Partitioning Techniques in LTL$_f$ Synthesis

Lucas Martinelli Tabajara and **Moshe Y. Vardi**
Rice University
lucasmt@rice.edu, vardi@cs.rice.edu

Abstract

Decomposition is a general principle in computational thinking, aiming at decomposing a problem instance into easier subproblems. Indeed, decomposing a transition system into a partitioned transition relation was critical to scaling BDD-based model checking to large state spaces. Since then, it has become a standard technique for dealing with related problems, such as Boolean synthesis. More recently, partitioning has begun to be explored in the synthesis of reactive systems. LTL$_f$ synthesis, a finite-horizon version of reactive synthesis with applications in areas such as robotics, seems like a promising candidate for partitioning techniques. After all, the state of the art is based on a BDD-based symbolic algorithm similar to those from model checking, and partitioning could be a potential solution to the current bottleneck of this approach, which is the construction of the state space.

In this work, however, we expose fundamental limitations of partitioning that hinder its effective application to symbolic LTL$_f$ synthesis. We not only provide evidence for this fact through an extensive experimental evaluation, but also perform an in-depth analysis to identify the reason for these results. We trace the issue to an overall increase in the size of the explored state space, caused by an inability of partitioning to fully exploit state-space minimization, which has a crucial effect on performance. We conclude that more specialized decomposition techniques are needed for LTL$_f$ synthesis which take into account the effects of minimization.

1 Introduction

Decomposing problems into smaller tasks and recombining them to find a solution is a fundamental tool of computational thinking. A notable example in formal verification is the idea of decomposing a transition system using a *partitioned transition relation*. In this approach, the system is represented by the product of individual components, and the transition relation of the entire system can be expressed by the set of individual transition relations. Partitioning was critical to scaling BDD-based model checking to large state spaces [Burch *et al.*, 1991], by producing a more compact and efficient representation for the transition relation of the system. Thanks to the success in model checking, partitioning has become a standard technique in symbolic and BDD-based algorithms, and was applied also to related problems such as symbolic Boolean satisfiability [Pan and Vardi, 2005] and Boolean synthesis [John *et al.*, 2015; Tabajara and Vardi, 2017].

A problem that might seem to be a promising candidate for partitioning techniques is that of LTL$_f$ synthesis [De Giacomo and Vardi, 2015], an adaptation to finite-horizon semantics of the classic problem of synthesizing reactive systems from specifications in Linear Temporal Logic (LTL), which has promising applications in areas such as robotics [He *et al.*, 2017]. The current state of the art for LTL$_f$ synthesis is based on a reduction to a reachability game played on a deterministic finite automaton (DFA), which is solved using a BDD-based symbolic fixpoint algorithm [Zhu *et al.*, 2017], similar to those that benefited from partitioning in model checking. Furthermore, the main limiting factor of current approaches is the construction of the DFA, which can often be very large and in the worst case doubly-exponential in the size of the formula [Kupferman and Vardi, 2001]. This problem is amplified by the fact that state-of-the-art techniques use an explicit construction of the automaton state space, only later encoding it symbolically. Partitioning would allow the DFA to be represented as a product of smaller component DFAs that can be constructed much more efficiently than a single monolithic DFA, thus eliminating the bottleneck of DFA construction.

In fact, previous efforts have already been made to apply similar decomposition techniques to reactive synthesis frameworks. Decomposition has been used successfully in explicit-state (infinite-horizon) LTL synthesis by the tool STRIX [Meyer *et al.*, 2018], which won the 2018 reactive synthesis competition (SYNTCOMP). Decomposition has also been employed in [Camacho *et al.*, 2018], where LTL$_f$ synthesis is reduced to FOND planning, though their results show that even using decomposition the FOND-planning approach does not always perform more efficiently than the standard BDD-based algorithm. A natural question to ask, then, is how such techniques perform when integrated into the BDD-based symbolic algorithm. It would be natural to assume that partitioning could be applied to symbolic LTL$_f$ synthesis to reap similar improvements as the ones obtained in model checking and related problems.

In this work, however, we expose fundamental limitations of partitioning that hinder its application in symbolic LTL$_f$ synthesis. An extensive experimental evaluation shows that, although decomposing the LTL$_f$ formula avoids an initial overhead during DFA construction, this comes at a significant expense for computing the game's winning strategy. We follow this evaluation with an in-depth analysis of the algorithm's execution that traces the issue to an overall increase in the size of the explored state space. We are able to attribute this increase to the partitioned representation being unable to fully exploit state-space minimization when constructing the DFA. Although at the initial phases of the fixpoint computation the partitioned transition relation is able to provide a compact representation that mitigates this issue, as the fixpoint computation progresses BDD sizes increase significantly in relation to the monolithic version. Our analysis indicates that this increase cannot be contained by simply using better heuristics for processing the partitioned transition relation.

These results suggest that, while DFA minimization is expensive, it is crucial to the success of LTL$_f$-synthesis algorithms. Therefore, for a decomposition strategy to be effective for this problem, it must interact well with minimization, a property that partitioning lacks. The conclusion that we draw is that, while partitioning is an effective technique for symbolic model checking and other related problems, LTL$_f$ synthesis may require a different way of thinking and perhaps more powerful decomposition approaches.

2 Preliminaries

2.1 LTL over Finite Traces (LTL$_f$)

The syntax of LTL$_f$ is identical to that of LTL over infinite traces. We define it over a set of propositions P as follows:

$$\varphi ::= \top \mid \bot \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

We also define in the usual way additional temporal operators $\mathbf{F}\varphi \equiv \top\mathbf{U}\varphi$ and $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$.

LTL$_f$ differs from LTL in its semantics, being interpreted over finite rather than infinite traces. A *(finite) trace* of length m is a sequence $\rho = \rho[0], \rho[1], \ldots, \rho[m-1] \in (2^P)^*$, where each $\rho[i]$ is an *interpretation* or *assignment* of the propositions in P, represented by the subset of the propositions that are true in instant i. In contrast, an *infinite trace* is a sequence $\rho = \rho[0], \rho[1], \ldots \in (2^P)^\omega$ of infinite length. The semantics of LTL$_f$ are defined inductively through a relation $\rho, i \models \varphi$, which denotes that finite trace ρ satisfies φ at instant i. See [De Giacomo and Vardi, 2013] for the formal definition of $\rho, i \models \varphi$. We say that a trace ρ satisfies formula φ, denoted by $\rho \models \varphi$, if $\rho, 0 \models \varphi$.

Definition 1 (LTL$_f$ Synthesis). *Let φ be an LTL$_f$ formula over set of propositions $P = \mathcal{X} \cup \mathcal{Y}$, for disjoint sets $\mathcal{X}$ and $\mathcal{Y}$. $\mathcal{X}$ is the set of* input variables *and $\mathcal{Y}$ is the set of* output variables. *φ is* realizable *if there exists a function $\gamma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ from histories of input variables to output variables such that, for every infinite sequence $X_0, X_1, \ldots \in (2^{\mathcal{X}})^\omega$, there is $m \geq 0$ such that $\rho = (X_0, \gamma(X_0)), (X_1, \gamma(X_0, X_1)), \ldots, (X_m, \gamma(X_0, \ldots, X_m))$ satisfies φ. The problem of LTL$_f$ synthesis is, given φ, to decide if φ is realizable, and if so to construct such a γ.*

The function γ works as a model of a system whose traces always satisfy φ. The output of the system in the current instant depends on the entire history of the inputs received so far. It is worth noting that in this work we consider that the system is able to take the current input into account when choosing an output, rather than only past inputs. This accounts for the slight difference in the definition of γ from [Zhu *et al.*, 2017]. We choose this formulation first because it is more natural for specifying the benchmarks in Section 4, and second because in the symbolic algorithm (see Section 3) the order of quantifiers is more amenable to partitioning. Yet, our results will show that even using this formulation LTL$_f$ synthesis is not able to benefit from partitioning.

2.2 DFA Game

An instance of LTL$_f$ synthesis can be solved by reduction to a game played over a Deterministic Finite Automaton (DFA). A DFA is a tuple $A = (S, \Sigma, \iota, \Delta, F)$, where:

- S is the set of states of the automaton.
- Σ is a finite alphabet.
- $\iota \in S$ is the initial state.
- $\Delta \subseteq S \times \Sigma \times S$ is the (deterministic) transition relation.
- $F \subseteq S$ is the set of accepting states.

The *run* of a finite trace on a DFA A is the sequence of states visited by the trace when starting from the initial state. The fact that Δ is deterministic guarantees that the run of a trace is unique. A run is *accepting* if the last state is in F. The *language* of A, denoted $\mathcal{L}(A)$, is the set of traces for which the corresponding run is accepting.

The *product* of DFAs $A_1, \ldots, A_k$ over the same alphabet Σ is the DFA $A_1 \times \ldots \times A_k = (S_1 \times \ldots \times S_k, \Sigma, (\iota_1, \ldots, \iota_k), \Delta, F_1 \times \ldots \times F_k)$, where $((s_1, \ldots, s_k), \sigma, (s'_1, \ldots, s'_k)) \in \Delta$ if and only if $(s_1, \sigma, s'_1) \in \Delta_i$ for all i. It is not hard to see that $\mathcal{L}(A_1 \times \ldots \times A_k) = \mathcal{L}(A_1) \cap \ldots \cap \mathcal{L}(A_k)$.

Every LTL$_f$ formula φ can be translated into a DFA A over alphabet $\Sigma = 2^P$ such that $\rho \in \mathcal{L}(A)$ if and only if $\rho \models \varphi$ [De Giacomo and Vardi, 2015].

A *DFA game* is a reachability game played using a DFA as the arena. The two players of the game represent the system and the environment. The game starts at the initial state of the DFA, and at every round the environment chooses an assignment to the $\mathcal{X}$ variables, while the system chooses an assignment to the $\mathcal{Y}$ variables. The combined assignment determines the state the game moves to. The system wins the game if it eventually reaches an accepting state of the DFA. We say that a state is a *winning state* if from that state the system can always make choices that will cause the game to reach an accepting state.

A *strategy* for the system is represented by a transducer (specifically, a Mealy machine) $T = (Q, 2^{\mathcal{X}}, 2^{\mathcal{Y}}, \tau, \nu)$ with state space Q, input and output alphabets $2^{\mathcal{X}}$ and $2^{\mathcal{Y}}$, initial state τ and transition function $\nu : Q \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}} \times Q$. The state of the transducer stores information about the history of the game so far, with the transition function serving to choose what output to produce and how to update its state. A strategy is *winning* for the system if, starting from the initial state of the DFA, a system that follows the strategy wins regardless of the choices of the environment. A winning strategy for the DFA game implicitly defines a solution to the LTL$_f$ synthesis problem for the original formula φ [Zhu *et al.*, 2017].

3 Partitioned LTL$_f$ Synthesis

Integrating partitioning into LTL$_f$ synthesis requires having a way to construct a partitioned transition relation from an LTL$_f$ formula. This construction should also be more efficient than constructing the monolithic DFA. The fact that most specifications take the form of a conjunction of constraints gives a natural way to decompose the formula: translate each conjunct individually into a DFA, then take the transition relation of all DFAs together as a partitioned transition relation for their product. This approach, which we detail below, follows the same general idea employed for decomposing specifications in [Camacho *et al.*, 2018] and [Meyer *et al.*, 2018].

3.1 DFA of a Conjunctive Formula

Let $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_k$ be an LTL$_f$ formula over input variables $\mathcal{X}$ and output variables $\mathcal{Y}$. φ can be translated into a DFA $A = (S, 2^{\mathcal{X}\cup\mathcal{Y}}, \iota, \Delta, F)$ that accepts the language of traces that satisfy φ. We call this a *monolithic* DFA for φ. This is the DFA that is originally used in [Zhu *et al.*, 2017]. Alternatively, each conjunct φ_i can be individually translated to a DFA $A_i = (S_i, 2^{\mathcal{X}\cup\mathcal{Y}}, \iota_i, \Delta_i, F_i)$ that accepts the language of traces that satisfy φ_i. Although φ_i might depend on only a subset of the variables, for simplicity we interpret it as a formula over $\mathcal{X} \cup \mathcal{Y}$. Under this interpretation, $\mathcal{L}(A) = \mathcal{L}(A_1) \cap \ldots \cap \mathcal{L}(A_k) = \mathcal{L}(A_1 \times \ldots \times A_k)$. Therefore, $A_1 \times \ldots \times A_k$ is also a DFA for φ. We call this a *partitioned* DFA for φ. Rather than explicitly computing the product state space $S_1 \times \ldots \times S_k$, a partitioned DFA can be represented simply by its individual components $A_1, \ldots, A_k$.

In order to apply partitioned techniques, each component DFA A_i must be encoded symbolically. This can be done using the same symbolic DFA encoding of [Zhu *et al.*, 2017]:

- The state space S_i is encoded as a set of state variables $\mathcal{Z}_i$ such that $|\mathcal{Z}_i| = \log_2(|S_i|)$. We create a copy z'_i of each variable $z_i \in \mathcal{Z}_i$ to use as a next-state variable, and denote this set by $\mathcal{Z}'_i$.
- The initial state ι_i is represented by an assignment $I_i \in 2^{\mathcal{Z}_i}$ of the state variables.
- The transition relation Δ_i is represented by a boolean formula δ_i over $\mathcal{Z}_i$, $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}'_i$.
- The set of accepting states F_i is represented by a boolean formula f_i over $\mathcal{Z}_i$.

A symbolic representation of the product DFA can then be obtained from the representation of the individual components, by taking $\mathcal{Z} = \bigcup_{i=1}^{k} \mathcal{Z}_i$, $\mathcal{Z}' = \bigcup_{i=1}^{k} \mathcal{Z}'_i$, $I = I_1 \cup \ldots \cup I_k$, $\delta = \delta_1 \wedge \ldots \wedge \delta_k$ and $f = f_1 \wedge \ldots \wedge f_k$. As in [Zhu *et al.*, 2017], BDDs can be used to represent the boolean formulas. In the case of a partitioned DFA, however, rather than constructing a single BDD for the conjunctions δ and f, it is possible to instead construct one BDD for each individual component δ_i and f_i, which can be expected to be much smaller. Next, we describe how to adapt the symbolic algorithm of [Zhu *et al.*, 2017] to a partitioned representation.

3.2 Solving a DFA Game over a Partitioned DFA

The algorithm of [Zhu *et al.*, 2017] is based on a least fixpoint that expands at every iteration an under-approximation of the set W of winning states (for the system) of the DFA for φ. The initial under-approximation, W_0, is simply the set of accepting states. After that, every iteration expands W_j into a larger under-approximation W_{j+1}, until no more states can be added. At this point, we know that the current set is the entire set of winning states, and the system can win the game if and only if the initial state is in the set. It is also possible to stop as soon as the initial state is added to the set, since at this point we already know that the system can win from the initial state. When using a partitioned DFA as described in Section 3.1, W_i can be defined recursively as follows:

$$
\begin{aligned}
&W_0 = F_1 \times \ldots \times F_k \\
&W_{j+1} = W_j \cup \{(s_1, \ldots, s_k) \in S_1 \times \ldots \times S_k \mid \\
&\qquad \forall X \in 2^{\mathcal{X}}. \exists Y \in 2^{\mathcal{Y}}. \exists s'_1, \ldots, s'_k. \\
&\qquad \bigwedge_{i=1}^{k} ((s_i, X, Y, s'_i) \in \Delta_i) \wedge (s'_1, \ldots, s'_k) \in W_j\}
\end{aligned}
$$

Note that every iteration adds to W_{j+1} those states of the form $(s_1, \ldots, s_k)$ from which, for every input X, the system can choose an output Y that moves the game into a state in W_j (which is already known to be winning). When $W_{j+1} = W_j$, we know that no more winning states exist.

Just as the DFA is represented symbolically, so can the sets $W_0, W_1, \ldots$, as formulas $w_0, w_1, \ldots$ over the state variables $\mathcal{Z}$. The fixpoint algorithm above can then be implemented symbolically as follows, where for a formula α and a variable set $\mathcal{V} \in \{\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}'\}$ the notation $\alpha(\mathcal{V})$ defines formula α over the variables in $\mathcal{V}$:

$$w_0(\mathcal{Z}) = f_1(\mathcal{Z}) \wedge \ldots \wedge f_k(\mathcal{Z})$$

$$w_{j+1}(\mathcal{Z}) = w_j(\mathcal{Z}) \vee \forall \mathcal{X}. \exists \mathcal{Y}. \exists \mathcal{Z}'. \bigwedge_{i=1}^{k} \delta_i(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge w_j(\mathcal{Z}')$$

As mentioned in Section 3.1, each f_i and δ_i can be represented as a BDD, and similarly for w_j. BDDs implement all the common boolean operations used in the computation, such as $\wedge$, $\vee$, universal and existential quantification, and variable substitution. Furthermore, they are canonical representations of boolean functions, allowing logical equivalence to be performed in constant time. This is useful to perform the test $w_{j+1} \equiv w_j$ of whether the fixpoint has been reached.

To compute the intermediate BDD for the formula $\exists \mathcal{Y}. \exists \mathcal{Z}'. \bigwedge_{i=1}^{k} \delta_i(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge w_j(\mathcal{Z}')$, the straightforward option would be to simply take the conjunction of all the individual BDDs of each δ_i and w_j, and then existentially quantify the $\mathcal{Y}$ and $\mathcal{Z}'$ variables. This approach, however, would fail to take full advantage of the partitioned representation. When existentially quantifying a variable v over a conjunction of formulas, the quantifier need not be applied to those conjuncts where v does not appear. This allows some variables to be quantified early, before conjoining all the subformulas. For example, consider a variable $z' \in \mathcal{Z}'_k$. Then, z' only appears in δ_k and w_j. Therefore, $\exists z'. \bigwedge_{i=1}^{k} \delta_i(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge w_j(\mathcal{Z}') \equiv \bigwedge_{i=1}^{k-1} \delta_i(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge (\exists z'.\delta_k(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge w_j(\mathcal{Z}'))$. Quantification of output variables that only appear in some of the conjuncts can be pushed inside in a similar way. Although the final BDD is the same, when using early quantification the intermediate BDDs can be significantly smaller, as the existential quantification tends to reduce their size before their conjunction becomes too large. Several heuristics can be used to decide in which order to perform conjunctions and quantification [Geist and Beer, 1994; Pan and Vardi, 2005]. In Section 4, however, we show that using better heuristics does not improve the performance of partitioning in LTL$_f$ synthesis.

3.3 Computing a Winning Strategy

When existentially quantifying a variable v from a Boolean formula ψ, it is also possible to compute, as a side product, a function from an assignment of the free variables of ψ to an assignment of v that satisfies ψ whenever possible. This process is known, among other names, as Boolean synthesis [Fried *et al.*, 2016]. Then, when computing $\exists\mathcal{Y}.\exists\mathcal{Z}'. \bigwedge_{i=1}^{k} \delta_i(\mathcal{Z}, \mathcal{X}, \mathcal{Y}, \mathcal{Z}') \wedge w_j(\mathcal{Z}')$ at iteration $j+1$, we can also construct a Boolean function $g_{j+1} : 2^{\mathcal{Z}} \times 2^{\mathcal{X}} \to 2^{\mathcal{Y}} \times 2^{\mathcal{Z}'}$ representing the choice for output variables and next state at that iteration. If the game is solved in m iterations, then the collection of functions $g_1, \ldots, g_m$ encodes a winning strategy. If the first move is chosen according to g_m, the second according to g_{m-1}, and so on, then the move chosen according to g_1 is guaranteed to reach an accepting state of the DFA. As can be seen in [Tabajara and Vardi, 2017], Boolean synthesis can be performed even if $\exists Y.\exists Z'. \bigwedge_{i=1}^{k} \delta_i(Z, X, Y, Z') \wedge w_j(Z')$ is computed using early quantification, as we propose in Section 3.2. Therefore, we can compute a winning strategy while still taking advantage of the benefits of a partitioned transition relation.

4 Experimental Evaluation

After describing how partitioning can be integrated into symbolic LTL$_f$ synthesis, we proceed to perform an experimental evaluation to point out the limitations of partitioning in this context[1]. We compare the performance of the partitioned approach with the original monolithic approach and show that partitioning introduces a significant overhead into the computation of the winning strategy. We follow these results with a thorough analysis that identifies the source of the overhead to be an enlargement of the DFA game's state space, and explain the reason for this phenomenon.

Following standard practice in LTL$_f$-synthesis literature [Zhu *et al.*, 2017; Camacho *et al.*, 2018], we convert LTL$_f$ to first-order logic and use the tool MONA [Henriksen *et al.*, 1995] to translate FOL formulas to DFAs. For the partitioned version of the algorithm, we decompose the LTL$_f$ specification into its top-level conjuncts and convert them individually to FOL, and then to DFAs using MONA. All measured times are end-to-end, including translation to FOL, construction of the DFA and computation of the winning strategy.

It is important to note that MONA produces canonical DFAs, meaning that they are the DFAs of minimum size for their formulas. In the monolithic approach, the monolithic DFA is minimized, while in the partitioned approach each component DFA is minimized. Because of this, even though the partitioned representation makes the size of the transition relation smaller, the number of state variables in the partitioned approach will be larger. As will be seen later in this section, this fact turns out to have a large impact on the performance of the partitioned approach.

All experiments were performed on a cluster consisting of 2304 processor cores in 192 Westmere nodes (12 processor cores per node) at 2.83 GHz with 48 GB of RAM per node (4 GB per core). Although we partition the DFA game, the partitioned algorithm is not easily parallelizable because every iteration of the fixpoint consists of sequentially combining components into a monolithic BDD. Therefore, the different cores were used only for running multiple experiments in parallel. Every instance had a timeout of 8 hours.

4.1 Evaluation on LTL$_f$ Benchmarks

As a fairly recent problem, LTL$_f$ synthesis lacks the extensive sets of benchmarks that exist for LTL synthesis. This forces us to construct our own set of benchmarks to use in our experimental evaluation. We initially considered benchmarks in the style of the ones used in [Zhu *et al.*, 2017] and [Camacho *et al.*, 2018], formed by random conjunctions of smaller base cases taken from the *lilydemo* LTL benchmark suite from [Jobstmann and Bloem, 2006] interpreted with LTL$_f$ semantics. In these benchmarks, the partitioned approach did outperform the monolithic one. The DFA games produced by these benchmarks, however, were very easy to solve, having very shallow winning strategies. All realizable instances could be won in at most 2 moves by the system. This is likely due partly to their random nature and partly to the fact that the base cases were formulas designed for infinite-horizon LTL, which can often become trivial when interpreted with finite-horizon semantics. We therefore conclude that these benchmarks are not an appropriate basis of comparison for LTL$_f$ synthesis, and proceed to construct more suitable benchmark families by encoding finite-horizon games in LTL$_f$.

Each of the three families below can be scaled based on one or more parameters which, unlike the case of random conjunctions, also increase the number of moves to win the game. The first two families deal with counters, which are important non-trivial components of many synthesis problems. Counters can model, for example, the spatial position of robots in a grid, or the amount of resources available to complete a task. The third family is based on a real game that has been extensively studied and requires complex strategic reasoning.

[1] Supplemental data in https://bitbucket.org/ijcai2816/ijcai-2816/

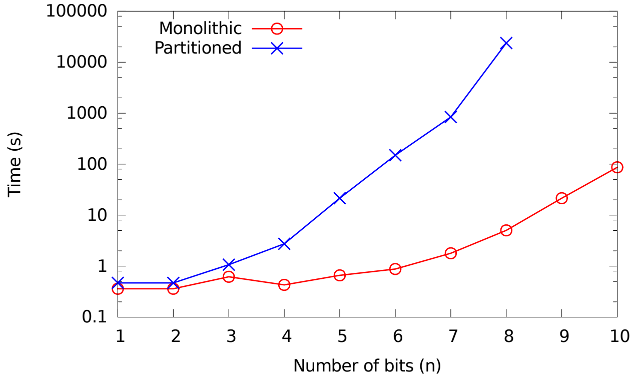


Figure 1: Running time of Single-Counter benchmarks. Plots are in log scale. Missing points mean that the instance either timed out or ran out of memory. The results show that running time increases significantly when partitioning is used.

	Monolithic	Partitioned
$n = 1, m = 1$	0.09	30.7
$n = 1, m = 2$	0.16	31.9
$n = 1, m = 3$	0.32	326.37
$n = 1, m = 4$	0.06	-
$n = 2, m = 1$	0.1	-
$n = 2, m = 2$	0.15	-
$n = 2, m = 3$	0.32	-
$n = 2, m = 4$	0.63	-
$n = 3, m = 1$	0.1	-
$n = 3, m = 2$	0.59	-
$n = 3, m = 3$	2.51	-
$n = 3, m = 4$	6.62	-

Table 1: Running time, in seconds, of the Nim benchmarks for different values of n and m. Missing values for the partitioned version indicate that the instance either timed out or ran out of memory.

	Monolithic	Partitioned
$n = 1, m = 1$	4	67
$n = 1, m = 2$	7	107
$n = 1, m = 3$	10	176
$n = 1, m = 4$	10	445
$n = 2, m = 1$	10	200
$n = 2, m = 2$	14	3047
$n = 2, m = 3$	20	3782
$n = 2, m = 4$	16	15457
$n = 3, m = 1$	6	423
$n = 3, m = 2$	14	2394
$n = 3, m = 3$	15	42175
$n = 3, m = 4$	17	472323

Table 2: Number of nodes of the BDD for the winning states of the Nim benchmarks for different values of n and m. For easier visualization, we show only the initial size of the BDD. In the monolithic version BDD sizes remain relatively constant during the computation, reaching around 60 nodes at most, while in the partitioned version most instances fail after one or two iterations.

Single Counter. The first benchmark family is a simple example where the behavior of the system is completely determined by the actions of the environment. Therefore, the challenge in this family lies mostly in proving that the specification is realizable. The system stores an n-bit counter (where n is the scaling parameter) which it must increment upon a signal by the environment. The system wins if the counter eventually overflows to 0. To guarantee that the game is winning for the system, the specification assumes that the environment will send the increment signal at least once every two timesteps.

Double Counters. This family of benchmarks is similar to the previous one, except that in this case there are two n-bit counters, one incremented by the environment and another by the system. The goal of the system is for its counter to eventually catch up with the environment's counter. To guarantee that this is achievable, the specification assumes that the environment cannot increment its counter twice in a row.

Nim. This family of benchmarks describes a generalized version of the game of Nim [Bouton, 1901] with n heaps of m tokens each. The environment and the system take turns removing any number of tokens from one of the heaps, and the player who removes the last token loses.

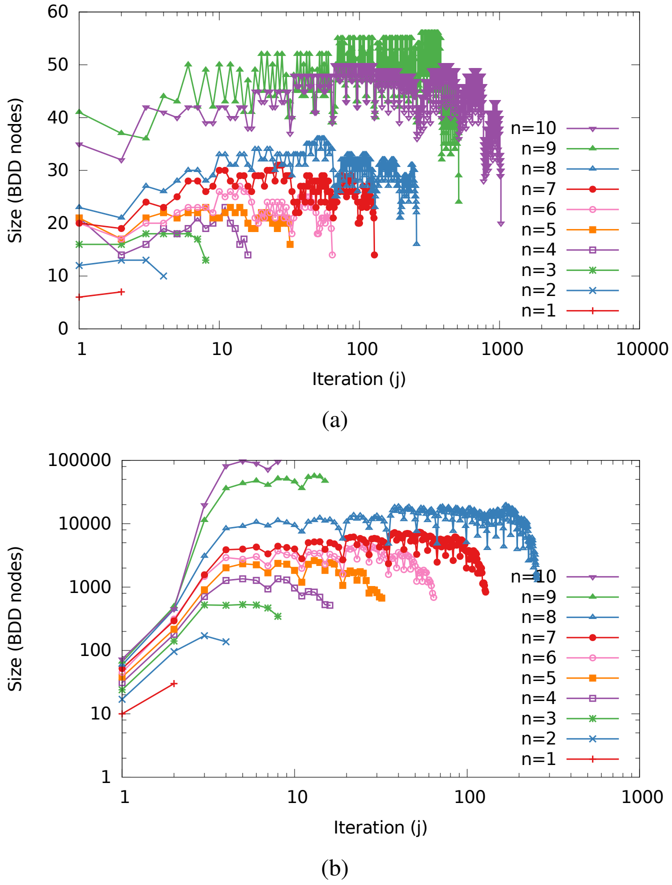
The LTL$_f$ encoding of the benchmarks above take the form of conjunctions of several subformulas describing the possible actions and goal of the system. These formulas were given as specifications, with increasing values of n and m, to the monolithic and partitioned versions of the algorithm.

We tried numerous different combinations of early-quantification heuristics, as well as variable ordering heuristics for the BDDs. For all of them, the same general results were obtained, showing a stark contrast with previous applications of partitioning. The best performance for both the partitioned and monolithic versions was achieved by using dynamic variable reordering for the BDDs. Figure 1 shows these results for the Single-Counter benchmarks when n varies from 1 to 10, using the Bucket-Elimination heuristic [McMahan *et al.*, 2004] for early quantification. Results for the Double-Counter benchmarks, as well as for other heuristics, follow a similar trend. Table 1 shows the same comparison for the Nim benchmarks.

In our experiments, unlike the case for model checking and other problems, symbolic LTL$_f$ synthesis was not able to benefit from partitioning. In fact, despite making DFA construction more feasible as expected (construction of the partitioned DFA always finished in under a second), partitioning created a massive overhead during computation of the winning strategy that lead to an orders-of-magnitude increase in running time. This overhead was large enough to nullify all benefits from the partitioned DFA construction, as in all cases where the monolithic version failed during construction, the partitioned version failed during the fixpoint computation. In the next section we provide an analysis of this phenomenon.

4.2 Analysis of the Results

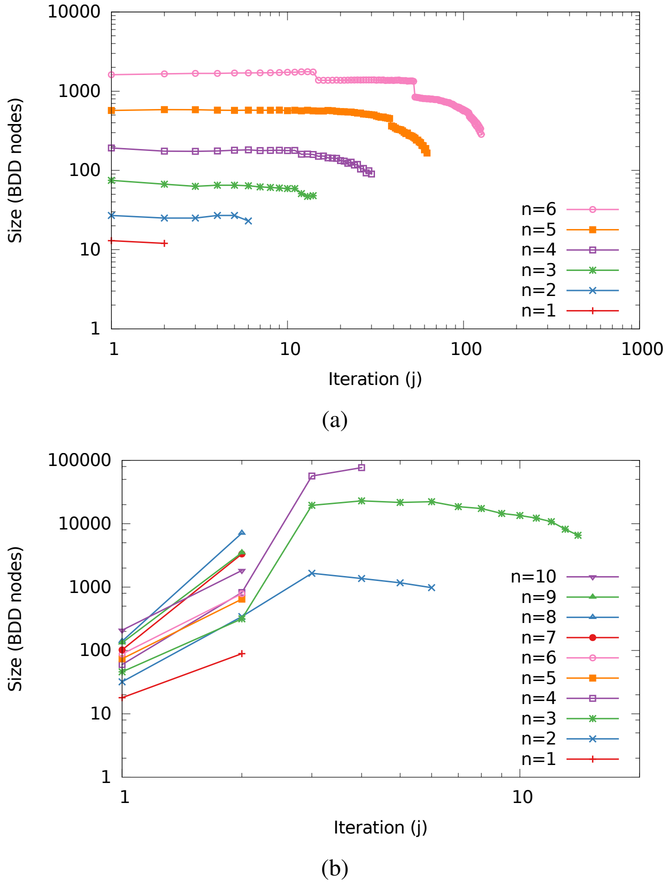
Explaining the disparity between the above results and previous applications of partitioning requires a more in-depth look into the behavior of the fixpoint algorithm in the mono-



(a)

(b)

Figure 2: Node count of the BDD for the winning states per iteration of the fixpoint algorithm on the Single-Counter benchmarks, using the (a) Monolithic or (b) Partitioned approach. For better visualization, one or both axes are in log scale in each of the plots.



(a)

(b)

Figure 3: Node count of the BDD for the winning states per iteration of the fixpoint algorithm on the Double-Counters benchmarks, using the (a) Monolithic or (b) Partitioned approach. Plots are in log-log scale. The monolithic DFA could not be constructed for $n > 6$.

lithic and partitioned cases. For this purpose, we proceeded to monitor the size of the BDD w_j representing the set of winning states across iterations of the fixpoint. Figure 2 displays these measurements for the Single-Counter benchmark family, again using dynamic variable reordering and Bucket Elimination, with every line representing a different value for the number of bits n. Figure 3 and Table 2 show analogous results for the Double-Counters and Nim families.

It is clear from the plots that there is a notable difference between the behavior of the monolithic and partitioned approaches. In the monolithic version, the BDD sizes change very little during the fixpoint computation (the changes are even smaller when dynamic variable reordering is not used). In the partitioned version, however, although the BDD sizes are initially small, they increase rapidly in the first few iterations of the fixpoint computation. By the time it becomes relatively constant, BDD size is already several orders of magnitude larger than in the monolithic version.

Interestingly, in the case of the Single-Counter benchmarks, even though the state space is exponential, the initial BDD sizes of the monolithic version increase linearly with n, indicating that a good symbolic encoding is achieved. This is not the case, for example, in the Double-Counter benchmarks, where the initial BDD sizes are exponential in n for the monolithic version, while being linear for the partitioned version. Even in the Double-Counter case, however, the BDDs in the partitioned version quickly grow exponentially to surpass the ones in the monolithic version.

These measurements of BDD growth allow us to conclude that the issue is not caused by inefficiencies in processing the partitioned transition relation during each iteration, but rather by a massive increase in BDD sizes during the course of the fixpoint computation. An important conclusion that can be drawn from this observation is that better early-quantification schemes are not enough to solve this problem: since BDDs are canonical data structures, the BDD computed at each iteration will be the same regardless of the order in which conjunctions and quantification are performed. At the same time, this observation also helps explain why the partitioned approach performed well on the random conjunctions mentioned in the beginning of Section 4.1, since those benchmarks can be solved in only one or two iterations. If the game can be solved in a single iteration, for example, then it is essentially equivalent to an instance of Boolean synthesis, where partitioning is already known to work [Tabajara and Vardi, 2017].

The blowup in BDD sizes for the partitioned version can be explained by the fact that, although the partitioned transition relation provides a more compact representation of a DFA game, the underlying state space can be significantly larger than for the monolithic version, leading to a major increase in the representation of the set of winning states. For example, in the Single-Counter benchmarks the number of state variables is in the order of n for the monolithic DFA, and $7n$ for the partitioned DFA. The large difference in size of the state spaces is due to the fact that MONA constructs canonical DFAs, and therefore the state space of the monolithic DFA is as small as possible. In the partitioned version, while the individual DFAs are minimized, the full state space of their product can be much larger than the canonical DFA, to the point that not even the symbolic representation can handle.

A reasonable question to ask is if it would be possible to find a balance between minimization and partitioning by constructing each DFA from the conjunction of multiple subformulas, rather than a single subformula. In this case, the individual DFAs might be more complex, but MONA would be able to perform more aggressive minimization. We tried both selecting which formulas to combine manually and using clustering heuristics from previous applications of partitioning such as [Pan and Vardi, 2005]. In either case, the running time and BDD sizes were smaller than in the fully-partitioned version, but still larger than the monolithic version, suggesting that performance decreases proportionally to the degree of partitioning.

The results of our experiments reveal critical issues that prevent the effective application of classic partitioned approaches to LTL$_f$ synthesis. Although such approaches might seem to be a promising solution to the current bottleneck of synthesis algorithms, i.e. DFA construction, this benefit is invalidated by the increased complexity of the resulting game.

5 Discussion

Ultimately, our results show that, despite having seen widespread use in related problems, partitioning has crucial weaknesses when it comes to LTL$_f$ synthesis. Partitioning does not interact effectively with DFA construction and minimization, giving rise to a compact representation of the transition relation but an enlarged representation of state sets.

Although our evaluation encompasses only a few benchmark families, these families illustrate common elements of synthesis problems, such as counters and strategic reasoning. Therefore it is not unreasonable to believe that our results extend to other interesting cases. At the very least, we have shown that partitioning is not a simple general solution, as one might have expected. Its applicability, if any, is likely to be limited to specific cases and might require domain knowledge to partition the problem in a way that still allows some degree of minimization.

The role that minimization plays in LTL$_f$ synthesis accounts for the crucial difference with partitioning in model checking. In the latter we have no access to a canonical transition system in the same sense as in the former. Instead, the model is already provided with a fixed state space, and the transition relation is partitioned along the already-existing set of state variables. Therefore, unlike in LTL$_f$ synthesis, partitioning in model checking produces no overhead on the state-space representation.

Our evaluation focused on LTL$_f$ due to the attention that this variant has recently attracted in the field of AI, and we cannot answer conclusively yet whether similar approaches would be effective or not in synthesis algorithms for the standard, infinite-horizon LTL. LTL$_f$ does have the distinction, however, that DFAs can be fully minimized. This favors the monolithic approach, that can benefit more from minimization than the partitioned approach. Nevertheless, it would not be surprising if a similar state-space explosion occurred also in the infinite-horizon case, even if to a lesser degree due to the smaller role of minimization. Therefore, the results of our evaluation should be considered if one intends to apply similar partitioning techniques to LTL synthesis.

Note that the LTL-synthesis tool STRIX [Meyer *et al.*, 2018] mentioned in the introduction also uses a similar type of decomposition, with successful results. The main difference is that rather than using symbolic techniques, STRIX maintains an explicit representation of the state space throughout, allowing the algorithm to compute states of the product on-the-fly as they are needed. This means that the explored state space needs to grow only as much as necessary to compute the winning strategy, and therefore the full size of the state space matters less than in symbolic approaches. We cannot rely solely on explicit-state approaches for synthesis, however, since they will necessarily need to use exponential space when the winning strategy requires an exponential number of states, while symbolic techniques might be able to compute the same strategy in polynomial space.

Yet, our results indicate that to improve current symbolic LTL$_f$ synthesis approaches, it might be necessary to look beyond classic partitioning towards more sophisticated decomposition techniques. Alternative decomposition approaches to partitioning have been proposed for Boolean relations in the context of Boolean synthesis [Akshay *et al.*, 2017; Chakraborty *et al.*, 2018], and it is possible that some of these ideas can be leveraged to LTL$_f$ synthesis as well, although further research is necessary to determine exactly how. The work of [Fried *et al.*, 2018] has also explored several novel notions of decomposition, both of relations and of DFAs. Given that minimization has shown itself to have a crucial effect on the performance of LTL$_f$ synthesis, finding a decomposition approach that will display good results for this problem will require carefully considering how the decomposition can exploit minimization as much as possible, and the ultimate effect that it has on the explored state space.

Acknowledgements

Work partially supported by NSF Grant no. IIS-1830549, by the Brazilian agency CNPq through the Ciência Sem Fronteiras program, and by the Data Analysis and Visualization Cyberinfrastructure funded by NSF under grant OCI-0959097 and Rice University. The authors thank Dror Fried for his helpful comments and feedback on the paper.

References

[Akshay *et al.*, 2017] S. Akshay, Supratik Chakraborty, Ajith K. John, and Shetal Shah. Towards Parallel Boolean Functional Synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Proceedings, Part I*, pages 337–353, 2017.

[Bouton, 1901] Charles L Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3(1/4):35–39, 1901.

[Burch *et al.*, 1991] Jerry R. Burch, Edmund M. Clarke, and David E. Long. Representing Circuits More Efficiently in Symbolic Model Checking. In *Proceedings of the 28th Design Automation Conference*, pages 403–407, 1991.

[Camacho *et al.*, 2018] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Finite LTL Synthesis as Planning. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS 2018*, pages 29–38, 2018.

[Chakraborty *et al.*, 2018] Supratik Chakraborty, Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. Functional Synthesis via Input-Output Separation. In *2018 Formal Methods in Computer Aided Design, FMCAD 2018*, pages 1–9, 2018.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, pages 854–860, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on Finite Traces. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 1558–1564, 2015.

[Fried *et al.*, 2016] Dror Fried, Lucas M. Tabajara, and Moshe Y. Vardi. BDD-Based Boolean Functional Synthesis. In *Computer Aided Verification - 28th International Conference, CAV 2016, Proceedings, Part II*, pages 402–421, 2016.

[Fried *et al.*, 2018] Dror Fried, Axel Legay, Joël Ouaknine, and Moshe Y. Vardi. Sequential Relational Decomposition. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018*, pages 432–441, 2018.

[Geist and Beer, 1994] Daniel Geist and Ilan Beer. Efficient Model Checking by Automated Ordering of Transition Relation Partitions. In *Computer Aided Verification, 6th International Conference, CAV '94, Proceedings*, pages 299–310, 1994.

[He *et al.*, 2017] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. Reactive Synthesis for Finite Tasks under Resource Constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017*, pages 5326–5332, 2017.

[Henriksen *et al.*, 1995] Jesper G. Henriksen, Jakob L. Jensen, Michael E. Jørgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders Sandholm. Mona: Monadic Second-Order Logic in Practice. In *Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Proceedings*, pages 89–110, 1995.

[Jobstmann and Bloem, 2006] Barbara Jobstmann and Roderick Bloem. Optimizations for LTL Synthesis. In *Formal Methods in Computer-Aided Design, 6th International Conference, FMCAD 2006*, pages 117–124, 2006.

[John *et al.*, 2015] Ajith K. John, Shetal Shah, Supratik Chakraborty, Ashutosh Trivedi, and S. Akshay. Skolem Functions for Factored Formulas. In *Formal Methods in Computer-Aided Design, FMCAD 2015*, pages 73–80, 2015.

[Kupferman and Vardi, 2001] Orna Kupferman and Moshe Y. Vardi. Model Checking of Safety Properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[McMahan *et al.*, 2004] Benjamin J. McMahan, Guoqiang Pan, Patrick Porter, and Moshe Y. Vardi. Projection Pushing Revisited. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Proceedings*, pages 441–458, 2004.

[Meyer *et al.*, 2018] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit Reactive Synthesis Strikes Back! In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Proceedings, Part I*, pages 578–586, 2018.

[Pan and Vardi, 2005] Guoqiang Pan and Moshe Y. Vardi. Symbolic Techniques in Satisfiability Solving. *J. Autom. Reasoning*, 35(1-3):25–50, 2005.

[Tabajara and Vardi, 2017] Lucas M. Tabajara and Moshe Y. Vardi. Factored Boolean Functional Synthesis. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017*, pages 124–131, 2017.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTLf Synthesis. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 1362–1369, 2017.