LTLf Synthesis under Partial Observability: From Theory to Practice

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

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

LTL synthesis is the problem of synthesizing a reactive system from a formal specification in Linear Temporal Logic. The extension of allowing for partial observability, where the system does not have direct access to all relevant information about the environment, allows generalizing this problem to a wider set of real-world applications, but the difficulty of implementing such an extension in practice means that it has remained in the realm of theory. Recently, it has been demonstrated that restricting LTL synthesis to systems with finite executions by using LTL with finite-horizon semantics (LTL_f) allows for significantly simpler implementations in practice. With the conceptual simplicity of LTL_f, it becomes possible to explore extensions such as partial observability in practice for the first time. Previous work has analyzed the problem of LTL_f synthesis under partial observability theoretically and suggested two possible algorithms, one with 3EXPTIME and another with 2EXPTIME complexity. In this work, we first prove a complexity lower bound conjectured in earlier work. Then, we complement the theoretical analysis by showing how the two algorithms can be integrated in practice into an established framework for LTL_f synthesis. We furthermore identify a third, MSO-based, approach enabled by this framework. Our experimental evaluation reveals very different results from what the theory seems to suggest, with the 3EXPTIME algorithm often outperforming the 2EXPTIME approach. Furthermore, as long as it is able to overcome an initial memory bottleneck, the MSO-based approach can often outperforms the others.

1 Introduction

LTL synthesis [26] is the problem of automatically generating a reactive system from a high-level specification of its behavior described in Linear Temporal Logic (LTL) [25]. Since its introduction [26], this problem has become a prominent area of research in formal methods, with a number of LTL-synthesis tools [14, 1, 15, 24] being developed over the years despite the problem's 2EXPTIME-completeness and the fact that algorithms often rely on complex operations such as determinization of ω-automata [18] and parity-game solving [36]. A line of follow-up work has focused on extending and generalizing the problem, such as allowing for partial observability (incomplete information) [22]. Because this adds an additional layer of complexity over the already-complex LTL-synthesis algorithms, however, this extension has resisted practical implementation and therefore have mostly remained in the theoretical realm.

Recent times have seen interest in a finite-horizon variant of LTL called Linear Temporal Logic over finite traces (LTL_f) [9]. LTL_f can be used to describe reactive systems with finite executions, making it relevant for example in the area of robotics [20], and LTL_f synthesis is closely connected to planning in AI [7]. Despite having the same doubly-exponential complexity as the synthesis problem for LTL over infinite traces [10], the appeal of LTL_f synthesis is that it requires much simpler algorithms, which translates into better practical performance. LTL_f specifications can be translated into finite automata, which are much easier to determinize and minimize than ω-automata, and can then be synthesized by playing a reachability game over the state space of the automaton [10], rather than a more complex

J.-F. Raskin and D. Bresolin (Eds.): 11th International Symposium
on Games, Automata, Logics, and Formal Verification (GandALF'20).
EPTCS 326, 2020, pp. 1–17, doi:10.4204/EPTCS.326.1

type of game such as a parity game. Thus, practical tools for LTL_f synthesis have already started being developed [35, 6] which compare very favorably with existing LTL-synthesis tools. These successes suggest that extensions of the synthesis problem that have resisted implementation in the infinite setting might also now have the potential to be realized in the finite-horizon case.

The extension that this paper focuses on is *synthesis under partial observability*, also called *synthesis with incomplete information* [22]. This generalization introduces *unobservable inputs*, which are propositions on which the specification depends but whose value is unknown to the system. This variant of the problem can thus model scenarios in which the system does not have access to all relevant information about the environment at all times; for example, a robot that is only able to sense its local vicinity. The extension of LTL_f synthesis from full to partial observability was first investigated in [11], which presents two approaches for this problem: one based on the construction of a belief-states space and a projection-based approach. Although the belief-states approach leads to a 3EXPTIME complexity, the projection-based approach is 2EXPTIME, matching the complexity of both synthesis under full observability and planning under partial observability [28], both of which this problem generalizes.

The analysis in [11] was theoretical, but with the subsequent development of practical tools for LTL_f synthesis we can now investigate how practical concerns may affect these results. Although the theoretical analysis suggests a clear advantage to the projection-based approach, there have been examples, when dealing with automata, of worst-case exponential gaps not manifesting in practice. For example, NFA often become smaller when determinized and minimized, even though in the worst case the minimal DFA may be exponentially larger [32]. Thus, to understand how to best solve the problem of synthesis under partial observability it is necessary to observe the performance of the algorithms in practice.

This works offers the following three contributions, which complement the results of [11]: First, we prove the conjecture from [11] that synthesis under partial observability for NFA specifications is 2EXPTIME-hard (and therefore 2EXPTIME-complete). This result completes the landscape of theoretical complexity presented in that work, which had proved tight bounds for DFA, AFW, and logical specifications. Second, we investigate how the two approaches for LTL_f synthesis under partial observability discussed in [11] can be implemented in practice within the SYFT framework [35], which currently represents the state of the art in LTL_f synthesis. Benefiting from SYFT's use of a symbolic synthesis algorithm, we implement the two approaches symbolically, potentially avoiding an exponential memory blowup. We additionally propose a third, alternative approach for the problem that naturally emerges from SYFT's use of the tool MONA [21] to convert from monadic second-order logic (MSO) [5] to DFA.

Finally, we implement the three approaches within SYFT and evaluate their performance, thus complementing the theoretical analysis from [11] with an empirical evaluation. To the best of our knowledge, this is the first instance of algorithms for temporal synthesis under partial observability being implemented in practice and evaluated empirically. Our evaluation reveals that the story is more nuanced than the theoretical analysis would lead us to believe. While in terms of worst-case complexity there is an exponential gap between the belief-states and projection-based approaches, this gap does not necessarily appear in practice, and in fact the projection-based approach turns out to be in many cases outperformed by the belief-states approach due to the latter producing a more efficient symbolic representation. We also find that while the MSO-based approach leads to significantly larger automata initially and is more likely to run out of memory during automaton construction, if this hurdle is overcome, then synthesis tends to be more efficient than the other approaches. This suggests that the MSO approach may be a promising option for LTL_f synthesis under partial observability, and furthermore motivates improving automata-construction algorithms from MSO formulas.

2 Preliminaries

Linear Temporal Logic over Finite Traces Linear Temporal Logic over finite traces, i.e. LTL$_f$ [9] extends propositional logic with finite-horizon temporal operators. LTL$_f$ is a variant of Linear Temporal Logic, or LTL [25], with the difference that LTL$_f$ 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:

$$\varphi ::= \top \mid \bot \mid p \in \mathcal{P} \mid (\neg\varphi) \mid (\varphi_1 \wedge \varphi_2) \mid (X\varphi) \mid (\varphi_1 U \varphi_2)$$

$\top$ and $\bot$ represent *true* and *false* respectively. X for “Next” and U for “Until” are temporal operators. Other operators can be written in terms of those. A *trace* $\rho = \rho[0], \rho[1], \ldots$ is a sequence of propositional interpretations (sets) $\rho[i] \in 2^{\mathcal{P}}$. Intuitively, $\rho[i]$ is interpreted as the set of propositions which are *true* at instant i. Trace ρ is an *infinite* trace if $|\rho| = \infty$, denoted as $\rho \in (2^{\mathcal{P}})^{\omega}$; otherwise ρ is a *finite* trace, denoted as $\rho \in (2^{\mathcal{P}})^*$. We assume standard semantics from [9].

An LTL$_f$ formula can be represented by an automaton over finite words that accepts a trace if and only if that trace satisfies the formula. A *nondeterministic finite automaton* (NFA) is a tuple $A = (\Sigma, S, s_0, \delta, F)$, where Σ is the alphabet, S is the state space, $s_0 \in S$ is the initial state, $\delta : S \times \Sigma \to 2^S$ is the (nondeterministic) transition function and $F \subseteq S$ is the set of accepting states. If the transition function δ is such that $|\delta(s,\sigma)| = 1$ for all $s \in S$ and $\sigma \in \Sigma$, then we say that A is a *deterministic finite automaton* (DFA) and we simplify the signature of δ to $\delta : S \times \Sigma \to S$. In the case of finite automata obtained from an LTL$_f$ formula, the alphabet is comprised of interpretations to the propositions of the formula, i.e. $\Sigma = 2^{\mathcal{P}}$. In this case, it is often useful to represent the transition function symbolically using Binary Decision Diagrams (BDDs) [3] and similar data structures.

LTL$_f$ Synthesis The *full-observability* version of the problem of LTL$_f$ *synthesis* [10] is defined as follows:

Definition 1 (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 $f : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$, such that for an arbitrary infinite sequence $\pi = X_0, X_1, \ldots \in (2^{\mathcal{X}})^{\omega}$ of propositional interpretations over $\mathcal{X}$, we can find $k \geq 0$ such that the finite trace $\rho = (X_0 \cup f(\varepsilon)), (X_1 \cup f(X_0)), \ldots, (X_k \cup f(X_0, X_1, \ldots, X_{k-1}))$ satisfies φ.*

Intuitively, LTL$_f$ synthesis can be thought of as a game between two players: the *environment*, who controls the input variables, and the *system*, who controls the output variables. Solving the synthesis problem means synthesizing a strategy f for the system such that no matter how the environment behaves, the combined behavior trace of both players satisfy the logical specification φ [10].

In [10] the authors introduce an algorithm for LTL$_f$ synthesis based on a reduction to a DFA game. The current state of the art for solving this problem is based on a symbolic version of this algorithm, proposed in [35]. Refer to those papers for details. In [11] the authors extend the problem of LTL$_f$ synthesis to the setting of *partial observability*, where the system does not have complete information about what happens in the environment. This situation is modeled by partitioning the set of input propositions $\mathcal{X}$ into *Obs*, the set of *observable* propositions, and *Unobs*, the set of *unobservable* propositions. When deciding on an action, the system can only base its decision on the observable inputs. Therefore, this variant of the problem asks for a strategy $f : (2^{Obs})^* \to 2^{\mathcal{Y}}$ such that for all infinite sequences $X_0, X_1, \ldots \in (2^{\mathcal{X}})^{\omega}$, a finite trace $(X_0 \cup f(\varepsilon)), (X_1 \cup f(X_0|_{Obs}), \ldots, (X_k \cup f(X_0|_{Obs}, X_1|_{Obs}, \ldots, X_{k-1}|_{Obs}))$ satisfies the specification, for some $k \geq 0$. Both the full- and partial-observability versions are 2EXPTIME-complete [10, 11].

3 Partial Observability for NFA Specifications

In addition to proving 2EXPTIME-completeness of LTL_f synthesis under partial observability, [11] also analyzed the complexity of the problem when starting already from automaton specifications. The problem was proved to be EXPTIME-complete from a DFA specification, and 2EXPTIME-complete from a specification given as a alternating finite-word automaton (AFW). For NFA specifications, the problem was shown to be in 2EXPTIME, but no lower bound was proved, although the authors conjectured that it was 2EXPTIME-complete. In this section we present a sketch of a proof that this conjecture is correct, and synthesis under partial observability from NFA specifications is indeed 2EXPTIME-complete. We prove the lower bound by simulating an alternating Turing machine that uses at most exponential space. As it is known that $AEXPSPACE = 2EXPTIME$ [8], this proves that the problem is 2EXPTIME-complete. The reduction uses a technique of modeling configurations of the Turing machine using the alphabet of the automaton (see [33, 30, 29]). For ease of exposition, we first describe a reduction to an NFA with polynomial number of states but with an exponential-size alphabet. Later we explain how to modify the reduction to use a polynomial alphabet.

Let $M = (Q, \Gamma, \delta, q_0, g)$ be an alternating Turing machine (ATM) [8] that requires space at most 2^{cn}, where n is the size of the input and c is a constant. Q is the set of states, Γ the tape alphabet, $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function, $q_0 \in Q$ is the initial state and $g : Q \to \{\forall, \exists, accept, reject\}$ indicates whether a state is universal, existential, accepting or rejecting. Transitions $(q', \gamma', d) \in \delta(q, \gamma)$ indicate the next state q' of the machine, the symbol γ' to write on the tape, and the direction d to move the head. Computations of an ATM can be seen as a game between a *universal* and an *existential* player. Which transition in $\delta(q, \gamma)$ is taken is chosen by the universal player if $g(q) = \forall$ and the existential player if $g(q) = \exists$. The machine accepts if the existential player has a strategy to reach an accepting state.

For simplicity, assume $\Gamma = \{0, 1, \#\}$, where # is the blank symbol. Let $x = x_1 \dots x_n \in \Gamma^*$ be an input string, which starts out on the tape. We construct an instance of the problem of NFA synthesis under partial observability that is realizable if and only if M accepts x. This instance is given by an NFA $N = (\Sigma, S, \Delta, s_0, F)$, with alphabet $\Sigma = Obs \times Unobs \times Out$, where Obs is the set of observable inputs, $Unobs$ is the set of unobservable inputs and Out is the set of outputs. Note that in this case Obs, $Unobs$ and Out are sets of symbols rather than of propositions, but if desired each can be encoded using a logarithmic number of propositions.

Simulating ATM Computations In the reduction, the environment and the system take the roles of universal and existential players, respectively. We define $Obs = \{1, \dots, m_\forall\}$, where $m_\forall$ is the highest branching factor of a universal state in M (i.e. $m_\forall = \max\{|\delta(q, \gamma)| \mid g(q) = \forall\}$). If the current state is universal, the environment player uses the observable inputs to choose a transition from $\Delta(q, \gamma)$. Likewise, $Out = \{1, \dots, m_\exists\} \times \{1, \dots, 2^{cn}\} \times (\Gamma \cup (Q \times \Gamma))$, where $m_\exists$ is similarly the highest branching factor of an existential state in M (i.e. $m_\exists = \max\{|\delta(q, \gamma)| \mid g(q) = \exists\}$). The first component of Out is similarly used by the system player to choose a transition from an existential state. The other two components are used to encode a cell (k, u), where k is a counter indicating which position of the tape the cell occupies and u is the contents of the cell, which are either a symbol γ or a tuple (q, γ) if the head of the machine is on that cell and on state q.

Once we have taken care of universal and existential branching, the idea of the reduction is that a trace of the NFA represents a sequence of configurations of the ATM. A configuration is given by a sequence of cells (k, u) of the form $(1, u_1), (2, u_2), \dots, (2^{cn}, u_{2^{cn}})$. After $k = 2^{cn}$, in the next step it should reset back to 1, indicating the start of a new configuration that should follow from the previous one according to the transition function δ. The NFA accepts if the trace reaches an accepting configuration.

The challenge of the reduction is to enforce that the configurations produced by the system player are consistent: that the counter k increases by 1 each time and resets after 2^{cn}, and that one configuration follows from the previous one, and the first configuration has x on the tape. If this is the case, then the trace of the NFA corresponds to an accepting computation of M. We cannot enforce this consistency just by storing information in the state, because this would require an exponential number of states. Instead, we use the unobservable inputs to constrain the actions of the system player.

Using Partial Observability and Nondeterminism We define $Unobs = \{1,\dots,cn\} \times \{0,1\}$. The first component $i \in \{1,\dots,cn\}$ is only used at the first step of the trace, and represents the choice of a bit k_i of the counter $k \in \{1,\dots,2^{cn}\}$ for the environment to monitor. At each step of the computation, the automaton will determine from the current value of k what the value of k_i should be at the next step, and store that in the state. If at any point the value of k_i differs from the expected, the NFA rejects. Note that, since the system player does not know which bit the environment has chosen to monitor, the only way to guarantee a win is to always keep the entire counter consistent.

The second component $p \in \{0,1\}$ is a flag that should be raised exactly twice during the computation, on two adjacent configurations (if the environment breaks this assumption, the NFA accepts). If on the two times that $p = 1$ the counter has the same value (i.e., p points to the same cell both times), then the contents of the cell on the second configuration must follow from the first configuration according to the transition relation (e.g., if the head was in that cell, it must have written the correct symbol and moved away, etc.). To check for that, N makes a nondeterministic guess the first time $p = 1$. If N guesses that p will point to a different cell, it guesses also which bit will be different between the two counters and stores that in the state. The second time $p = 1$, N checks that the bits are indeed different. If it guesses that p will point to the same cell, it stores in the state what the value of the cell should be in the next configuration, and checks that it is correct once $p = 1$ again. Similarly to the counter, since the system player does not know when p is raised, the only way to guarantee a win is to ensure that adjacent configurations follow from one another.

Polynomial State Space. Note that the states of N must keep track of the following information: which state $q \in Q$ the machine is in; what was the bit $i \in \{1,\dots,cn\}$ chosen by the environment in the first step of the trace; how many times p has been raised (0, 1, 2 or more); if p has been raised once, how long ago that was (this configuration, last configuration, earlier than that); if N has guessed that p will point to different cells, what is the index i and value k_i of the bit that will be different; if N has guessed that p will point to the same cell, what is the expected content of that cell in the next configuration; the contents of the previous cell on the tape, in case p is raised (the contents of a cell can be affected only by its adjacent cells). Since each component of the state is polynomial on M and x, the NFA has a polynomial number of states. Accepting states are those where $g(q) = accept$. For lack of space, we omit the details of the transition function.

Polynomial Alphabet. Note that the alphabet Σ of N is polynomial except for the counter k that forms the second component of Out. We can reduce the alphabet to polynomial size by encoding each cell (k,u) over multiple time steps as a sequence $k_1,\dots,k_{cn},u$, where $k_i \in \{0,1\}$ is the i-th bit of k. This requires splitting each state of the automaton into $cn+1$ states, and also keeping track of additional information in the state (necessary, for example, to compute the next value of the bit k_i being monitored by the environment). Yet, none of these changes make the state space larger than polynomial.

Therefore, the reduction from acceptance of an ATM to synthesis under partial observability from an NFA specification is polynomial.

Theorem 1. *Synthesis under partial observability from an NFA specification is 2EXPTIME-complete.*

4 Partial-Observability Synthesis in Practice

Two algorithms for LTL$_f$ synthesis under partial observability were proposed in [11]: a belief-states construction with worst-case 3EXPTIME complexity and a projection-based construction that achieves an optimal 2EXPTIME complexity. In this section we show how algorithms for synthesis under partial observability can be practically implemented within the context of existing tools for LTL$_f$ synthesis. We first review the SYFT framework [35], which represents the state-of-the-art for LTL$_f$ synthesis under full observability, combining an explicit automaton construction with symbolic BDD-based techniques for synthesizing the strategy efficiently. Then, we introduce novel versions of the two algorithms for partial observability that perform part of the automaton construction symbolically. This serves two purposes. First, it allows them to be easily integrated into SYFT's framework, as the symbolic automata can be passed directly to the symbolic strategy computation. Second, it avoids an explicit exponential blow-up in the automaton-construction step of the algorithms, as it avoids ever constructing the final automaton explicitly and instead directly constructs a symbolic representation. This representation tends to be much more compact and sometimes exponentially smaller. Finally, we describe a third, novel MSO-based approach that is made possible specifically by the DFA-construction approach employed by SYFT.

The SYFT Framework. SYFT's synthesis approach can be summarized as follows. First, translate the LTL$_f$ formula φ into a formula in first-order logic $fol(\varphi)$, using the procedure described in [9]. Then, use the tool MONA [21] to convert $fol(\varphi)$ into a minimal DFA A. Next, convert A into a symbolic-state representation over a set of state variables $\mathcal{Z}$, logarithmic in the number of states. Each state is implicitly encoded as an interpretation of the variables in $\mathcal{Z}$, and the transition relation and set of accepting states are then represented by BDDs. Finally, use a symbolic fixpoint algorithm to compute a winning strategy in the DFA game given by A. Details of each step can be found in [35].

4.1 Projection-Based Construction

We start by describing the second approach from [11], as the first approach can be seen as a special case of it. We can summarize this approach as follows: 1. construct an NFA $\bar{N}$ for $\neg\varphi$; 2. project unobservable inputs from $\bar{N}$'s transition function; 3. determinize $\bar{N}$ into a DFA $\bar{A}$; 4. complement $\bar{A}$ into A. After the second step, $\bar{N}$ accepts those traces that can be extended by a trace of unobservable inputs such that the result violates φ. By complementing the automaton we obtain a DFA game that can be won by the system iff φ can be realized under partial observability. This construction takes advantage of the fact that LTL$_f$ formulas are closed under negation, NFAs are closed under projection and DFAs are closed under complementation, and each of these operations can be performed in polynomial time. Therefore, the only exponential steps are the conversions from LTL$_f$ to NFA and NFA to DFA, making the entire construction doubly exponential.

The challenge in implementing this construction in the SYFT framework is that SYFT is based on MONA, which translates logical formulas to DFAs, while we need to first construct an NFA $\bar{N}$ for $\neg\varphi$. We do this in two steps. First, we construct a minimal DFA for the *reverse* of the language of $\neg\varphi$ (this DFA is guaranteed to be at most exponential in the size of the formula [8]). Then, we reverse this DFA by switching the initial and final states and reversing all transitions. The result is an NFA for the language of $\neg\varphi$, and this NFA is at most exponential. To construct the DFA for the reverse language, we employ a technique introduced in [34], which converts an LTL$_f$ formula into a Past LTL$_f$ formula for its reverse language, then converts this Past LTL$_f$ formula into first-order logic to give as input to MONA. Besides providing theoretical guarantees that the NFA constructed is exponential at most, this approach has also performed well in our preliminary experiments against alternative approaches for NFA construction, such as using the automaton package SPOT [13].

The next three steps, particularly the determinization step, may lead to an exponential blow-up in the automaton. To mitigate this problem, we describe how to perform these steps symbolically, so that we construct a symbolic representation of the DFA directly from the explicit representation of $\bar{N}$, without ever building the state space of the DFA explicitly. This can be done because the standard subset-construction approach for determinization lends itself naturally to being performed symbolically. Because the symbolic representation can be exponentially more compact, this construction might avoid an explicit exponential blowup. We now describe the symbolic construction.

The NFA $\bar{N} = (2^{\mathcal{P}}, S, \delta, s_0, F)$ is generated with transitions represented symbolically by a BDD $T_{i,j}$ for every pair of states s_i and s_j, such that $T_{i,j}$ evaluates to 1 under an interpretation $\sigma \in 2^{\mathcal{P}}$ iff $s_j \in \delta(s_i, \sigma)$. We project the unobservable propositions by simply applying a standard BDD operation of existential quantification to every $T_{i,j}$. To perform determinization symbolically, we create a state variable z_i for each state s_i of $\bar{N}$. Then, an interpretation Z to the state variables $\mathcal{Z}$ represents the subset that contains exactly those states for which the corresponding variable is *true*. The transition function is then represented by BDDs $\Delta_1, \dots, \Delta_{|\mathcal{Z}|}$, where $\Delta_j = \bigvee_{z_i \in \mathcal{Z}} (z_i \wedge \exists u_1, \dots, u_n . T_{i,j})$ for $Unobs = \{u_1, \dots, u_n\}$. Note that Δ_j evaluates to 1 iff z_j is in the successor subset according to subset construction. The accepting states (after complementation) are also represented by a BDD $\Phi = \neg \bigvee_{s_i \in F} z_i$, which evaluates to 1 for an interpretation Z if Z represents an accepting subset. Note that the existential quantification in Δ_j and the negation in Φ come respectively from the projection and complementation steps. This final symbolic DFA represented by the BDDs $\Delta_1, \dots, \Delta_{|\mathcal{Z}|}$ and Φ can then be given directly to the symbolic game-solving algorithm implemented in SYFT to compute a strategy.

4.2 Belief-State Construction

The belief-states approach described in [11] is based on a standard construction used in planning under partial observability [19, 2, 4, 23]. Given a DFA D for the LTL$_f$ formula φ, this approach constructs a new DFA B where the state space is formed of *belief states*, which are sets of states of D representing the possible states in which the game can be given the information observed by the system. Since B is exponential in D, and D is in the worst case doubly-exponential in φ, in the worst case this approach is triple-exponential.

As pointed out in [11], the belief-state construction is equivalent to starting the projection-based construction outlined in Section 4.1 from a DFA D (constructed normally by MONA) rather than an LTL$_f$ formula. In this case, rather than negating the formula, we simply complement D. Since a DFA is a special case of an NFA, the last three steps can be performed exactly in the same way as in the projection-based approach. Therefore, the belief-state construction can likewise be performed symbolically, potentially saving one exponential as well. Note that the subset construction used to determinize the NFA now constructs the belief states. The existential quantification in the definition of Δ_j can be interpreted as adding to the belief state every state s_j for which there is a possibility of the unobservable inputs having moved the automaton to s_j. Finally, note that since the set F of accepting states of D was complemented in the first step, the final BDD for the accepting states of A is $\Phi = \neg \bigvee_{s_i \notin F} z_i = \bigwedge_{s_i \notin F} \neg z_i$. This corresponds to the accepting belief-states being those that contain only accepting states of D, i.e., only those where the system can be sure that it is in an accepting state.

The fact that the DFA for an LTL$_f$ formula may be doubly-exponential, while a NFA is at most exponential, seems to reinforce the notion that the projection-based approach is strictly better. In practice, however, it has been observed that fully-minimized DFA (as is the case of the DFAs produced by MONA) are rarely doubly-exponential, and in some cases when NFA are determinized and minimized they actually become smaller [32]. Therefore, it is important to compare the two approaches empirically as well, which we do in Section 5.

4.3 MSO Construction

Although the above two approaches were the only ones presented in [11], the synthesis framework employed by SYFT naturally suggests a third approach for synthesis under partial observability. In the second step of SYFT's workflow, MONA is used to convert the first-order-logic formula $fol(\varphi)$ into a DFA. MONA, however, can handle not only first-order formulas, but also more general formulas in *monadic second-order logic* (MSO) [5]. MSO can easily model quantification over traces, allowing us to express in MSO the language of traces over $Obs \cup \mathcal{Y}$ such that for all traces over $Unobs$ the LTL$_f$ formula φ is satisfied. This language is represented simply by the formula $\forall U_1 . \ldots \forall U_n . fol(\varphi)$, where each U_i is a second-order variable corresponding to one of the unobservable propositions. Thus, by simply adding to SYFT's workflow the step of quantifying the unobservable propositions, we can get SYFT to solve the synthesis problem under partial observability. The following theorem follows directly from the MSO semantics and states the correctness of this approach.

Theorem 2. *A strategy for the DFA game specified by the MSO formula $\forall U_1 . \ldots \forall U_n . fol(\varphi)$ is winning for the system iff that strategy is a solution to the synthesis problem for φ under partial observability.*

Interestingly, the procedure used by MONA to construct the DFA for this MSO formula resembles the projection-based construction. MONA uses a syntactic approach for constructing DFAs, first rewriting $\forall U_1 . \ldots \forall U_n . fol(\varphi)$ as $\neg(\exists U_1 . \ldots \exists U_n . \neg fol(\varphi))$, then building a DFA for $fol(\varphi)$ and applying complementation and projection as appropriate. Thus, it follows the same sequence of steps outlined in Section 4.1. Note, however, that MONA not only starts with a DFA, like in the belief-states construction, but also determinizes the intermediate automata after every projection operation, since it does not have an internal representation for NFAs. This means that although the final DFA is minimal, and therefore doubly-exponential at most, it is possible that the subset-construction operation may add a third exponential to the running time. On the other hand, MONA minimizes intermediate DFAs after every operation, which can actually make them significantly smaller and may improve the running time in practice. Furthermore, because the final DFA output by MONA is fully minimized, the number of states may be much smaller than that of the final DFAs produced by the other procedures, for which subset construction is performed symbolically and therefore does not go through minimization. On the other hand, because the final DFA is not generated directly in symbolic representation, if the number of states is large the DFA construction is more likely to fail. Considering these points, it is not clear in general how this approach would compare with the others, and answering this question requires an experimental evaluation.

5 Experimental Evaluation

As mentioned in the previous section, theory is not necessarily a good indicator for performance in practice. There are a number of factors that are not factored into the theoretical analysis but can affect the performance of the three approaches described in Section 4, including the difference in practice of DFA vs. NFA size, the DFA-construction algorithm implemented by MONA, and the use of symbolic representation. Therefore, it is essential to complement the theory with an empirical evaluation in order to determine the relative advantage of each of the three approaches. We first present three families of benchmarks that we used in our evaluation, and then describe our experimental setup and results.

5.1 Benchmarks

To model settings where the system must behave strategically in the presence of partial observability, keeping track of information learned during interaction with the environment, we constructed LTL_f specifications describing games with incomplete information. We constructed three benchmark families for our evaluation. Below we provide a brief high-level description of each family, then present and explain the general form of the LTL_f specification in each case, indicating the observable inputs, unobservable inputs and outputs, as well as whether the specification is realizable or not and some intuition about the winning strategy if it is realizable. It is worth noting that the first two benchmark families are simpler in the sense that they use only the X, G and F operators, while the third family also uses the more general U operator. Nevertheless, we obtain the same general conclusions from all three of them.

5.1.1 Moving-Target

The environment controls a target moving along a line with n positions. The target's location and movement are unknown to the system, who at every turn tries to guess where the target is.

$$\varphi_{Target} = G(\texttt{exactly-one}(target_1,\ldots,target_n))$$
$$\varphi_{Move} = G(X\,true \rightarrow \texttt{move-left-or-right}(target_1,\ldots,target_n))$$
$$\varphi_{Hit} = \bigwedge_{i=1}^{n} G((target_i \wedge guess_i) \rightarrow hit) \qquad \varphi_{System} = G(\texttt{exactly-one}(guess_1,\ldots,guess_n)) \wedge F\,hit$$

Full specification: $(\varphi_{Target} \wedge \varphi_{Move} \wedge \varphi_{Hit}) \rightarrow \varphi_{System}$

$target_1,\ldots,target_n$ are unobservable input variables such that $target_i$ is true if the target is in position i of the line. Exactly one $target_i$ variable must be true at a given time (as specified in φ_{Target}). We omit details of the subformula $\texttt{move-left-or-right}(target_1,\ldots,target_n)$ in φ_{Move}, but it suffices to know that it establishes a relation between the values of $target_1,\ldots,target_n$ in adjacent time steps, namely that the target must always move to the position immediately to the left or to the right of the previous position (and cannot stay in the same position). If the target is in position 1 or n, then the only option is for it to move to position 2 or $n-1$, respectively. $guess_1,\ldots,guess_n$ are output variables such that $guess_i$ is set to true to guess that the target is in position i. hit is an observable input variable that is set to true if the guess is correct (as specified in φ_{Hit}). The system can only make one guess at a time, and it wins if it guesses correctly (φ_{System}). All specifications in this family are realizable regardless of the value of n. The winning strategy for the system player is based on two rules: first, if the target is not in position 2 at time t, then it cannot be in position 1 at time $t+1$ (same for $n-1$ and n); second, if the target is neither in position $i-1$ nor $i+1$ at time t, then it cannot be in position i at time $t+1$. Using these two rules, the system can guess in such a way that it narrows down the positions the target can be in over time, guaranteeing that it will hit the target eventually.

5.1.2 Coin-Game

This is an n-coin generalization of the game described in [12]. Every turn the system chooses a coin to flip, and wins once all coins are heads. The environment reports whether the coin was flipped to heads or tails, and can secretly swap the two coins adjacent to it.

$$\varphi_{Init} = \texttt{exactly-one}(\neg coin_1,\ldots,\neg coin_n) \qquad \varphi_{Valid} = X\,G(valid \leftrightarrow \texttt{exactly-one}(flip_1,\ldots,flip_n))$$

$$\varphi_{Update} = \bigwedge_{i=1}^{n} G(X\, valid \rightarrow \mathtt{update}_i(flip_1, \ldots, flip_n, coin_1, \ldots, coin_n, swap))$$

$$\varphi_{Heads} = X\, G\,(valid \rightarrow (heads \leftrightarrow \bigvee_{i=1}^{n} (flip_i \wedge coin_i))) \qquad \varphi_{System} = F \bigwedge_{i=0}^{n} coin_i$$

Full specification: $(\varphi_{Init} \wedge \varphi_{Valid} \wedge \varphi_{Heads} \wedge \varphi_{Update}) \rightarrow \varphi_{System}$

$coin_1, \ldots, coin_n$ are unobservable input variables such that $coin_i$ is true if the i-th coin is heads. Initially, exactly one coin is heads (φ_{Init}). $flip_1, \ldots, flip_n$ are output variables such that $flip_i$ is set to true to flip the i-th coin. The system may only flip a single coin, otherwise the observable input variable $valid$ is set to false (φ_{Valid}). The unobservable input variable $swap$ indicates whether the environment decides to swap the two coins adjacent to the coin flipped by the system. If the move is valid, the state of the coins is updated by the environment (φ_{Update}). The subformula $\mathtt{update}_i(flip_1, \ldots, flip_n, coin_1, \ldots, coin_n, swap)$ in φ_{Update} expresses how $coin_i$ variables are updated. We omit details, but intuitively when $flip_i$ is true $coin_i$ changes value, and if additionally $swap$ is true then $coin_{(i-1) \bmod n}$ and $coin_{(i+1) \bmod n}$ have their values swapped. If the move is valid the environment also reports using the observable input variable $heads$ whether the flipped coin was flipped to heads or not (φ_{Heads}). The system wins if all coins are flipped to heads (φ_{System}). The specification is unrealizable for $n = 3$ and realizable for all $n > 3$. The winning strategy for even n is simple: flip all even-numbered coins to heads, then flip all odd-numbered coins to heads. This prevents the environment from secretly swapping a flipped coin with an unflipped coin, since it is only able to swap coins that are adjacent to the last coin that was flipped. A similar strategy also works with some adjustment for odd n, except for $n = 3$, where flipping a coin always gives the environment the opportunity to swap the other two.

5.1.3 Private-Peek

This family of benchmarks is based on the game described in [27], which is an incomplete-information version of the *Peek* game from [31]. Players push plates with holes in and out of a box. Depending on their configuration they might uncover holes on the box so that one or the other player can peek through to the other side. The first player able to do so wins. Each player has n plates to control and m holes that they might be able to peek through, and only half of the plates (rounded up) of the environment are visible to the system. As the positions of the holes in each plate are arbitrary, we can generate multiple instances for each value of m and n by randomly selecting the hole positions.

$$\varphi^p_{In} = \bigwedge_{i=1}^{n} plate^p_i \qquad \varphi^p_{Wait} = \bigwedge_{i=1}^{n} G(X \neg turn^p \rightarrow (X\, plate^p_i \leftrightarrow plate^p_i))$$

$$\varphi^p_{Move} = \bigwedge_{i=1}^{n} G(X\, turn^p \rightarrow \mathtt{at\text{-}most\text{-}one}(X\, plate^p_1 \leftrightarrow \neg plate^p_i, \ldots, X\, plate^p_n \leftrightarrow \neg plate^p_n))$$

$$\varphi^p_{Peek} = \bigwedge_{j=1}^{m} G(peek^p_j \leftrightarrow \mathtt{random\text{-}cube}^p_j(plate^e_1, \ldots, plate^e_n, plate^s_1, \ldots, plate^s_n))$$

$$\varphi_{Turn} = \neg turn^e \wedge \neg turn^s \wedge X\, turn^s \wedge X\, G(turn^s \leftrightarrow \neg turn^e) \wedge X\, G(X\, true \rightarrow (X\, turn^s \leftrightarrow turn^e))$$

$$\varphi_{Goal} = \left(\bigwedge_{j=1}^{m} (turn^e \rightarrow \neg peek^e_j) \right) U \left(turn^s \wedge \bigvee_{j=1}^{m} peek^s_j \right)$$

Full specification: $(\varphi^e_{In} \wedge \varphi^e_{Wait} \wedge \varphi^e_{Move} \wedge \varphi^e_{Peek} \wedge \varphi^s_{Peek}) \rightarrow (\varphi_{Turn} \wedge \varphi^s_{In} \wedge \varphi^s_{Wait} \wedge \varphi^s_{Move} \wedge \varphi_{Goal})$

The game alternates between system and environment turns. Output variables $turn^s$ and $turn^e$ are used to keep track of turns, and are set to true when it is the system's and the environment's turn, respectively. In the first timestep, which serves just to set up the initial state of the game, both $turn^s$ and $turn^e$ are set to false. The first turn of the system occurs in the second timestep, and turns alternate after that (φ_{Turn}). Variables $plate_1^p, \ldots, plate_n^p$ are input variables for $p = e$ and output variables for $p = s$, and $plate_i^p$ is true if the i-th plate of player p is in, and false if it is out. Initially, all plates are in (φ_{In}^p), and on their own turn each player can choose to slide at most one of them in or out (φ_{Move}^p). On the opponent's turn the player cannot move their plates (φ_{Wait}^p).

$peek_1^p, \ldots, peek_m^p$ are input variables such that $peek_j^p$ is true if player $p \in \{e,s\}$ can peek through their j-th hole to the other side. The system wins if it is able to peek through one of its holes before the environment can (φ^{Goal}). The set of configurations that determines whether the j-th hole of player $p \in \{e,s\}$ is uncovered is encoded in φ_{Peek}^p by the formula $\texttt{random-cube}_j^p(plate_1^e, \ldots, plate_n^e, plate_1^s, \ldots, plate_n^s)$, which is generated randomly by selecting a subset of the variables $plate_1^e, \ldots, plate_n^e, plate_1^s, \ldots, plate_n^s$, each in either positive of negative form, and taking their conjunction. This encoding is justified by the equivalence of *Peek* to a formula game played over a formula in disjunctive normal form [31, 27]. Intuitively, if $plate_i^p$ appears in positive (respectively, negative) form it means that the i-th plate of player p needs to be in (out) to uncover the hole. If $plate_i^p$ does not appear at all, then either configuration works. In this way, we can generate multiple random instances of the *Private-Peek* family for each m and n. For our experiments, each variable has a $1/2$ chance of being selected for a given random cube, and each selected variable is negated with probability $1/2$ as well. Whether the instance is realizable or unrealizable depends on the random formulas generated. Out of the input variables, $peek_1^e, \ldots, peek_m^e, plate_1^e, \ldots, plate_{\lceil \frac{n}{2} \rceil}^e$ are unobservable. This corresponds to barriers keeping the system from seeing the holes on the environment side, as well as half of the environment plates (rounded up).

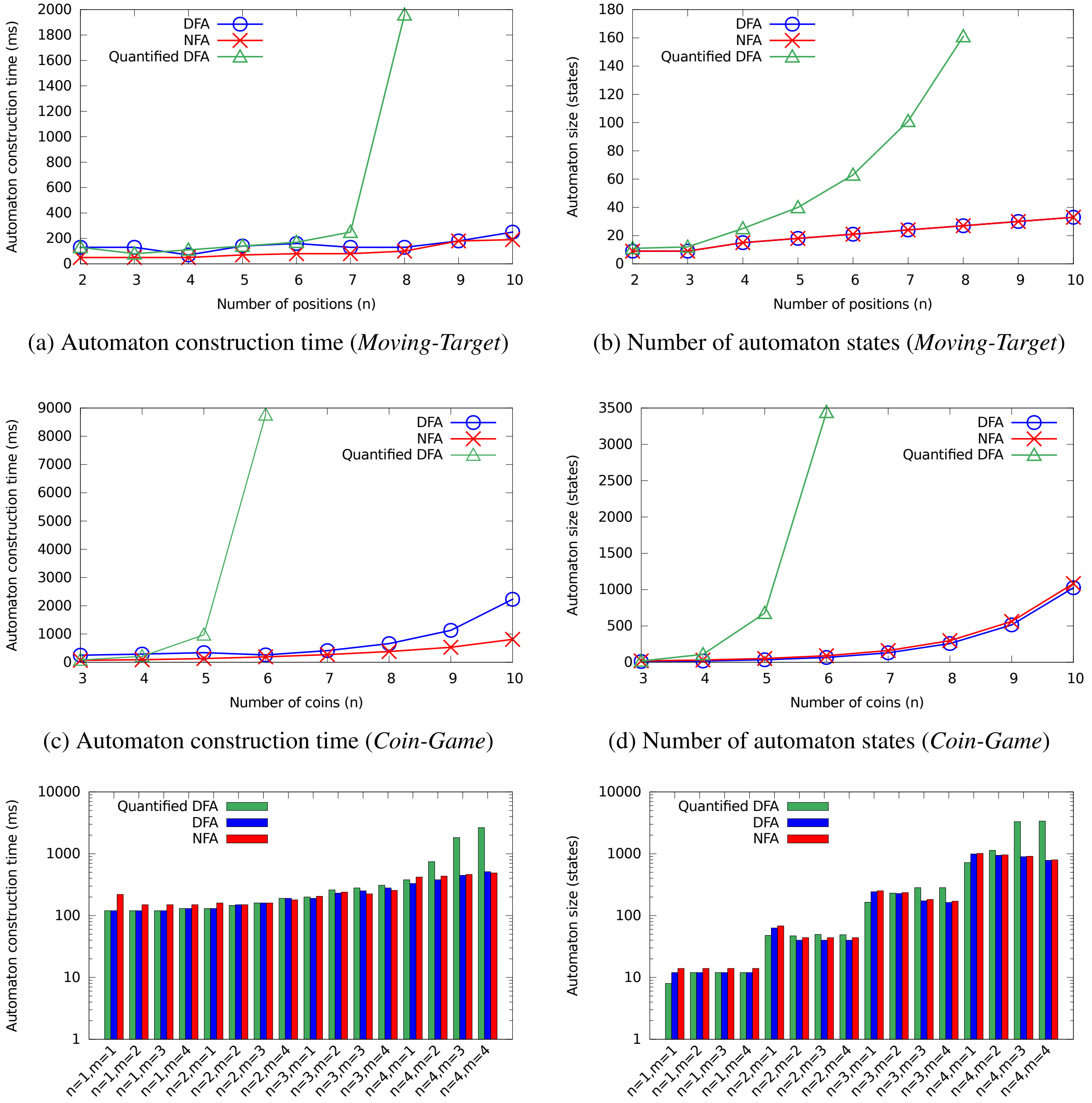
5.2 Experimental Setup and Results

We generated instances with n varying from 2 to 10 for the *Moving-Target* benchmarks and 3 to 10 for the *Coin-Game* benchmarks. For the *Private-Peek* benchmarks, we varied n and m from 1 to 4 and generated 30 random instances for each combination using the procedure described above. We report the median results for each combination of n and m. We ran all experiments on a single node of a high-performance cluster consisting of an Intel Xeon processor runninng at 2.6 GHz. Experiments had 32 GB of memory available and a timeout of 8 hours. Failed instances are due to either timeouts or memouts.

The LTL$_f$-synthesis procedure implemented in SYFT consists of two phases, one explicit and one symbolic. The first phase is the construction of an explicit automaton by MONA, and the second phase is the conversion of this automaton to a symbolic representation followed by the symbolic fixpoint computation used to compute the winning strategy. We analyze how the three approaches perform in each of these two phases, and then see how they contribute to the overall performance. Although SYFT uses a fixed variable ordering for BDDs, in order to reduce the impact that a single variable ordering has on BDD sizes and make for a more fair comparison between the different approaches, we enabled dynamic variable reordering [16], which tries to optimize the ordering of variables on the fly during execution.

5.2.1 Explicit Phase

We first analyze the explicit phase across the three approaches. Recall that MONA constructs an explicit NFA in the projection-based approach (via a DFA for the reverse language) and explicit DFAs for the other two approaches (in the MSO approach, with the unobservable inputs universally quantified). Recall



(a) Automaton construction time (*Moving-Target*)

(b) Number of automaton states (*Moving-Target*)

(c) Automaton construction time (*Coin-Game*)

(d) Number of automaton states (*Coin-Game*)

(e) Automaton construction time (*Private-Peek*)

(f) Number of automaton states (*Private-Peek*)

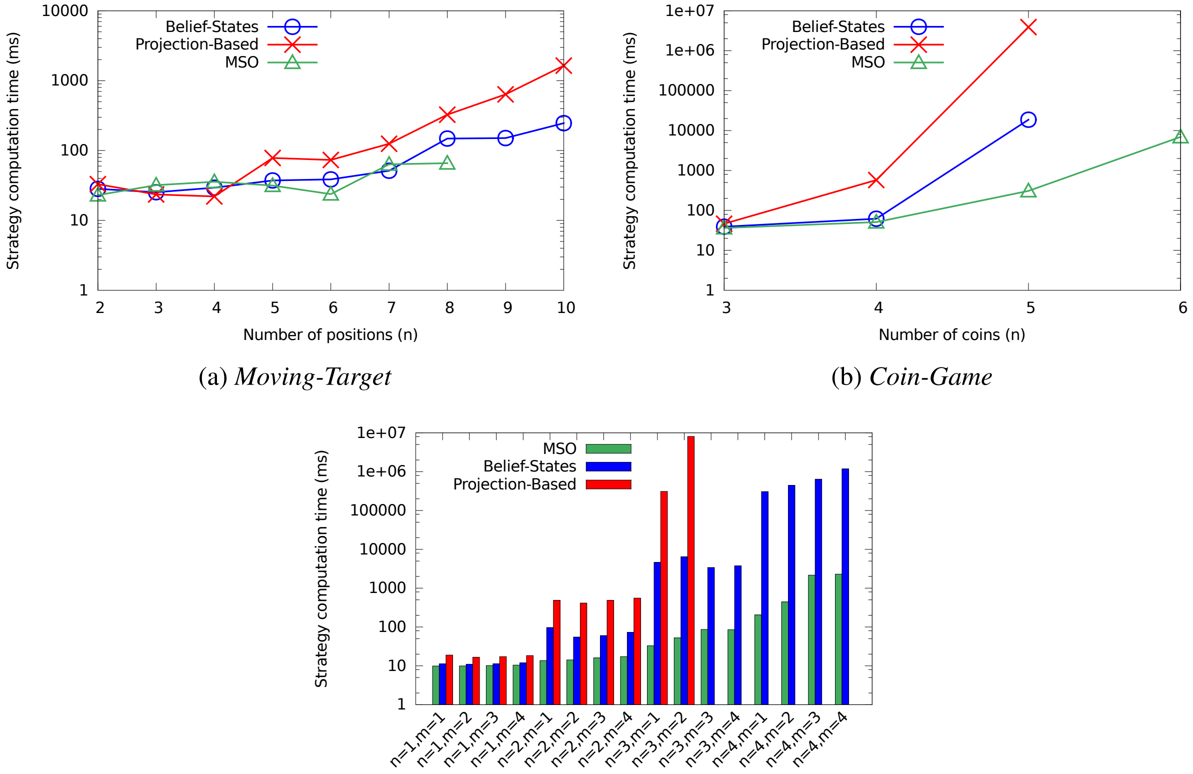
Figure 1: Automaton construction times and number of automaton states for different values of *n* and *m* for each benchmark family. Values for the *Private-Peek* benchmarks are the median of the 30 random instances, and are presented in log scale.

that the NFA can be at most exponential in the size of the original LTL$_f$ formula, while the DFA (quantified or not) will be doubly-exponential in the worst case. Interestingly, as can be seen in Figures 1a, 1c and 1e, there is not a big difference in running time between constructing a DFA and an NFA for the LTL$_f$ formula. In fact, as Figures 1b, 1d and 1f show, for the *Moving-Target* formulas the DFA and NFA of the formula have exactly the same number of states, while for the *Coin-Game* and *Private-Peek* benchmarks the NFA was in fact slightly larger. This happens in part because moves in these games are reversible, meaning that the DFA for the reverse language (which is reversed to produce the NFA) has a similar structure to the original language. Nevertheless, these results reinforce the observation from [32] that the exponential gap between DFA and NFA often does not occur in practice when the DFA is minimized. As this gap was the central reason for the exponential gap in complexity between the belief-states and projection-based constructions, this result is significant to highlight how theoretical analysis may not always accurately predict behavior in practice. On the other hand, the universally-quantified DFA, despite having the same worst-case as the non-quantified DFA, in practice grows much faster. In the *Private-Peek* instances it tends to be smaller than the DFA for low values of m but quickly grow to surpass it as m increases. In the *Moving-Target* and *Coins-Game* families the construction of the quantified DFA also could not be completed for larger values of n (> 8 for *Moving-Target* and > 6 for *Coins-Game*), not only due to the size of the final DFA but likely also due to the overhead of MONA's construction algorithm, as mentioned in Section 4.3.

5.2.2 Symbolic Phase

In the second phase, the explicit automata generated by MONA are converted into symbolic DFAs. In the case of the projection-based and belief-states constructions, this means performing symbolic subset construction as described in Sections 4.1 and 4.2, which means that the number of state variables in the symbolic representation is equal to the number of states in the explicit automaton. As each assignment of the state variables represents a state, this represents an exponential blowup in the state space caused by subset construction. On the other hand, in the MSO approach no subset construction is necessary and the state space of the universally-quantified DFA produced by MONA can be encoded in a logarithmic number of state variables. As a result, even though the quantified DFA is significantly larger, the number of state variables used in the symbolic representation of the MSO approach is smaller than in the other approaches. Even so, the size of the initial symbolic representation, measured in total number of nodes in the BDDs for the transition relation and accepting states, is larger for the MSO approach. This might be because the symbolic representations generated by subset construction display more structure than the one generated by a logarithmic encoding.

As can be seen in Figure 2, however, when computing the winning strategy from the symbolic representation of the DFA game the MSO approach was the fastest across the board. This suggests that the size of the implicit state space, represented by the number of state variables, is a more important factor in the performance of the second phase than the initial symbolic representation of the automaton. Note that the lack of results for larger n for the MSO approach in the *Moving-Target* and *Coin-Game* benchmarks is not due to the performance of the algorithm in this phase, but rather due to the quantified DFA not being able to be constructed in the first phase. Interestingly, even though the other two approaches were able to construct automata up to $n = 10$ for the *Coin-Game* benchmarks, they failed to solve the game for $n > 5$, while the MSO approach can still construct the automaton and solve the game for $n = 6$. Perhaps surprisingly, the projection-based approach had the worst performance overall. In addition to the number of states of the NFA being equal or slightly larger than the DFA, the symbolic representation and strategy-computation time were also significantly larger. It was also unable to solve the game in



(a) *Moving-Target*

(b) *Coin-Game*

(c) *Private-Peek*

Figure 2: Time taken to solve the DFA game for each value of *n* and *m* for each benchmark family, in log scale. Values for the *Private-Peek* family are the median of the 30 random instances, and missing values mean the median could not be computed because most random instances timed out.

most cases for larger values of *n* and *m* in the *Private-Peek* family. This serves as final evidence of the importance of complementing theoretical analysis with empirical evaluation.

5.2.3 End-to-end Picture

Given the results presented in the two phases above, two major conclusions can be drawn. The first is that when comparing between the MSO approach and the other two, there is a tradeoff between the explicit and symbolic phases. The MSO approach produces a much larger explicit automaton and requires more time to do so, and as a result is more likely to run out of time or memory to complete the first phase. On the other hand, the fact that this automaton is minimized, compared to the symbolic DFA generated by the other two approaches, leads to a significantly better performance when solving the game in the second phase. If we add the times for the two phases, the MSO approach takes slightly longer for the *Moving-Target* benchmarks, but performs significantly better in the *Coin-Game* and *Private-Peek* benchmarks. Overall, the MSO approach seems to be a good option as long as the construction of the quantified DFA can be completed. The second conclusion is that, unlike what the theory seems to suggest, the performance of the projection-based approach is more often than not worse than that of the belief-states construction. Although in the worst case there might be an exponential gap between NFA and DFA, this behavior has not been observed in practice, which closes the gap between the 2EXPTIME and 3EXPTIME complexity of the two constructions. Furthermore, in the second phase the projection-based approach produced a less efficient symbolic representation, and ultimately required more time to solve the game. These results also highlight the necessity of considering algorithmic details that are hard to account for in a purely theoretical analysis, such as the use of symbolic techniques.

6 Discussion

We have undertaken the first steps to bring synthesis under partial observability, which previously inhabited only the realm of pure theory, closer to practical application. Much work still remains to be done to scale to real-world scenarios, but supported by the conceptual simplicity of LTL_f synthesis compared to LTL synthesis and the availability of efficient tools such as MONA and SYFT we have presented the first implementation and empirical evaluation of algorithms for temporal synthesis under partial observability. Our experimental evaluation showed that the choice of algorithm for LTL_f synthesis under partial observability is not as straightforward in practice as the theoretical analysis from [11] would suggest.

First, although the projection-based approach is exponentially better than the belief-states construction in theory, this advantage depends on the assumption that the NFA constructed is smaller than the corresponding DFA. In our examples this assumption was violated, negating the advantage of this approach. Second, the use of symbolic algorithms means that synthesis performance depends not only on the number of automaton states, but also on the size of its symbolic representation. The projection-based approach's giving rise to a less efficient symbolic representation had a larger effect in the results than automata size. Finally, MONA enabled the introduction of an MSO-based approach that has its own pros and cons. Although that approach can be more efficient for computing a winning strategy, it pays a price during explicit DFA construction and may fail during this phase if the DFA is large. It would be interesting to investigate how the improvement of explicit DFA construction algorithms could help the MSO approach overcome this obstacle, and how this would change the general picture. As for the other algorithms, the priority should be to bridge their gap in performance during the symbolic fixpoint computation, which as observed in Section 5 is primarily due to the lack of minimization of the final symbolic DFA. This is not a trivial problem to solve, as minimization of symbolic state spaces is not always effective [17].

Our observations suggest that rather than there being a single best algorithm for LTL_f synthesis under partial observability, we need a portfolio of algorithms, and the best option will likely depend on the nature of the problem being solved. When the quantified DFA can be constructed explicitly within the available time and memory, the MSO approach will probably excel. For most other cases, the belief-states construction is likely to be the best option, the exception being extreme cases where the DFA is doubly exponential, and therefore the use of the NFA by the projection-based approach provides an advantage. This is a very different result than what is suggested by the purely theoretical analysis, showing the importance of studying the problem empirically as well.

Acknowledgments

Work supported in part by NSF grants IIS-1527668, CCF-1704883, IIS-1830549, and an award from the Maryland Procurement Office.

References

[1] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin & Jean-François Raskin (2012): *Acacia+, a Tool for LTL Synthesis*. In: *CAV*, pp. 652–657, doi:10.1007/978-3-642-31424-7_45.

[2] Blai Bonet & Hector Geffner (2000): *Planning with Incomplete Information as Heuristic Search in Belief Space*. In: *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pp. 52–61.

[3] Randal E. Bryant (1986): *Graph-Based Algorithms for Boolean Function Manipulation*. *IEEE Trans. Computers* 35(8), pp. 677–691, doi:10.1109/TC.1986.1676819.

[4] Daniel Bryce, Subbarao Kambhampati & David E. Smith (2006): *Planning Graph Heuristics for Belief Space Search*. *J. Artif. Intell. Res.* 26, pp. 35–99, doi:10.1613/jair.1869.

[5] J. Büchi (1960): *Weak Second-Order Arithmetic and Finite Automata*. *Mathematical Logic Quarterly - MLQ* 6, pp. 66–92, doi:10.1002/malq.19600060105.

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

[7] Alberto Camacho, Meghyn Bienvenu & Sheila A. McIlraith (2019): *Towards a Unified View of AI Planning and Reactive Synthesis*. In: *ICAPS*, pp. 58–67.

[8] Ashok K. Chandra, Dexter Kozen & Larry J. Stockmeyer (1981): *Alternation*. *J. ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.

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

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

[11] Giuseppe De Giacomo & Moshe Y. Vardi (2016): *LTL_f and LDL_f Synthesis under Partial Observability*. In: *IJCAI*, pp. 1044–1050.

[12] Laurent Doyen & Jean-François Raskin (2011): *Games with Imperfect Information: Theory and Algorithms*, p. 185–212. Cambridge University Press, doi:10.1017/CBO9780511973468.007.

[13] 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*, doi:10.1007/978-3-319-46520-3_8.

[14] Rüdiger Ehlers (2011): *Unbeast: Symbolic Bounded Synthesis*. In Parosh Aziz Abdulla & K. Rustan M. Leino, editors: *TACAS*, *Lecture Notes in Computer Science* 6605, Springer, pp. 272–275, doi:10.1007/978-3-642-19835-9_25.

[15] Peter Faymonville, Bernd Finkbeiner & Leander Tentrup (2017): *BoSy: An Experimentation Framework for Bounded Synthesis*. In: *CAV*, pp. 325–332, doi:10.1007/978-3-319-63390-9_17.

[16] Eric Felt, Gary York, Robert K. Brayton & Alberto L. Sangiovanni-Vincentelli (1993): *Dynamic Variable Reordering for BDD Minimization*. In: *EURO-DAC*, IEEE Computer Society, pp. 130–135, doi:10.1109/EURDAC.1993.410627.

[17] Kathi Fisler & Moshe Y. Vardi (2002): *Bisimulation Minimization and Symbolic Model Checking*. *Formal Methods Syst. Des.* 21(1), pp. 39–78, doi:10.1023/A:1016091902809.

[18] 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.

[19] Robert P. Goldman & Mark S. Boddy (1996): *Expressive Planning and Explicit Knowledge*. In: *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, pp. 110–117.

[20] 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.

[21] 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.

[22] Orna Kupferman & Moshe Vardi (1997): *Synthesis with Incomplete Informatio*. In: *ICTL*, pp. 1044–1050, doi:10.1007/978-94-015-9586-5_6.

[23] Shlomi Maliah, Ronen I. Brafman, Erez Karpas & Guy Shani (2014): *Partially Observable Online Contingent Planning Using Landmark Heuristics*. In: *ICAPS*.

[24] Philipp J. Meyer, Salomon Sickert & Michael Luttenberger (2018): *Strix: Explicit Reactive Synthesis Strikes Back!* In Hana Chockler & Georg Weissenbacher, editors: *CAV*, *Lecture Notes in Computer Science* 10981, Springer, pp. 578–586, doi:10.1007/978-3-319-96145-3_31.

[25] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *18th Annual Symposium on Foundations of Computer Science*, pp. 46–57, doi:10.1109/SFCS.1977.32.

[26] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *Sixteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 179–190, doi:10.1145/75277.75293.

[27] John H. Reif (1984): *The Complexity of Two-Player Games of Incomplete Information*. *J. Comput. Syst. Sci.* 29(2), pp. 274–301, doi:10.1016/0022-0000(84)90034-5.

[28] Jussi Rintanen (2004): *Complexity of Planning with Partial Observability*. In: *ICAPS*, pp. 345–354.

[29] Roni Rosner (1991): *Modular Synthesis of Reactive Systems*.

[30] A. Prasad Sistla, Moshe Y. Vardi & Pierre Wolper (1985): *The Complementation Problem for Büchi Automata with Applications to Temporal Logic (Extended Abstract)*. In: *Automata, Languages and Programming, 12th Colloquium*, pp. 465–474, doi:10.1007/BFb0015772.

[31] Larry J. Stockmeyer & Ashok K. Chandra (1979): *Provably Difficult Combinatorial Games*. *SIAM J. Comput.* 8(2), pp. 151–174, doi:10.1137/0208013.

[32] 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.

[33] Moshe Y. Vardi & Larry J. Stockmeyer (1985): *Improved Upper and Lower Bounds for Modal Logics of Programs: Preliminary Report*. In: *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pp. 240–251, doi:10.1145/22145.22173.

[34] 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.

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

[36] Wieslaw Zielonka (1998): *Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees*. *Theor. Comput. Sci.* 200(1-2), pp. 135–183, doi:10.1016/S0304-3975(98)00009-7.