of propositions in $\mathcal{Z}$;*
- *$I \in 2^{\mathcal{Z}}$ is the initial assignment corresponding to s_0;*
- *$\delta : 2^{\mathcal{Z}} \times 2^{\mathcal{P}} \to 2^{\mathcal{Z}}$ is the transition function. Given assignment Z of current state s and transition condition σ, $\delta(Z,\sigma)$ returns the assignment Z' corresponding to the successor state $s' = \eta(s,\sigma)$;*
- *f is a propositional formula over $\mathcal{Z}$ describing the accepting states, that is, each satisfying assignment Z of f corresponds to an accepting state $s \in Acc$.*

Note that the transition function δ can be represented by an indexed family consisting of a Boolean formula δ_z for each state variable $z \in \mathcal{Z}$, which when evaluated over an assignment to $\mathcal{Z} \cup \mathcal{P}$ returns the next assignment to z. Since the states are encoded into a logarithmic number of state variables, depending on the structure of these formulas, the symbolic representation can be exponentially smaller than the semi-symbolic representation.

2.3 Minimized DFA from NFA

For every NFA, there exists a unique smallest DFA that recognizes the same language, called the *canonical* or *minimal* DFA. A typical way to construct the canonical DFA for a given NFA is to determinize the automaton using subset construction [50] and then minimize it using, for example, Hopcroft's DFA minimization algorithm [39]. The idea of Hopcroft's algorithm is as follows. Consider state s in automaton $\mathcal{N}$, we define $\mathcal{L}(s)$ as the set of accepting words of $\mathcal{N}$ having s as the initial state. Note that a minimal DFA cannot have two different states such that $\mathcal{L}(s) = \mathcal{L}(s')$. Hopcroft's algorithm computes equivalence classes of states, such that two states s and s' are considered equivalent if $\mathcal{L}(s) = \mathcal{L}(s')$.

Theorem 2. [39] *Let $\mathcal{N}$ be an NFA. Then $\mathcal{A} = [equivalence \circ determinize](\mathcal{N})$ is the minimal DFA accepting the same language as $\mathcal{N}$.*

Fcuntion $determinize(\mathcal{N})$ is the DFA obtained by applying subset construction to $\mathcal{N}$, and function $equivalence(\mathcal{N})$ partitions the set of states into equivalence classes, which then form the states in the canonical DFA. The initial partition is Acc and $\mathcal{S} \backslash Acc$. Then at each iteration this partition is refined by splitting each equivalence class, until no longer possible. MONA [38], the state-of-the-art practical tool for constructing minimal DFA from logic specifications [38, 60], operates by induction on the structure of the input formulas, constructing DFAs for the subformulas and then combining them recursively, while applying Hopcroft's minimization algorithm after each step. We thus say that MONA follows the Hopcroft approach for minimization, through an optimized variant adapted to the semi-symbolic automata used by MONA.

A less direct way to construct minimal DFAs from NFAs, in which there is no explicit step of minimization, is due to Brzozowski [13]. We use the following formulation of Brzozowski's approach [54]. For notation, *reverse*$(\mathcal{N})$ is the function that maps the NFA $\mathcal{N} = (\mathcal{P}, \mathcal{S}, \mathcal{S}_0, \eta, Acc)$ to the NFA $\mathcal{N}^R = (\mathcal{P}, \mathcal{S}, Acc, \eta^R, \mathcal{S}_0)$, where $(d, \sigma, s) \in \eta^R$ iff $(s, \sigma, d) \in \eta$; *determinize*$(\mathcal{N})$ again returns the DFA by applying subset construction to $\mathcal{N}$; and *reachable*$(\mathcal{N})$ is the automaton resulting from removing all states that are not reachable from the initial states of $\mathcal{N}$.

Theorem 3. [13] *Let $\mathcal{N}$ be an NFA. Then $\mathcal{A} = [reachable \circ determinize \circ reverse]^2(\mathcal{N})$ is the minimal DFA accepting the same language as $\mathcal{N}$.*

2.4 Symbolic LTL$_f$ synthesis

Definition 3 (LTL$_f$ Synthesis)**.** *Let ϕ be an LTL$_f$ formula over $\mathcal{P}$ and $\mathcal{X}, \mathcal{Y}$ be two disjoint sets of propositions such that $\mathcal{X} \cup \mathcal{Y} = \mathcal{P}$. $\mathcal{X}$ is the set of* input variables *and $\mathcal{Y}$ is the set of* output variables*. ϕ is* realizable *with respect to $\langle \mathcal{X}, \mathcal{Y} \rangle$ if there exists a strategy $g : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, such that for an arbitrary infinite sequence $\pi = X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$ of propositional assignments over $\mathcal{X}$, we can find $k \geq 0$ such that ϕ is true in the finite trace $\rho = (X_0 \cup g(\varepsilon)), (X_1 \cup g(X_0)), \ldots, (X_k \cup g(X_0, X_1, \ldots, X_{k-1}))$.*

Intuitively, LTL$_f$ synthesis can be thought of as a game between two players: the *environment*, which controls the input variables, and the *agent*, which controls the output variables. Solving the synthesis problem means synthesizing a strategy g for the agent such that no matter how the environment behaves, the combined behavior trace of both players satisfies the logical specification ϕ [28]. There are two versions of the synthesis problem depending on which player acts first. Here we consider the agent as the first player, but a version where the environment moves first can be obtained with small modifications. In both versions, however, the agent decides when to end the game.

The state-of-the-art approach to LTL$_f$ synthesis is the symbolic approach proposed in [60]. This approach first translates the LTL$_f$ specification to a first-order formula, which is then fed to MONA to get the fully-minimized semi-symbolic DFA. This DFA is transformed to a fully-symbolic DFA, using BDDs [12] to represent each δ_z as well as the formula f for the accepting states. Solving a reachability game over this symbolic DFA settles the original synthesis problem. The game is solved by performing a least fixpoint computation over two Boolean formulas w over $\mathcal{Z}$ and t over $\mathcal{Z} \cup \mathcal{Y}$, which represent the set of all winning states and all pairs of winning states with winning outputs, respectively. Intuitively, winning states are those from which the agent has a winning strategy, and each winning state Z has a winning output Y that refers to the agent action returned by the winning strategy. t and w are initialized as $t_0(Z,Y) = f(Z)$ and $w_0(Z) = f(Z)$, since every accepting state is an agent winning state. Then t_{i+1} and w_{i+1} are constructed as follows:

- $t_{i+1}(Z,Y) = t_i(Z,Y) \vee (\neg w_i(Z) \wedge \forall X. w_i(\delta(X,Y,Z)))$
- $w_{i+1}(Z) = \exists Y. t_{i+1}(Z,Y)$

The computation reaches a fixpoint when $w_{i+1} \equiv w_i$. At this point, no more states will be added, and so all agent winning states have been collected. By evaluating w_i on I we can know if there exists a winning strategy. If that is the case, t_i can be used to compute a winning strategy. This can be done through the mechanism of Boolean synthesis [34]. We note that extensions of LTL$_f$ synthesis were studied in [59] and [57]. In all of these works, compiling LTL$_f$ formulas as corresponding DFAs proved to be the computational bottleneck.

3 Brzozowski's Algorithm from LTL$_f$

As mentioned in Section 2.3, a variation of Hopcroft's algorithm is already implemented in the tool MONA, which is the standard tool employed in LTL$_f$ synthesis applications. In contrast, there is no existing tool that directly implements Brzozowski's algorithm for temporal specifications. In this section, we describe how the algorithm can be adapted to compile an LTL$_f$ formula into a minimal DFA, presenting both an explicit and symbolic version of the algorithm. This section focuses on the theory of the algorithm; implementation details can be found in Section 4.1.

Theorem 3 in Section 2.3 describes how to obtain a minimal DFA from an NFA, here we start instead from an LTL$_f$ formula. This leads to the following sequence of operations:

1. **Reverse DFA construction:** Construct a minimal DFA that recognizes the reverse of the language of ϕ. This corresponds to the first round of $reachable \circ determinize \circ reverse$.
2. **Reversal into a co-DFA:** Reverse the DFA for the reverse language into a co-DFA for the original language. This corresponds to the *reverse* operation in the second round.
3. **Determinization and pruning:** The last two steps, corresponding to the $determinize$ and $reachable$ operations, can be performed either explicitly or symbolically.
   (a) **Explicit:** Apply subset construction to the co-DFA to obtain an explicit DFA, removing states that are not reachable from the initial states.
   (b) **Symbolic:** Convert the explicit co-DFA into a symbolic DFA (defined in Section 2.2). Next, compute a symbolic representation of the set of reachable states of the symbolic DFA. Since removing states cannot be easily done in the symbolic representation, the symbolic set of reachable states is instead used later to prune the search space during the game-solving step.

3.1 Reverse DFA Construction

Starting from an LTL$_f$ formula ϕ, we first produce a minimal DFA for the *reverse* language of ϕ. Note that, being minimal, this DFA has no unreachable states. It thus corresponds to a DFA obtained by applying the first round of operations $reachable \circ determinize \circ reverse$. Although minimality is a stronger condition than reachability, having the DFA be minimal improves the performance of future steps.

To construct such a DFA, we use the technique introduced in [58]. We first convert LTL$_f$ formula ϕ into a PLTL$_f$ formula ϕ^R for the reverse language. This can be done by simply replacing every temporal connective in ϕ with its corresponding past connective. Since PLTL$_f$ can be translated to first-order logic [58], ϕ^R can be converted into a DFA and minimized, for example using Hopcroft's algorithm.

It might seem odd to generate the minimal DFA for ϕ^R as an intermediate step in a minimization algorithm, when one could simply directly generate the minimal DFA for the original formula ϕ. The difference, however, is that the minimal DFA for the ϕ^R is guaranteed to have size at most exponential in the size of the formula [19, 27, 24], while the DFA for ϕ itself can be doubly-exponential [42]. Specifically, it is shown in [27] how to convert an LTL$_f$ formula to an alternating word automaton with a linear number of states, and it is shown in [19] how to convert an alternating word automaton to a DFA for the reverse language with exponential state blow-up. Therefore, constructing the DFA for the reverse language is, in theory, exponentially more efficient than the direct construction.

Theorem 4. *Let ϕ be an LTL$_f$ formula, $\mathcal{A}$ be the minimal DFA for ϕ^R. Then $\mathcal{A}' = [reachable \circ determinize \circ reverse](\mathcal{A})$ is the minimal DFA accepting the same language as ϕ.*

Proof. Since $\mathcal{A}$ is the minimal DFA for ϕ^R, $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\phi^R)$ holds. Moreover, ϕ^R accepts the reverse language of ϕ s.t. $\mathcal{L}(\phi^R) = \mathcal{L}^R(\phi)$, so we have $\mathcal{L}(\mathcal{A}) = \mathcal{L}^R(\phi)$. As stated in Theorem 3, the first round of $reachable \circ determinize \circ reverse$ returns a deterministic automaton that contains only reachable states and accepts the reverse language of the original input. Note all of these properties hold for $\mathcal{A}$. Specifically, $\mathcal{A}$ is a minimal DFA, and thus $\mathcal{A}$ is deterministic and contains only reachable states. Also, $\mathcal{L}(\mathcal{A}) = \mathcal{L}^R(\phi)$, so $\mathcal{A}$ accepts the reverse language of the original input ϕ. Therefore, after applying the second round of $reachable \circ determinize \circ reverse$ over $\mathcal{A}$, we get $\mathcal{A}'$, the minimal DFA of ϕ. □

Theorem 4 states that the construction of the minimal DFA for the reverse language is able to achieve the first round of $reachable \circ determinize \circ reverse$ in Theorem 3. In the remainder of this section we describe how to perform the second round of operations.

3.2 Reversal into a Co-DFA

The first step produces the minimal DFA $\mathcal{A}$ for the reverse language of ϕ. Reversing a semi-symbolic co-DFA can be done easily in linear time, by only swapping initial states with final states and swapping source with destination for every transition. Note that all transition conditions do not need to be changed. The result is a co-DFA for the original language ϕ. As explained in Section 2.2, a co-DFA is a special case of an NFA in which there is only a single transition into a state for each assignment.

More formally, let $\mathcal{A} = (\mathcal{P}, \mathcal{S}, \mathcal{S}_0, H, Acc)$ be the semi-symbolic DFA for the reverse language, with the (deterministic) transition relation given as $H \subseteq \mathcal{S} \times \Lambda \times \mathcal{S}$, where Λ is a set of propositional formulas, as described in Section 2.2. This representation of the transition relation is easy to obtain from the output of MONA. Reversing $\mathcal{A}$ produces the co-DFA $\mathcal{C} = (\mathcal{P}, \mathcal{S}, Acc, H^R, \mathcal{S}_0)$, where $H^R = \{(d, \lambda, s) \mid (s, \lambda, d) \in H\}$. Since co-DFA is a special case of NFA, for simplicity, later we still use NFA to refer to this co-DFA.

3.3 Explicit Minimal DFA Construction

The standard way of determinizing an NFA is using subset construction. This construction can be performed in the semi-symbolic representation, with explicit states and symbolic transitions. In this case, each state in the resulting DFA represents a subset of the states in the NFA, and a transition between two states representing subsets S_1 and S_2 is labeled by the disjunction of all labels λ such that $(s_1, \lambda, s_2) \in H$ for $s_1 \in S_1$ and $s_2 \in S_2$. In this semi-symbolic representation, finding the reachable states can be performed by a simple graph search on the graph of the automaton.

The problem with this explicit-state approach is that the subset construction causes an exponential blowup in the state space. This blowup can nullify the advantage obtained by constructing an exponential DFA for ϕ^R rather than a doubly-exponential DFA for ϕ. In the next section we describe how this problem may be mitigated by instead directly constructing a fully-symbolic representation of the DFA.

3.4 Symbolic Minimal DFA Construction

As described in Section 2.4, the state-of-the-art approach for solving LTL$_f$ synthesis uses a fully-symbolic representation of the DFA, which as noted in Section 2.2 can be exponentially smaller. Therefore, the exponential blowup caused by the explicit subset construction described in Section 3.3 above might be canceled out when the DFA is made fully-symbolic. Constructing the explicit DFA, then, seems like a waste that could be prevented by directly obtaining a symbolic DFA from the semi-symbolic co-DFA. With that in mind, in this section we describe an alternative approach that performs the subset construction and pruning of unreachable states symbolically.

3.4.1 Symbolic Subset Construction

The intuition of the symbolic determinization procedure is that, after subset construction, each state in the DFA corresponds to a set of NFA states. Each DFA state can therefore be represented by an assignment to a set of Boolean variables, one for each NFA state, where the variable is set to *true* if the corresponding state is in the set. This corresponds naturally to a symbolic representation $\mathcal{D}$ where each explicit state of the NFA is a state variable in $\mathcal{D}$. Therefore, the set of NFA states $\mathcal{S}$ is overloaded as the set of state variables in $\mathcal{D}$. Moreover, in addition to denoting a set of NFA states, S here is also used to denote a DFA state. In this way, $\mathcal{S}$ is able to encode the entire state space of $\mathcal{D}$. This approach is reminiscent of the symbolic determinization construction studied in [2] in the context of SAT-based safety LTL model checking, except that our symbolic approach here is BDD-based. (The symbolic determinization construction in [45] is in the context of Büchi and co-Büchi automata.)

Recall that the transition function $\delta : 2^{\mathcal{S}} \times 2^{\mathcal{P}} \to 2^{\mathcal{S}}$ in $\mathcal{D}$ can be represented as an indexed family $\{\delta_s \mid 2^{\mathcal{S}} \times 2^{\mathcal{P}} \to \{0,1\} \mid s \in \mathcal{S}\}$. Intuitively speaking, given current DFA state $S \in 2^{\mathcal{S}}$ and transition condition $\sigma \in 2^{\mathcal{P}}$, δ_s indicates whether state variable s is assigned as $true$ or $false$ in the successor state. Variable s is assigned as $true$ if there is a transition in the NFA from a state in S that leads to s under transition condition σ, and $false$ otherwise.

Therefore, given an NFA $\mathcal{N} = (\mathcal{P}, \mathcal{S}, \mathcal{S}_0, H, Acc)$, the symbolic determinization for the symbolic DFA $\mathcal{D} = (\mathcal{P}, \mathcal{S}, I, \delta, f)$ proceeds as follows:

- $\mathcal{S}$ is the set of state variables;
- $I \in 2^{\mathcal{S}}$ is such that $I(s) = 1$ if and only if $s \in \mathcal{S}_0$;
- $f = \bigvee_{s \in Acc} s$.
- $\delta_s : 2^{\mathcal{S}} \times 2^{\mathcal{P}} \to \{0,1\}$ is such that $\delta_s(S, \sigma) = 1$ iff $(d, \lambda, s) \in H$ for some d such that $S(d) = 1$ and λ such that $\sigma \models \lambda$. Each δ_s can be represented by a formula (or BDD) $\delta_s = \bigvee_{(d,\lambda,s) \in H} (d \wedge \lambda)$, with d interpreted as a state variable.

In order to show that the symbolic determinization described above is correct, i.e., that $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{N})$, we need to prove that the state where the DFA $\mathcal{D}$ reaches after reading a trace ρ corresponds exactly to the set of states where the NFA $\mathcal{N}$ can reach after reading ρ, which follows the standard subset construction. In the following, we use $\delta(S, \rho)$ to denote the DFA state that is reached from S by reading ρ. Likewise, $H(S, \rho)$ denotes the set $S' \subseteq \mathcal{S}$ of all NFA states that can be reached from all states $s \in S$ by reading ρ.

Lemma 1. *Let $\rho \in (2^{\mathcal{P}})^*$ be a finite trace. DFA state $\delta(I, \rho)$ encodes the set of NFA states $H(\mathcal{S}_0, \rho)$, that is, $\{s \mid \delta(I, \rho)(s) = 1\} = H(\mathcal{S}_0, \rho)$.*

The following theorem follows directly from Lemma 1.

Theorem 5. *$\mathcal{D}$ is equivalent to $\mathcal{N}$, that is, $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{N})$.*

The following theorem states the computational complexity of the symbolic determinization described above. The limiting factor is the construction of δ_s, each of which takes linear time. Therefore, the total complexity is quadratic.

Theorem 6. *The symbolic determinization can be done in quadratic time on the size of the NFA.*

3.4.2 Symbolic State-Space Pruning

The symbolic subset construction described in the previous subsection allows us to obtain a DFA for ϕ directly in symbolic representation. Although this representation is more compact, it presents further challenges for the final step of pruning unreachable states. This is because in the symbolic DFA the state space is fixed by the set of state variables. Since states are not represented explicitly, but rather implicitly by assignments over these variables, there is no easy way to remove states.

The alternative that we propose is to instead compute a symbolic representation of the set of reachable states, which can then be used during game-solving to restrict the search for a winning strategy. This means that the state space of the automaton, implicitly represented by the state variables, is not minimized, but during game-solving the additional, unreachable states are ignored.

We denote by $r(Z)$ the Boolean formula, over the set of state variables $\mathcal{Z}$, that is satisfied by an assignment $Z \in 2^{\mathcal{Z}}$ if Z encodes a state that is reachable from the initial state. We compute $r(Z)$ for the symbolic DFA $\mathcal{D} = (\mathcal{P}, \mathcal{Z}, I, \delta, f)$ by iterating the following recurrence until a fixpoint:

- $r_0(Z) = I(Z)$
- $r_{i+1}(Z) = r_i(Z) \vee \exists Z'.\exists X.\exists Y.r_i(Z') \wedge (\delta(Z', X \cup Y) = Z)$

Once a fixpoint is reached, the resulting formula $r(Z)$ denotes the set of reachable states of the automaton. Then, during the computation of the winning states as described in Section 2.4, we restrict t_i after each step to only those values of Z that correspond to reachable states.

4 Implementation and Evaluation

As mentioned in Section 2.3, Hopcroft's algorithm is represented by MONA [38], a sophisticated platform for obtaining minimized automata from logic specifications, in the way of constructing a minimal DFA by first generating DFAs for subformulas, then combining them recursively while applying Hopcroft's algorithm on the intermediate DFAs. For more details of MONA, we refer to [38]. In this section, we first present details of our implementation of Brzozowski's algorithm, and then show an experimental comparison between the two different minimization algorithms.

4.1 Implementation

As shown in Section 3, Brzozowski's algorithm consists of three steps: 1) reverse DFA construction, 2) reversal into a co-DFA, and 3) determinization and pruning. For the first step, we translate the PLTL$_f$ formula ϕ^R into a first-order formula $fol(\phi^R)$ following the translation in [58] and then use MONA to construct the DFA for ϕ^R. Since DFAs returned by MONA are always minimal, the reverse DFA constructed in this step is guaranteed to be at most exponential in the size of LTL$_f$ formula ϕ. Reversing this DFA into a co-DFA for ϕ is straightforward. Instead of implementing the reversal step as a separate operation, we optimize by performing the reversal while determinizing. There are two different versions of the determinization and pruning step: explicit and symbolic. We now elaborate on them. Note that each version starts with the DFA of ϕ^R, and we combine the reversal and the subsequent operations of subset construction followed by state-space pruning.

4.1.1 Explicit Minimal DFA Construction

Inspired by [5], we borrow the rich APIs from SPOT [29], a well-developed platform for automata manipulation, to perform each computation step, subset construction in particular. It should be noted that, SPOT adopts the semi-symbolic representation for automata, where the states are explicit and transitions are symbolic, and therefore, we use it to implement the explicit approach. Note that SPOT is a platform for ω-automata (automata over infinite words) manipulation. Therefore we represent the co-DFA as a weak Büchi Automaton (wBA) [23].

Since the reversal step is straightforward, to simplify the description we consider the co-DFA as the starting point to better show the implementation details. The transformation from co-DFA to wBA follows the techniques presented in [5]. Intuitively, the technique of transforming to wBA is similar to the translation from LTL$_f$ to LTL [27]. Note that in the translation from LTL$_f$ to LTL, a fresh proposition $alive \notin \mathcal{P}$ is introduced and required to stay $true$ until the LTL$_f$ formula is satisfied and then stay $false$ forever. In the transformation from co-DFA to wBA, we again use the same proposition *alive* and introduce a *sink* state that is triggered by the first moment of *alive* being false, such that the finite trace is accepted. Moreover, this *sink* state is considered as the unique accepting state in the wBA to make sure that *alive* stays $false$ forever. Formally, if a co-DFA accepts language $\mathcal{L}(\mathcal{C})$, then its wBA accepts infinite words in $\{(\rho \wedge alive) \cdot (\neg alive)^{\omega} \mid \rho \in \mathcal{L}(\mathcal{C})\}$, where $\rho \wedge alive$ denotes that *alive* holds at each instant of finite trace ρ. Given co-DFA $\mathcal{C} = (\mathcal{P}, \mathcal{S}, \mathcal{S}_0, H, Acc)$, we construct the wBA as follows: 1) 2) introduce an extra state *sink*; 3) for each accepting state s in Acc, add a transition from s to *sink*, with transition condition $\neg alive$; 4) for each transition in between states in $\mathcal{C}$, change the transition condition λ to $\lambda \wedge alive$; 5) add a self-loop for state *sink* on $\neg alive$; 6) assign *sink* as the unique accepting state.

Theorem 7. *Let $\mathcal{C}$ be a co-DFA, and $\mathcal{B}$ be the wBA generated from the construction (1)-(5) above. Then we have $\mathcal{L}(\mathcal{B}) = \{(\rho \wedge alive) \cdot (\neg alive)^{\omega} \mid \rho \in \mathcal{L}(\mathcal{C})\}$.*

Then, we are able to use SPOT APIs for wBA to conduct subset construction and unreachable states pruning, thus obtaining the wDBA $\mathcal{DB}$. The corresponding API functions are tgba_powerset() and purge_unreachable_states(), respectively. Finally, we convert the wDBA $\mathcal{DB}$ back to DFA as follows: a) remove the unique accepting state $\{sink\}$ and transitions leading to or coming from $\{sink\}$; b) eliminate *alive* on each transition by assigning it to $true$; c) assign all the states that move to $\{sink\}$ with transition condition $\neg alive$ as the accepting states of the DFA.

Theorem 8. *Let $\mathcal{C}$ be a co-DFA, $\mathcal{B}$ the corresponding wBA, $\mathcal{DB}$ the wDBA from SPOT, and $\mathcal{A}$ be the DFA obtained from the construction (a)-(c) above. Then $\mathcal{A}$ is minimal.*

4.1.2 Symbolic Minimal DFA Construction

Instead of having each DFA state explicit, the symbolic approach described in Section 3.4 is able to directly construct a symbolic DFA as in Definition 2. Here, we follow the representation technique used by [60], where the propositional formulas for the transition function and accepting states are represented by Binary Decision Diagrams (BDDs). Moreover, the reversal step is again combined with the symbolic subset construction operation into one step. Thus, for state variable s, in order to construct BDD for formula $\delta_s : 2^{\mathcal{S}} \times 2^{\mathcal{P}} \to \{0,1\}$, we have to take care of switching the current and successor states of a given transition from the reverse DFA. The same for exchanging BDDs of initial and accepting states.

As for the state-space pruning, formulas represented as BDDs allow us to perform the usual Boolean operations, such as conjunction, disjunction and quantifier elimination. After obtaining the set of reachable states $r(Z)$, during the computation of the winning states as described in Section 2.4, we need to restrict t_i after each step to only those values of Z that correspond to reachable states. There are two ways in which we can do this. The most obvious way is to simply take the conjunction of $t_i(Z,Y)$ with $r(Z)$, thus removing all assignments that correspond to unreachable states. The second option, since we are using BDDs to represent the Boolean formulas t_i and r, is to apply the standard BDD Restrict operation [51]. The BDD produced by Restrict(t_i, r) still returns 1 for all satisfying assignments to t_i that also satisfy r. Those satisfying assignments that do not satisfy r, however, are selectively mapped to either 1 or 0, using heuristics to try to make a choice that will lead to a smaller BDD. This corresponds to essentially choosing to keep a subset of the unreachable states if that will lead to a smaller symbolic representation of the set of winning states.

Note that the predecessor computation used to compute t_{i+1} may add unreachable states, therefore it is necessary to apply the conjunction or restriction operation at every iteration, rather than just once.

4.2 Experimental Evaluation

In order to evaluate the Hopcroft's and Brzozowski's minimization algorithms starting from LTL$_f$ formulas, we focus on the context of temporal synthesis. To do so, we conducted extensive experiments over different classes of benchmarks curated from prior works, spanning classes of realistic and synthetic benchmarks [60, 52, 5]. The benchmarks consist of two classes. The first class of benchmarks is the *Random* family, composed of 1000 LTL$_f$ formulas formed by random conjunction, generated as described in [60]. The second one is from [52, 5], and describes two-player games, split into the *Single-Counter*, *Double-Counters* and *Nim* benchmark families. Here we assign the agent as the first-player. It should be noted that, although different player order might lead to different realizability result, the automata minimization performance, nevertheless, stays the same.

The results shown here represent the end-to-end execution of the synthesis algorithms, from an LTL$_f$ specification to a winning strategy. Therefore, they include both the time for LTL$_f$-to-DFA compilation and game solving. All tests were run on a computer cluster with exclusive access to a node with Intel(R) Xeon(R) CPU E5-2650 v2 processors. Timeout was 1000 seconds and memory out was 8G.

Since there are two ways of performing Brzozowski's algorithm, as presented in Section 3, we have in total 3 different approaches, namely Hopcroft, Explicit-Brzozowski and Symbolic-Brzozowski. As introduced in Section 4.1.2, during symbolic state-space pruning we are able to either apply restriction or conjunction to access only reachable states during game-solving. We show only the results using restriction, as the difference is not significant and in most cases restriction gives slightly better results.

Figure 1 shows a cactus plot [2] comparing how the three different approaches perform on *Random* benchmarks. The curves show how many instances can be solved with a given timeout. The further up the curve is, the more benchmarks could be solved in less time. The graph shows that Hopcroft's minimization algorithm is in fact able to solve many more cases than both versions of Brzozowski's algorithm. Furthermore, the explicit Brzozowski approach slightly outperforms the symbolic one. In spite that with time limit of 10 seconds the symbolic version is able to handle more cases than the explicit one, if we take 1000 seconds as the time limit the explicit version is able to handle in total more cases than the symbolic one. This shows that the explicit version is more scalable than the symbolic one. Increasing the time limit does not change the results, since unsolved instances reached the memory limit.

The results are not much different in the case of the non-random benchmarks, shown in Figure 2. There we can see that symbolic Brzozowski's algorithm timed out for the vast majority of instances, only being able to solve the smaller instances of the *Nim* family, and none of the instances of the *Single-Counter* and *Double-Counters* families. The explicit version performs slightly better, being able to handle some smaller instances of the *Single-Counter* and *Double-Counters* families. Hopcroft's algorithm, on the other hand, can solve a large number of instances within the timeout.

The results shown above allow us to answer the question of which minimization algorithm is more efficient in the context of temporal synthesis. Our data points to Hopcroft's algorithm as the better choice. It might seem surprising that Hopcroft's algorithm outperforms Brzozowski so significantly.

[2] Figures best viewed on a computer.

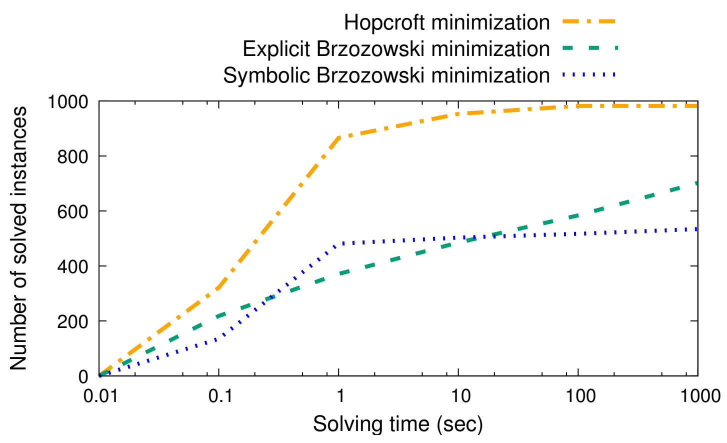

Figure 1: Total Running time with Hopcroft's or Brzozowski's minimization algorithms on *Random* benchmarks.

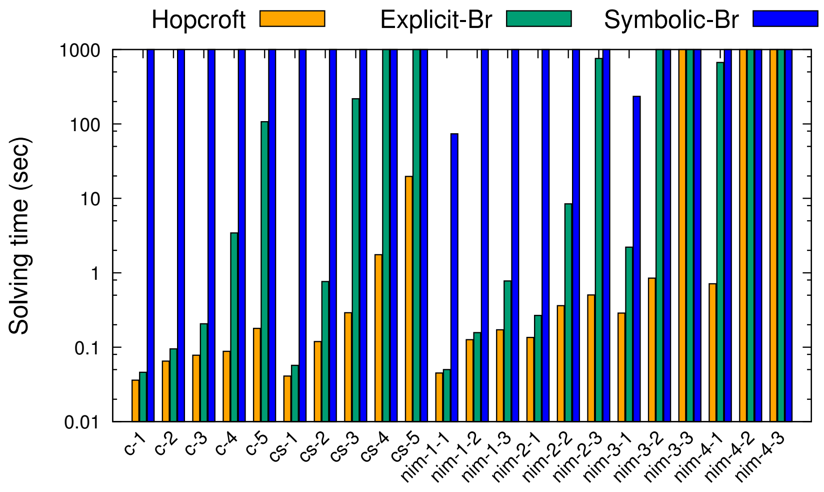

Figure 2: Total Running time with Hopcroft's or Brzozowski's minimization algorithms on *Single-Counter*, *Double-Counters* and *Nim* benchmarks.

The symbolic version in particular was expected to benefit from the fact that it is able to avoid ever having to construct the explicit DFA for the LTL$_f$ formula, instead constructing only the DFA for the reverse language, which should be exponentially smaller. Yet, it fails to compete even against the explicit version. Understanding the failure of Brzozowski's algorithm requires a more in-depth investigation of the internals of the minimization procedure. We perform this analysis in the next section.

5 Analysis and Discussion

To understand the reasons for the results that we observed in Section 4, we have taken a closer look at the comparison between the minimal DFA for the formula and the DFA for the reverse language obtained in the first half of the Brzozowski's construction. We started by measuring the number of states of the DFA and reverse DFA constructed for *Random* formulas. Figure 3 displays a scatter plot (in log scale) comparing the two for each instance. The blue curve indicates an exponential blowup of the number

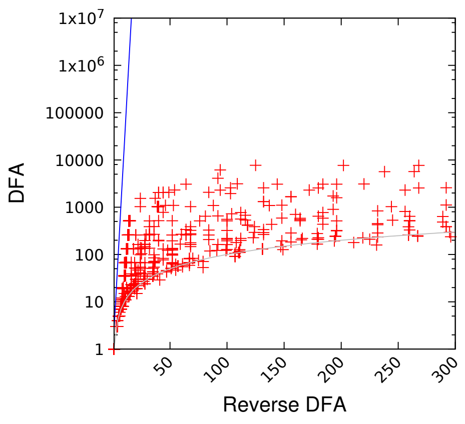

Figure 3: #states on *Random* benchmarks.

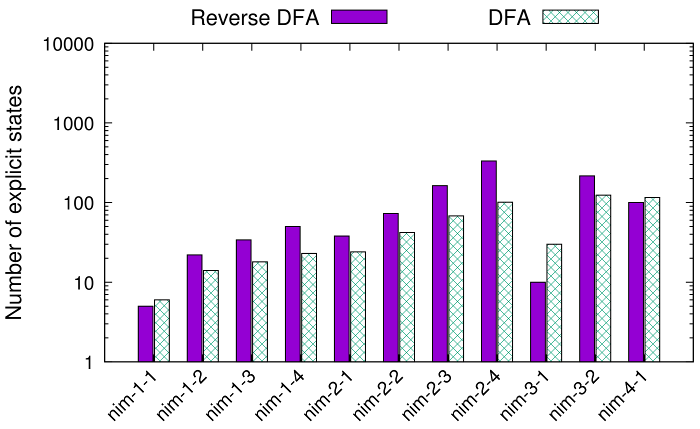

Figure 4: #states on *Nim* benchmarks.

of states of the reverse DFA. In addition, the gray curve represents the points where the x-axis value is equal to the y-axis value. Thus points above the gray curve represent instances where the minimal DFA is larger than the reverse DFA. In several cases, the reverse DFA is indeed smaller, as expected. We observe, however, that there is a significant number of cases where the two automata tend to have approximately the same number of states, which differs from what the theory would lead us to believe. In such cases, the benefits of using Brzozowski's construction disappear, as we can expect no advantage in constructing the minimal reverse DFA instead of directly constructing the DFA for the formula.

Note, furthermore, that even in those cases where the reverse DFA is indeed smaller, these points are far below the blue curve, indicating that they are not *exponentially* smaller. This is a problem because, after being reversed again, the reverse DFA will go through a subset construction, which causes an exponential blowup. This is represented in the symbolic version of Brzozowski's algorithm by the number of state variables in the symbolic representation being linear in the number of the states of the reverse DFA, instead of logarithmic in the number of states of the DFA. Therefore, unless the reverse DFA is exponentially smaller, the number of state variables will be larger than in the symbolic representation of the explicitly-constructed minimal DFA. This in turn impacts the performance of the winning-strategy computation. The increase in the state space during subset construction seems to be the main reason for failures of the Brzozowski approach. This is because the minimal reverse-DFA construction itself already takes comparable time to the minimal DFA construction and succeeds in almost all 1000 benchmarks. Moreover, Brzozowski's method requires further step of determinization on the reverse-DFA, which leads to another exponential blowup that takes a lot of costs, and therefore accounts for the failure of the performance.

Interestingly, although the symbolic state-space pruning helps with the large state space during synthesis, significantly reducing the size of the BDDs representing the sets of winning states at each iteration, the computation of the set of reachable states itself ends up consuming a majority of the running time. So it turns out to not be helpful in getting the symbolic version of Brzozowski's algorithm scale. The situation for the non-random benchmarks is even more extreme. Figures 4 and 5 show the comparison of DFA and reverse-DFA size for the *Nim* and *Single-Counter* benchmarks, respectively [3]. With the exception of a few of the smaller benchmarks in the *Nim* family, in these instances the DFA is actually

[3]The plot for the *Double-Counters* benchmarks shows analogous results to the *Single-Counter* benchmarks, and can be found in the appendix. The full version is on arXiv

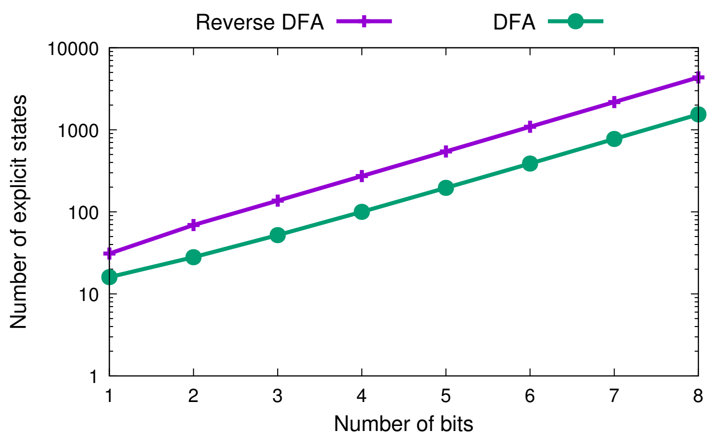

Figure 5: #states on *Single-Counter* benchmarks.

smaller than the reverse DFA. The results for the counter benchmarks in particular are useful to better understand where our assumptions are violated, as we can observe the scalability of the automata in terms of the number of bits n in the counter, which is proportional to the formula length. The plots (in log scale) show that the reverse DFA grows exponentially with n, which is the predicted behavior. Yet, the DFA is exponential as well, rather than doubly-exponential.

These results highlight an important detail that is easily overlooked in the justification for the reverse DFA construction in Section 3.1: the theoretical lower bounds on automata sizes refer to the *worst-case*. This means that even though there are formulas for which the smallest DFA is doubly exponential, it might be that such cases occur very rarely. Our experimental results suggest that this might indeed be the case. This is consistent with previous results from [53], that minimal DFA constructed from temporal formulas are often orders of magnitude smaller in practice than the corresponding NFA. Thus, even though the worst-case size of the reverse DFA is exponentially smaller, our conclusion is that in practice the worst case is not common enough for making an approach based on constructing the reverse DFA worthwhile. Therefore, directly constructing the minimal DFA using Hopcroft's algorithm seems to be a better option for synthesis than employing Brzozowski's construction.

Furthermore, note that the size of the reverse DFA of LTL$_f$ formula ϕ is actually the size of the minimal DFA of PLTL$_f$ formula ϕ^R. That is to say, despite the fact that the DFA of PLTL$_f$ is supposed to be exponentially smaller than the DFA of LTL$_f$, referring to the theoretical advantage of PLTL$_f$ over LTL$_f$ when compiled into the corresponding DFA [24], our data suggests that this advantage may not be common in practice. Thus, we believe that this disappearing advantage applies not only to reactive synthesis but also to other applications of LTL$_f$ that use compilation to DFA. In the future, we would like to revisit this problem in other applications to confirm our conjecture.

Acknowledgments

Research partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), National Key R&D Program of China (2020AAA0107800), NSF grants IIS-1527668, CCF-1704883, IIS-1830549, DoD MURI grant N00014-20-1-2787, and an award from the Maryland Procurement Office.

References

[1] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman (1974): *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[2] Roy Armoni, Sergey Egorov, Ranan Fraer, Dmitry Korchemny & Moshe Y. Vardi (2005): *Efficient LTL compilation for SAT-based model checking*. In: ICCAD 2005, pp. 877–884, doi:10.1109/ICCAD.2005.1560185.

[3] Fahiem Bacchus & Froduald Kabanza (1998): *Planning for Temporally Extended Goals*. Ann. Math. Artif. Intell. 22(1-2), pp. 5–27, doi:10.1023/A:1018985923441.

[4] Fahiem Bacchus & Froduald Kabanza (2000): *Using temporal logics to express search control knowledge for planning*. Artif. Intell. 116(1-2), pp. 123–191, doi:10.1016/S0004-3702(99)00071-5.

[5] Suguman Bansal, Yong Li, Lucas M. Tabajara & Moshe Y. Vardi (2020): *Hybrid Compositional Reasoning for Reactive Synthesis from Finite-Horizon Specifications*. In: AAAI, pp. 9766–9774, doi:10.1609/aaai.v34i06.6528.

[6] Nicola Bertoglio, Gianfranco Lamperti & Marina Zanella (2019): *Temporal Diagnosis of Discrete-Event Systems with Dual Knowledge Compilation*. In Andreas Holzinger, Peter Kieseberg, A Min Tjoa & Edgar R. Weippl, editors: Machine Learning and Knowledge Extraction, Lecture Notes in Computer Science 11713, Springer, pp. 333–352, doi:10.1007/978-3-030-29726-8_21.

[7] Meghyn Bienvenu, Christian Fritz & Sheila A. McIlraith: *Planning with Qualitative Temporal Preferences*. In Patrick Doherty, John Mylopoulos & Christopher A. Welty, editors: KR.

[8] Roderick Bloem (2015): *Reactive Synthesis*. In: FMCAD, IEEE, p. 3, doi:10.1109/FMCAD.2015.7542241.

[9] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Martin Weiglhofer (2007): *Specify, Compile, Run: Hardware from PSL*. Electron. Notes Theor. Comput. Sci. 190(4), pp. 3–16, doi:10.1016/j.entcs.2007.09.004.

[10] Ronen I. Brafman & Giuseppe De Giacomo (2019): *Planning for LTL_f /LDL_f Goals in Non-Markovian Fully Observable Nondeterministic Domains*. In Sarit Kraus, editor: IJCAI, pp. 1602–1608, doi:10.24963/ijcai.2019/222.

[11] Ronen I. Brafman, Giuseppe De Giacomo & Fabio Patrizi (2018): *LTL_f/LDL_f Non-Markovian Rewards*. In Sheila A. McIlraith & Kilian Q. Weinberger, editors: AAAI, pp. 1771–1778.

[12] Randal E. Bryant (1992): *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*. ACM Comput. Surv. 24(3), pp. 293–318, doi:10.1145/136035.136043.

[13] Janusz A. Brzozowski (1962): *Canonical Regular Expressions and Minimal State Graphs for Definite Events*.

[14] Marco Cadoli & Francesco M. Donini (1997): *A Survey on Knowledge Compilation*. AI Commun. 10(3-4), pp. 137–150.

[15] Diego Calvanese, Giuseppe De Giacomo & Moshe Y. Vardi: *Reasoning about Actions and Planning in LTL Action Theories*. In Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness & Mary-Anne Williams, editors: KR.

[16] Alberto Camacho, Jorge A. Baier, Christian J. Muise & Sheila A. McIlraith (2018): *Finite LTL Synthesis as Planning*. In: ICAPS, pp. 29–38.

[17] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano & Sheila A. McIlraith (2019): *LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning*. In Sarit Kraus, editor: IJCAI, pp. 6065–6073, doi:10.24963/ijcai.2019/840.

[18] Alberto Camacho, Eleni Triantafillou, Christian Muise, Jorge A. Baier & Sheila McIlraith (2017): *Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces*. In: AAAI, pp. 3716–3724.

[19] A.K. Chandra, D.C. Kozen & L.J. Stockmeyer (1981): *Alternation*. J. ACM 28(1), pp. 114–133, doi:10.1145/322234.322243.

[20] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali & Jan Mendling (2017): *Resolving inconsistencies and redundancies in declarative process models*. Inf. Syst. 64, pp. 425–446, doi:10.1016/j.is.2016.09.005.

[21] Luca Console, Paolo Terenziani & Daniele Theseider Dupré (2002): *Local Reasoning and Knowledge Compilation for Efficient Temporal Abduction*. IEEE Trans. Knowl. Data Eng. 14(6), pp. 1230–1248, doi:10.1109/TKDE.2002.1047764.

[22] Adnan Darwiche & Pierre Marquis (2002): *A Knowledge Compilation Map*. J. Artif. Intell. Res. 17, pp. 229–264, doi:10.1613/jair.989.

[23] Christian Dax, Jochen Eisinger & Felix Klaedtke (2007): *Mechanizing the Powerset Construction for Restricted Classes of* omega *-Automata*. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino & Yoshio Okamura, editors: ATVA, pp. 223–236, doi:10.1007/978-3-540-75596-8_17.

[24] Giuseppe De Giacomo, Antonio Di Stasio, Francesco Fuggitti & Sasha Rubin (2020): *Pure-Past Linear Temporal and Dynamic Logic on Finite Traces*. In Christian Bessiere, editor: IJCAI, pp. 4959–4965, doi:10.24963/ijcai.2020/690.

[25] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito & Fabio Patrizi (2019): *Foundations for Restraining Bolts: Reinforcement Learning with LTL_f/LDL_f Restraining Specifications*. In: ICAPS, pp. 128–136.

[26] Giuseppe De Giacomo & Sasha Rubin (2018): *Automata-Theoretic Foundations of FOND Planning for LTL_f/LDL_f Goals*. In: IJCAI, pp. 4729–4735, doi:10.24963/ijcai.2018/657.

[27] Giuseppe De Giacomo & Moshe Y. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In: IJCAI, pp. 854–860, doi:10.5555/2540128.2540252.

[28] Giuseppe De Giacomo & Moshe Y. Vardi (2015): *Synthesis for LTL and LDL on Finite Traces*. In: IJCAI, pp. 1558–1564, doi:10.5555/2832415.2832466.

[29] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault & Laurent Xu (2016): *Spot 2.0 — A Framework for LTL and ω-automata Manipulation*. In: ATVA, pp. 122–129, doi:10.1007/978-3-319-46520-3_8.

[30] Sonali Dutta, Moshe Y. Vardi & Deian Tabakov (2013): *CHIMP: A Tool for Assertion-Based Dynamic Verification of SystemC Models*. In: DIFTS@FMCAD.

[31] Ronald Fagin, Joseph Y. Halpern, Yoram Moses & Moshe Y. Vardi (1995): *Reasoning About Knowledge*. MIT Press, doi:10.7551/mitpress/5803.001.0001.

[32] Michael Fisher & Michael J. Wooldridge (2005): *Temporal Reasoning in Agent-Based Systems*. In Michael Fisher, Dov M. Gabbay & Lluís Vila, editors: Handbook of Temporal Reasoning in Artificial Intelligence, Foundations of Artificial Intelligence 1, Elsevier, pp. 469–495, doi:10.1016/S1574-6526(05)80017-3.

[33] Seth Fogarty, Orna Kupferman, Moshe Y. Vardi & Thomas Wilke (2013): *Profile Trees for Büchi Word Automata, with Application to Determinization*. In: GandALF, pp. 107–121, doi:10.4204/EPTCS.119.11.

[34] Dror Fried, Lucas M. Tabajara & Moshe Y. Vardi (2016): *BDD-Based Boolean Functional Synthesis*. In: CAV, pp. 402–421, doi:10.1007/978-3-319-41540-6_22.

[35] Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi & Marco Montali (2014): *Monitoring Business Metaconstraints Based on LTL and LDL for Finite Traces*. In Shazia Wasim Sadiq, Pnina Soffer & Hagen Völzer, editors: BPM, Lecture Notes in Computer Science 8659, pp. 1–17, doi:10.1007/978-3-319-10172-9_1.

[36] Giuseppe De Giacomo & Moshe Y. Vardi (1999): *Automata-Theoretic Approach to Planning for Temporally Extended Goals*. In Susanne Biundo & Maria Fox, editors: ECP, Lecture Notes in Computer Science 1809, Springer, pp. 226–238, doi:10.1007/10720246_18.

[37] Keliang He, Andrew M. Wells, Lydia E. Kavraki & Moshe Y. Vardi (2019): *Efficient Symbolic Reactive Synthesis for Finite-Horizon Tasks*. In: ICRA, pp. 8993–8999, doi:10.1109/ICRA.2019.8794170.

[38] Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe & Anders Sandholm (1995): *Mona: Monadic Second-order Logic in Practice*. In: TACAS, pp. 89–110, doi:10.1007/3-540-60630-0_5.

[39] John E. Hopcroft (1971): *An n Log n Algorithm for Minimizing States in a Finite Automaton*. Technical Report, Stanford, CA, USA.

[40] Orna Kupferman (2012): *Recent Challenges and Ideas in Temporal Synthesis*. In: *SOFSEM*, pp. 88–98, doi:10.1007/978-3-642-27660-6_8.

[41] Orna Kupferman & Moshe Y. Vardi (1998): *Freedom, Weakness, and Determinism: From Linear-Time to Branching-Time*. In: *LICS*, pp. 81–92, doi:10.1109/LICS.1998.705645.

[42] Orna Kupferman & Moshe Y. Vardi (2001): *Model Checking of Safety Properties*. *Formal Methods in System Design* 19(3), pp. 291–314, doi:10.1023/A:1011254632723.

[43] Orna Kupferman & Moshe Y. Vardi (2005): *Safraless Decision Procedures*. In: *FOCS*, pp. 531–542, doi:10.1109/SFCS.2005.66.

[44] Orna Lichtenstein, Amir Pnueli & Lenore D. Zuck (1985): *The Glory of the Past*. In: *Logics of Programs*, pp. 196–218, doi:10.1007/3-540-15648-8_16.

[45] Andreas Morgenstern & Klaus Schneider (2008): *From LTL to Symbolically Represented Deterministic Automata*. In Francesco Logozzo, Doron A. Peled & Lenore D. Zuck, editors: *VMCAI*, pp. 279–293, doi:10.1007/978-3-540-78163-9_24.

[46] Maja Pesic, Helen Schonenberg & Wil M. P. van der Aalst (2007): *DECLARE: Full Support for Loosely-Structured Processes*. In: *(EDOC*, pp. 287–300, doi:10.1109/EDOC.2007.14.

[47] Jean-Eric Pin (1987): *On the Language Accepted by Finite Reversible Automata*. In Thomas Ottmann, editor: *ICALP*, pp. 237–249, doi:10.1007/3-540-18088-5_19.

[48] Amir Pnueli (1977): *The temporal logic of programs*. pp. 46–57, doi:10.1109/SFCS.1977.32.

[49] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *POPL*, pp. 179–190, doi:10.1145/75277.75293.

[50] M.O. Rabin & D. Scott (1959): *Finite automata and their decision problems*. *IBM Journal of Research and Development* 3, pp. 115–125, doi:10.1147/rd.32.0114.

[51] Fabio Somenzi (2016): *CUDD: CU Decision Diagram Package 3.0.0. Universiy of Colorado at Boulder*.

[52] Lucas Martinelli Tabajara & Moshe Y. Vardi (2019): *Partitioning Techniques in LTL$_f$ Synthesis*. In: *IJCAI*, pp. 5599–5606, doi:10.24963/ijcai.2019/777.

[53] Deian Tabakov, Kristin Y. Rozier & Moshe Y. Vardi (2012): *Optimized temporal monitors for SystemC*. *Formal Methods in System Design* 41(3), pp. 236–268, doi:10.1007/s10703-011-0139-8.

[54] Deian Tabakov & Moshe Y. Vardi (2005): *Experimental Evaluation of Classical Automata Constructions*. In: *LPAR*, pp. 396–411, doi:10.1007/11591191_28.

[55] Andrew M. Wells, Morteza Lahijanian, Lydia E. Kavraki & Moshe Y. Vardi (2020): *LTL$_f$ Synthesis on Probabilistic Systems*. In Jean-François Raskin & Davide Bresolin, editors: *GandALF*, *EPTCS* 326, pp. 166–181, doi:10.4204/EPTCS.326.11.

[56] Yaqi Xie, Fan Zhou & Harold Soh (2021): *Embedding Symbolic Temporal Knowledge into Deep Sequential Models*. *CoRR* abs/2101.11981.

[57] Shufang Zhu, Giuseppe De Giacomo, Geguang Pu & Moshe Y. Vardi (2020): *LTL$_f$ Synthesis with Fairness and Stability Assumptions*. In: *AAAI*, pp. 3088–3095, doi:10.1609/aaai.v34i03.5704.

[58] Shufang Zhu, Geguang Pu & Moshe Y. Vardi (2019): *First-Order vs. Second-Order Encodings for LTL$_f$-to-Automata Translation*. In: *TAMC*, pp. 684–705, doi:10.1007/978-3-030-14812-6_43.

[59] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu & Moshe Y. Vardi (2017): *A Symbolic Approach to Safety LTL Synthesis*. In: *HVC*, pp. 147–162, doi:10.1007/978-3-319-70389-3_10.

[60] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu & Moshe Y. Vardi (2017): *Symbolic LTL$_f$ Synthesis*. In: *IJCAI*, pp. 1362–1369, doi:10.24963/ijcai.2017/189.