Engineering an $\mathsf{LTL_f}$ Synthesis Tool

Alexandre Duret-Lutz[1], Shufang Zhu[2], Nir Piterman[3], Giuseppe De Giacomo[4], and Moshe Y. Vardi[5]

[1] LRE, EPITA, Le Kremlin-Bicêtre, France
[2] University of Liverpool, Liverpool, UK
[3] University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden
[4] Sapienza University of Rome, Rome, Italy
[5] Rice University, Houston, Texas, USA

Abstract. The problem of $\mathsf{LTL_f}$ reactive synthesis is to build a transducer, whose output is based on a history of inputs, such that, for every infinite sequence of inputs, the conjoint evolution of the inputs and outputs has a prefix that satisfies a given $\mathsf{LTL_f}$ specification.
We describe the implementation of an $\mathsf{LTL_f}$ synthesizer that outperforms existing tools on our benchmark suite. This is based on a new, direct translation from $\mathsf{LTL_f}$ to a DFA represented as an array of Binary Decision Diagrams (MTBDDs) sharing their nodes. This MTBDD-based representation can be interpreted directly as a reachability game that is solved on-the-fly during its construction.

1 Introduction

Reactive synthesis is concerned with synthesizing programs (a.k.a. strategies) for reactive computations (e.g., processes, protocols, controllers, robots) in active environments [45,30,26], typically, from temporal logic specifications. In AI, Reactive Synthesis, which is related to (strong) planning for temporally extended goals in fully observable nondeterministic domains [16,3,4,14,6,34,21,15], has been studied with a focus on logics on finite traces such as $\mathsf{LTL_f}$ [33,7,22,23]. In fact, $\mathsf{LTL_f}$ synthesis [23] is one of the two main success stories of reactive synthesis so far (the other being the GR(1) fragment of LTL [44]), and has brought about impressive advances in scalability [53,8,18,20].

Reactive synthesis for $\mathsf{LTL_f}$ involves the following steps [23]: (1) distinguishing uncontrollable input ($\mathcal{I}$) and controllable output ($\mathcal{O}$) variables in an $\mathsf{LTL_f}$ specification φ of the desired system behavior; (2) constructing a DFA accepting the behaviors satisfying φ; (3) interpreting this DFA as a two-player reachability game, and finding a controller winning strategy. Step (2) has two main bottlenecks: the DFA is worst-case doubly-exponential and its propositional alphabet $\Sigma = 2^{\mathcal{I}\cup\mathcal{O}}$ is exponential. The first only happens in the worst case, while the second blow-up – which we call *alphabet explosion* – always happens.

Mona [39] addresses the alphabet-explosion problem, which happens also in MSO, by representing a DFA with Multi-Terminal Binary Decision Diagrams (MTBDDs) [36]. MTBDDs are a variant of BDDs [12] with arbitrary terminal values. If terminal values encode destination states, an MTBDD can compactly represent all outgoing transitions of a single DFA state. A DFA is represented, through its transition function, as an array of MTBDDs sharing their nodes.

The first $\mathsf{LTL_f}$ synthesizer, Syft [53], converted $\mathsf{LTL_f}$ into first-order logic in order to build a MTBDD-encoded DFA with Mona. Syft then converted this DFA into a BDD representation to solve the reachability game using a symbolic fixpoint computation. Syft demonstrated that DFA construction is the main bottleneck in $\mathsf{LTL_f}$ synthesis, motivating several follow-up efforts.

One approach to effective DFA construction uses compositional techniques, decomposing the input $\mathsf{LTL_f}$ formula into smaller subformulas whose DFAs can be minimized before being recombined. Lisa [8] decomposes top-level conjunctions, while Lydia [19] and LydiaSyft [29] decompose every operator.

Compositional methods construct the full DFA before synthesis can proceed, limiting their scalability. On-the-fly approaches [50] construct the DFA incrementally, while simultaneously solving the game, allowing strategies to be found before the complete DFA is built. The DFA construction may use various techniques. Cynthia [20] uses Sentential Decision Diagrams (SDDs) [17] to generate all outgoing transitions of a state at once. Alternatively, Nike [28] and MoGuSer [51] use a SAT-based method to construct one successor at a time. The game is solved by forward exploration with suitable backpropagation.

Contributions and Outline In Section 3, we propose a direct and efficient translation from $\mathsf{LTL_f}$ to MTBDD-encoded DFA (henceforth called MTDFA). In Section 4, we show that given an appropriate ordering of BDD variables, $\mathsf{LTL_f}$ realizability can be solved by interpreting the MTBDD nodes of the MTDFA as the vertices of a reachability game, known to be solvable in linear time by backpropagation of the vertices that are winning for the *output* player. We give a linear-time implementation for solving the game on-the-fly while it is constructed. For more opportunities to abort the on-the-fly construction earlier, we additionally backpropagate vertices that are known to be winning by the *input* player. We implemented these techniques in two tools (`ltlf2dfa` and `ltlfsynt`) that compare favorably with other existing tools in benchmarks from the $\mathsf{LTL_f}$-Synthesis Competition. To meet space limits, Section 5 only reports on the $\mathsf{LTL_f}$ realizability benchmark, and we refer readers to our artifact for the other results [24].

2 Preliminaries

2.1 Words over Assignments

A *word over* σ of length n over an alphabet Σ is a function $\sigma : \{0, 1, \ldots, n-1\} \to \Sigma$. We use Σ^n (resp. $\Sigma^\star$ and Σ^+) to denote the set of words of length n (resp. any length $n \geq 0$ and $n > 0$). We use $|\sigma|$ to represent the length of a word σ. For $\sigma \in \Sigma^n$ and $0 \leq i < n$, $\sigma(..i)$ denotes the prefix of σ of length $i + 1$.

Let $\mathcal{P}$ be a finite set of Boolean variables (a.k.a. *atomic propositions*). We use $\mathbb{B}^{\mathcal{P}}$ to denote the set of all assignments, i.e., functions $\mathcal{P} \to \mathbb{B}$ mapping variables to values in $\mathbb{B} = \{\bot, \top\}$.

Given two disjoint sets of variables $\mathcal{P}_1$ and $\mathcal{P}_2$, and two assignments $w_1 \in \mathbb{B}^{\mathcal{P}_1}$ and $w_2 \in \mathbb{B}^{\mathcal{P}_2}$, we use $w_1 \sqcup w_2 : (\mathcal{P}_1 \cup \mathcal{P}_2) \to \mathbb{B}$ to denote their combination.

In a system modeled using discrete Boolean signals that evolve synchronously, we assign a variable to each signal, and use a word $\sigma \in (\mathbb{B}^{\mathcal{P}})^+$ over assignments of $\mathcal{P}$ to represent the conjoint evolution of all signals over time.

We extend $\sqcup$ to such words. For two words $\sigma_1 \in (\mathbb{B}^{\mathcal{P}_1})^n$, $\sigma_2 \in (\mathbb{B}^{\mathcal{P}_2})^n$ of length n over assignments that use disjoint sets of variables, we use $\sigma_1 \sqcup \sigma_2 \in (\mathbb{B}^{\mathcal{P}_1 \cup \mathcal{P}_2})^n$ to denote a word such that $(\sigma_1 \sqcup \sigma_2)(i) = \sigma_1(i) \sqcup \sigma_2(i)$ for $0 \le i < n$.

2.2 Linear Temporal Logic over Finite, Nonempty Words.

We use classical $\mathsf{LTL_f}$ semantics over nonempty finite words [22].

Definition 1 ($\mathsf{LTL_f}$ formulas). *An $\mathsf{LTL_f}$ formula φ is built from a set $\mathcal{P}$ of variables, using the following grammar where $p \in \mathcal{P}$, and $\odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, ...\}$ is any Boolean operator: $\varphi ::= tt \mid f\!f \mid p \mid \neg\varphi \mid \varphi \odot \varphi \mid \mathsf{X}\varphi \mid \mathsf{X}^!\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{R}\, \varphi \mid \mathsf{G}\varphi \mid \mathsf{F}\varphi$.*

Symbols tt and $f\!f$ represent the true *and* false *$\mathsf{LTL_f}$ formulas. Temporal operators are X (weak next), $\mathsf{X}^!$ (strong next), U (until), R (release), G (globally), and F (finally). $\mathsf{LTL_f}(\mathcal{P})$ denotes the set of formulas produced by the above grammar. We use $\mathsf{sf}(\varphi)$ to denote the set of subformulas for φ. A* maximal temporal subformula *of φ is a subformula whose primary operator is temporal and that is not strictly contained within any other temporal subformula of φ.*

The satisfaction of a formula $\varphi \in \mathsf{LTL_f}(\mathcal{P})$ by word $\sigma \in (\mathbb{B}^{\mathcal{P}})^+$ of length $n > 0$ at position $0 \le i < n$, denoted $\sigma, i \models \varphi$, is defined as follows.

$$\begin{aligned}
\sigma, i \models tt &\iff i < n & \sigma, i \models \mathsf{X}\varphi &\iff (i+1=n) \vee (\sigma, i+1 \models \varphi)\\
\sigma, i \models f\!f &\iff i = n & \sigma, i \models \mathsf{X}^!\varphi &\iff (i+1<n) \wedge (\sigma, i+1 \models \varphi)\\
\sigma, i \models p &\iff p \in \sigma(i) & \sigma, i \models \mathsf{F}\varphi &\iff \exists j \in [i,n),\, \sigma, j \models \varphi\\
\sigma, i \models \neg\varphi &\iff \neg(\sigma, i \models \varphi) & \sigma, i \models \mathsf{G}\varphi &\iff \forall j \in [i,n),\, \sigma, j \models \varphi
\end{aligned}$$

$$\begin{aligned}
\sigma, i \models \varphi_1 \odot \varphi_2 &\iff (\sigma, i \models \varphi_1) \odot (\sigma, i \models \varphi_2)\\
\sigma, i \models \varphi_1 \,\mathsf{U}\, \varphi_2 &\iff \exists j {\in} [i,n), (\sigma, j \models \varphi_2) \wedge (\forall k {\in} [i,j),\, \sigma, k \models \varphi_1)\\
\sigma, i \models \varphi_1 \,\mathsf{R}\, \varphi_2 &\iff \forall j {\in} [i,n), (\sigma, j \models \varphi_2) \vee (\exists k {\in} [i,j),\, \sigma, k \models \varphi_1)
\end{aligned}$$

The set of words that satisfy $\varphi \in \mathsf{LTL_f}(\mathcal{P})$ is $\mathscr{L}(\varphi) = \{\sigma \in (\mathbb{B}^{\mathcal{P}})^+ \mid \sigma, 0 \models \varphi\}$.

Example 1. Consider the following $\mathsf{LTL_f}$ formulas over $\mathcal{P} = \{i_0, i_1, i_2, o_1, o_2\}$: $\Psi_1 = \mathsf{G}((i_0 \rightarrow (o_1 \leftrightarrow i_1)) \wedge ((\neg i_0) \rightarrow (o_1 \leftrightarrow i_2)))$, and $\Psi_2 = (\mathsf{GF}o_2) \leftrightarrow (\mathsf{F}i_0)$. If we interpret i_0, i_1, i_2 as input signals, and o_1, o_2 as output signals, formula Ψ_1 specifies a 1-bit multiplexer: the value of the signal o_1 should be equal to the value of either i_1 or i_2 depending on the setting of i_0. Formula Ψ_2 specifies that the last value of o_2 should be $\top$ if and only if i_0 was $\top$ at some instant.

Definition 2 (Propositional Equivalence [27]). *For $\varphi \in \mathsf{LTL_f}(\mathcal{P})$, let φ_P be the Boolean formula obtained from φ by replacing every maximal temporal subformula ψ by a Boolean variable x_ψ. Two formulas $\alpha, \beta \in \mathsf{LTL_f}(\mathcal{P})$ are* propositionally equivalent*, denoted $\alpha \equiv \beta$, if α_P and β_P are equivalent Boolean formulas.*

Example 2. Formulas $\alpha = (\mathsf{G}b) \vee ((\mathsf{F}a) \wedge (\mathsf{G}b))$ and $\beta = \mathsf{G}b$ are propositionally equivalent. Indeed, $\alpha_P = x_{\mathsf{G}b} \vee (x_{\mathsf{F}a} \wedge x_{\mathsf{G}b}) = x_{\mathsf{G}b} = \beta_P$.

Note that $\alpha \equiv \beta$ implies $\mathscr{L}(\alpha) = \mathscr{L}(\beta)$, but the converse is not true in general. Since $\equiv$ is an equivalence relation, we use $[\alpha]_\equiv \in \mathsf{LTL_f}(\mathcal{P})$ to denote some unique representative of the equivalence class of α with respect to $\equiv$.

2.3 $\mathsf{LTL_f}$ Realizability

Our goal is to build a tool that decides whether an $\mathsf{LTL_f}$ formula is *realizable*.

Definition 3 ([23,37]). *Given two disjoint sets of variables $\mathcal{I}$ (inputs) and $\mathcal{O}$ (outputs), a controller is a function $\rho : \mathcal{I}^* \to \mathcal{O}$, that produces an assignment of output variables given a history of assignments of input variables.*

Given a word of n input assignments $\sigma \in (\mathbb{B}^{\mathcal{I}})^n$, the controller can be used to generate a word of n output assignments $\sigma_\rho \in (\mathbb{B}^{\mathcal{I}})^n$. The definition of σ_ρ may use two semantics depending on whether we want to the controller to have access to the current input assignment to decide the output assignment:
Mealy semantics: $\sigma_\rho(i) = \rho(\sigma(..i))$ *for all* $0 \le i < n$.
Moore semantics: $\sigma_\rho(i) = \rho(\sigma(..i-1))$ *for all* $0 \le i < n$.

A formula $\varphi \in \mathsf{LTL_f}(\mathcal{I} \cup \mathcal{O})$ is said to be Mealy-realizable *or* Moore-realizable *if there exists a controller ρ such that for any word $\sigma \in (\mathbb{B}^{\mathcal{I}})^\omega$ there exists a position k such that $(\sigma \sqcup \sigma_\rho)(..k) \in \mathscr{L}(\varphi)$ using the desired semantics.*

Example 3. Formula Ψ_1 (from Example 1) is Mealy-realizable but not Moore-realizable. Formula Ψ_2 is both Mealy and Moore-realizable.

2.4 Multi-Terminal BDDs

Let $\mathcal{S}$ be a finite set. Given a finite set of variables $\mathcal{P} = \{p_0, p_1, \ldots, p_{n-1}\}$ (that are implicitly ordered by their index) we use $f : \mathbb{B}^{\mathcal{P}} \to \mathcal{S}$ to denote a function that maps an assignment of all those variables to an element of $\mathcal{S}$. Given a variable $p \in \mathcal{P}$ and a Boolean $b \in \mathbb{B}$, the function $f_{p=b} : \mathbb{B}^{\mathcal{P} \setminus \{p\}} \to \mathcal{S}$ represents a generalized co-factor obtained by replacing p by b in f. When $\mathcal{S} = \mathbb{B}$, a function $f : \mathbb{B}^{\mathcal{P}} \to \mathbb{B}$ can be encoded into a Binary Decision Diagram (BDD) [11]. Multi-Terminal Binary Decision Diagrams (MTBDDs) [42,43,32,39], also called Algebraic Decision Diagrams (ADDs) [5,49], generalize BDDs by allowing arbitrary values on the leaves of the graph.

A *Multi-Terminal BDD* encodes any function $f : \mathbb{B}^{\mathcal{P}} \to \mathcal{S}$ as a rooted, directed acyclic graph. We use the term *nodes* to refer to the vertices of this graph. All nodes in an MTBDD are represented by triples of the form (p, ℓ, h). In an internal node, $p \in \mathcal{P}$ and ℓ, h point to successors MTBDD nodes called the low and high links. The intent is that if (p, ℓ, h) is the root of the MTBDD representing the function f, then ℓ and h are the roots of the MTBDDs representing the functions $f_{p=\bot}$ and $f_{p=\top}$, respectively. Leaves of the graph, called *terminals*, hold values in $\mathcal{S}$. For consistency with internal nodes, we represent terminals with a triple of the form (∞, s, ∞) where $s \in \mathcal{S}$. When comparing the first elements of different triplets, we assume that ∞ is greater than all variables. We use $\mathsf{MTBDD}(\mathcal{P},\mathcal{S})$ to denote the set of MTBDD nodes that can appear in the representation of an arbitrary function $\mathbb{B}^{\mathcal{P}} \to \mathcal{S}$.

Following the classical implementations of BDD packages [11,1], we assume that MTBDDs are *ordered* (variables of $\mathcal{P}$ are ordered and visited in increasing order by all branches of the MTBDD) and *reduced* (isomorphic subgraphs are merged by representing each triplet only once, and internal nodes with identical low and high links are skipped over). Doing so ensures that each function $f : \mathbb{B}^{\mathcal{P}} \to \mathcal{S}$ has a unique MTBDD representation for a given order of variables.

Given $m \in \mathsf{MTBDD}(\mathcal{P},\mathcal{S})$ and an assignment $w \in \mathbb{B}^{\mathcal{P}}$, we note $m(w)$ the element of $\mathcal{S}$ stored on the terminal of m that is reached after following the assignment w in the structure of m. We use $|m|$ to denote the number of MTBDD nodes that can be reached from m.

Let $m_1 \in \mathsf{MTBDD}(\mathcal{P},\mathcal{S}_1)$ and $m_2 \in \mathsf{MTBDD}(\mathcal{P},\mathcal{S}_2)$ be two MTBDD nodes representing functions $f_i : \mathbb{B}^{\mathcal{P}} \to \mathcal{S}_i$, and let $\odot : \mathcal{S}_1 \times \mathcal{S}_2 \to \mathcal{S}_3$, be a binary operation. One can easily construct $m_3 \in \mathsf{MTBDD}(\mathcal{P},\mathcal{S}_3)$ representing the function $f_3(p_0,\ldots,p_{n-1}) = f_1(p_0,\ldots,p_{n-1}) \odot f_2(p_0,\ldots,p_{n-1})$, by generalizing the `apply2` function typically found in BDD libraries [32]. We use $m_1 \odot m_2$ to denote the MTBDD that results from this construction.

For $m \in \mathsf{MTBDD}(\mathcal{P},\mathcal{S})$ we use $\mathtt{leaves}(m) \subseteq \mathcal{S}$ to denote the elements of $\mathcal{S}$ that label terminals reachable from m. This set can be computed in $\Theta(|m|)$.

2.5 MTBDD-Based Deterministic Finite Automata

We now define an MTBDD-based representation of a DFA with a propositional alphabet, inspired by Mona's DFA representation [36,39].

Definition 4 (MTDFA). *An MTDFA is a tuple $\mathcal{A} = \langle \mathcal{Q}, \mathcal{P}, \iota, \Delta \rangle$, where $\mathcal{Q}$ is a finite set of* states*, $\mathcal{P}$ is a finite (and ordered) set of* variables*, $\iota \in \mathcal{Q}$ is the initial state, $\Delta : \mathcal{Q} \to \mathsf{MTBDD}(\mathcal{P}, \mathcal{Q} \times \mathbb{B})$ represents the set of outgoing transitions of each state. For a word $\sigma \in (\mathbb{B}^{\mathcal{P}})^{\star}$ of length n, let $(q_i, b_i)_{0 \le i \le n}$ be a sequence of pairs defined recursively as follows: $(q_0, b_0) = (\iota, \bot)$, and for $0 < i \le |\sigma|$, $(q_i, b_i) = \Delta(q_{i-1})(\sigma(i-1))$ is the pair reached by evaluating assignment $\sigma(i-1)$ on $\Delta(q_{i-1})$. The word σ is accepted by $\mathcal{A}$ iff $b_n = \top$. The language of $\mathcal{A}$, denoted $\mathscr{L}(A)$, is the set of words accepted by $\mathcal{A}$.*

Example 4. Figure 1 shows an MTDFA where $\mathcal{Q} \subseteq \mathsf{LTL_f}(\{i_0, i_1, i_2, o_1, o_2\})$. The set of states $\mathcal{Q}$ are the dashed rectangles on the left. For each such a state $q \in \mathcal{Q}$, the dashed arrow points to the MTBDD node representing $\Delta(q)$. The MTBDD nodes are shared between all states. If, starting from the initial state ι at the top-left, we read the assignment $w = (i_0 \to \top, i_1 \to \top, i_2 \to \top, o_1 \to \top, o_2 \to \top)$, we should follow only the high links (plain arrows) and we reach the $\Psi_1 \wedge (\mathsf{GF}o_2)$ accepting terminal. If we read this assignment a second-time, starting this time from state $\Psi_1 \wedge (\mathsf{GF}o_2)$ on the left, we reach the same accepting terminal. Therefore, non-empty words of the form $www\ldots w$ are accepted by this automaton.

An MTDFA can be regarded as a semi-symbolic representation of a DFA over propositional alphabet. From a state q and reading the assignment w, the

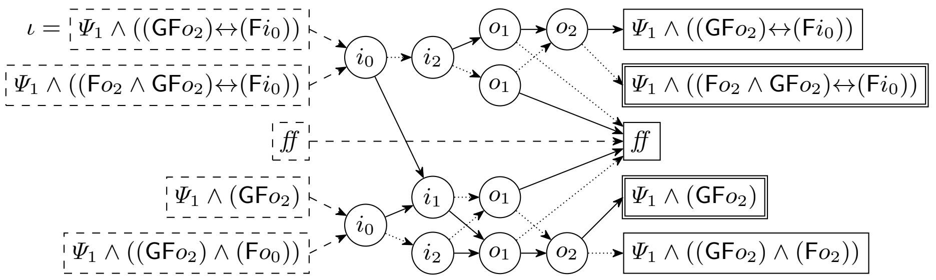
$\iota =$
$\Psi_1 \wedge ((\mathsf{GF}o_2)\leftrightarrow(\mathsf{F}i_0))$
$\Psi_1 \wedge ((\mathsf{F}o_2 \wedge \mathsf{GF}o_2)\leftrightarrow(\mathsf{F}i_0))$
ff
$\Psi_1 \wedge (\mathsf{GF}o_2)$
$\Psi_1 \wedge ((\mathsf{GF}o_2) \wedge (\mathsf{F}o_0))$
i_0
i_2
o_1
o_2
i_1
$\Psi_1 \wedge ((\mathsf{GF}o_2)\leftrightarrow(\mathsf{F}i_0))$
$\Psi_1 \wedge ((\mathsf{F}o_2 \wedge \mathsf{GF}o_2)\leftrightarrow(\mathsf{F}i_0))$
ff
$\Psi_1 \wedge (\mathsf{GF}o_2)$
$\Psi_1 \wedge ((\mathsf{GF}o_2) \wedge (\mathsf{F}o_2))$

Fig. 1. An MTDFA where $\mathcal{P} = \{i_0, i_1, i_2, o_0, o_1\}$ and $\mathcal{Q} \subseteq \mathsf{LTL_f}(\mathcal{P})$. Following classical BDD representations a BDD node (p, ℓ, h) is represented by $\textcircled{p}\overset{\ell}{\underset{h}{\rightrightarrows}}$. A terminal $(\infty, (\alpha, b), \infty)$ is represented by $\boxed{\alpha}$ if $b = \bot$, or $\boxed{\boxed{\alpha}}$ if $b = \top$. Finally, MTBDD $m = \Delta(\alpha)$ representing the successors of state α is indicated with $\boxed{\alpha} \dashrightarrow m$. Subformula Ψ_1 abbreviates $\mathsf{G}((i_0 \rightarrow (o_1 \leftrightarrow i_1)) \wedge ((\neg i_0) \rightarrow (o_1 \leftrightarrow i_2)))$.

automaton jumps to the state q' that is the result of computing $(q', b) = \Delta(q)(w)$. The value of b indicates whether that assignment is allowed to be the last one of the word being read. By definition, an MTDFA cannot accept the empty word.

MTDFAs are compact representations of DFAs, because the MTBDD representation of the successors of each state can share their common nodes. Boolean operations can be implemented over MTDFAs, with the expected semantics, i.e., $\mathscr{L}(\mathcal{A}_1 \odot \mathcal{A}_2) = \{\sigma \in (\mathbb{B}^{\mathcal{P}})^+ \mid (\sigma \in \mathscr{L}(\mathcal{A}_1)) \odot (\sigma \in \mathscr{L}(\mathcal{A}_2))\}$.

3 Translating $\mathsf{LTL_f}$ to MTBDD and MTDFA

This section shows how to directly transform a formula $\varphi \in \mathsf{LTL_f}(\mathcal{P})$ into an MTDFA $\mathcal{A}_\varphi = \langle \mathcal{Q}, \mathcal{P}, \varphi, \Delta \rangle$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathcal{A}_\varphi)$. The translation is reminiscent of other translations of $\mathsf{LTL_f}$ to DFA [22,20], but it leverages the fact that MTBBDs can provide a normal form for $\mathsf{LTL_f}$ formulas.

The construction maps states to $\mathsf{LTL_f}$ formulas, i.e., $\mathcal{Q} \subseteq \mathsf{LTL_f}(\mathcal{P})$. Terminals appearing in the MTBDDs of $\mathcal{A}_\varphi$ will be labeled by pairs $(\alpha, b) \in \mathsf{LTL_f}(\mathcal{P}) \times \mathbb{B}$, so we use $\mathsf{term}(\alpha, b) = (\infty, (\alpha, b), \infty)$ to shorten the notation from Section 2.4.

The conversion from φ to $\mathcal{A}_\varphi$ is based on the function $\mathsf{tr} : \mathsf{LTL_f}(\mathcal{P}) \rightarrow \mathsf{MTBDD}(\mathcal{P}, \mathsf{LTL_f}(\mathcal{P}) \times \mathbb{B})$ defined inductively as follows:

$$\mathsf{tr}(ff) = \mathsf{term}(ff, \bot) \qquad \mathsf{tr}(\mathsf{X}\alpha) = \mathsf{term}(\alpha, \top)$$
$$\mathsf{tr}(tt) = \mathsf{term}(tt, \top) \qquad \mathsf{tr}(\mathsf{X}^!\alpha) = \mathsf{term}(\alpha, \bot)$$
$$\mathsf{tr}(p) = (p, \mathsf{term}(ff,\bot), \mathsf{term}(tt,\top)) \text{ for } p \in \mathcal{P} \qquad \mathsf{tr}(\neg\alpha) = \neg\mathsf{tr}(\alpha)$$
$$\mathsf{tr}(\alpha \odot \beta) = \mathsf{tr}(\alpha) \odot \mathsf{tr}(\beta) \text{ for any } \odot \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$$
$$\mathsf{tr}(\alpha \,\mathsf{U}\, \beta) = \mathsf{tr}(\beta) \vee (\mathsf{tr}(\alpha) \wedge \mathsf{term}(\alpha \,\mathsf{U}\, \beta, \bot)) \qquad \mathsf{tr}(\mathsf{F}\alpha) = \mathsf{tr}(\alpha) \vee \mathsf{term}(\mathsf{F}\alpha, \bot)$$
$$\mathsf{tr}(\alpha \,\mathsf{R}\, \beta) = \mathsf{tr}(\beta) \wedge (\mathsf{tr}(\alpha) \vee \mathsf{term}(\alpha \,\mathsf{R}\, \beta, \top)) \qquad \mathsf{tr}(\mathsf{G}\alpha) = \mathsf{tr}(\alpha) \wedge \mathsf{term}(\mathsf{G}\alpha, \top)$$

Boolean operators that appear to the right of the equal sign are applied on MTBDDs as discussed in Section 2.4. Terminals in $\mathsf{LTL_f}(\mathcal{P}) \times \mathbb{B}$ are combined with: $(\alpha_1, b_1) \odot (\alpha_2, b_2) = ([\alpha_1 \odot \alpha_2]_\equiv, b_1 \odot b_2)$ and $\neg(\alpha, b) = ([\neg\alpha]_\equiv, \neg b)$.

Theorem 1. *For $\varphi \in \mathsf{LTL_f}(\mathcal{P})$, let $\mathcal{A}_\varphi = \langle \mathcal{Q}, \mathcal{P}, \iota, \Delta \rangle$ be the MTDFA obtained by setting $\iota = [\varphi]_\equiv$, $\Delta = \mathtt{tr}$, and letting $\mathcal{Q}$ be the smallest subset of $\mathsf{LTL_f}(\mathcal{P})$ such that $\iota \in \mathcal{Q}$, and such that for any $q \in \mathcal{Q}$ and for any $(\alpha, b) \in \mathtt{leaves}(\Delta(q))$, then $\alpha \in \mathcal{Q}$. With this construction, $|\mathcal{Q}|$ is finite and $\mathscr{L}(\varphi) = \mathscr{L}(\mathcal{A}_\varphi)$.*

Proof. (sketch) By definition of $\mathtt{tr}$, $\mathcal{Q}$ contains only Boolean combinations of subformulas of φ. Propositional equivalence implies that the number of such combinations is finite: $|\mathcal{Q}| \leq 2^{2^{|\mathsf{sf}(\varphi)|}}$. The language equivalence follows from the definition of $\mathsf{LTL_f}$, and from some classical $\mathsf{LTL_f}$ equivalences. For instance the rule for $\mathtt{tr}(\alpha \,\mathsf{U}\, \beta)$ is based on the equivalence $\mathscr{L}(\alpha \,\mathsf{U}\, \beta) = \mathscr{L}(\beta \vee (\alpha \wedge \mathsf{X}^!(\alpha \,\mathsf{U}\, \beta)))$.

Example 5. Figure 1 is the MTDFA for formula $\Psi_1 \wedge \Psi_2$, presented in Example 1. Many more examples can be found in the associated artifact [24].

The definition of $\mathtt{tr}(\cdot)$ as an MTBDD representation of the set of successors of a state can be thought as a symbolic representation of Antimirov's linear forms [2] for DFA with propositional alphabets. Antimirov presented linear forms as an efficient way to construct all (partial) derivatives at once, without having to iterate over the alphabet. For $\mathsf{LTL_f}$, *formula progressions* [20] are the equivalent of Brozozowski derivatives [13]. Here, $\mathtt{tr}(\cdot)$ computes all formulas progressions at once, without having to iterate over an exponential number of assignments.

Finally, note that while this construction works with any order for $\mathcal{P}$, different orders might produce a different number of MTBDD nodes.

Optimizations The previous definitions can be improved in several ways.

Our implementation of MTBDD actually supports terminals that are the Boolean terminals of standard BDDs as well as the terminals used so far. So we are actually using $\mathsf{MTBDD}(\mathcal{P}, (\mathsf{LTL_f}(\mathcal{P}) \times \mathbb{B}) \cup \mathbb{B})$, and we encode $\mathsf{term}(\mathit{ff}, \bot)$ and $\mathsf{term}(\mathit{tt}, \top)$ directly as $\bot$ and $\top$ respectively. With those changes, $\mathtt{apply2}$ may be modified to shortcut the recursion depending on the values of m_1, m_2, and $\odot$. For instance if $\odot = \wedge$ and $m_1 = \top$, then m_2 can be returned immediately. Such shortcuts may be implemented for $\mathsf{MTBDD}(\mathcal{P}, \mathcal{S} \cup \mathbb{B})$ regardless of the nature of $\mathcal{S}$, so our implementation of MTBDD operations is independent of $\mathsf{LTL_f}$.

When combining terminals during the computation of $\mathtt{tr}$, one has to compute the representative formula $[\alpha_1 \odot \alpha_2]_\equiv$. This can be done by converting α_{1P} and α_{2P} into BDDs, keeping track of such conversions in a hash table. Two propositionally equivalent formulas will have the same BDD representation. While we are looking for a representative formula, we can also use the opportunity to simplify the formula at hand. We use the following very simple rewritings, for patterns that occur naturally in the output of $\mathtt{tr}$:
$(\alpha \,\mathsf{U}\, \beta) \vee \beta \rightsquigarrow \alpha \,\mathsf{U}\, \beta$, $\quad(\alpha \,\mathsf{R}\, \beta) \wedge \beta \rightsquigarrow \alpha \,\mathsf{R}\, \beta$, $\quad(\mathsf{F}\beta) \vee \beta \rightsquigarrow \mathsf{F}\beta$, $\quad(\mathsf{G}\beta) \wedge \beta \rightsquigarrow \mathsf{G}\beta$.

Once $\mathcal{A}_\varphi$ has been built, two states $q, q' \in \mathcal{Q}$ such that $\Delta(q) = \Delta(q')$ can be merged by replacing all occurrences of q' by q in the leaves of Δ.

Example 6. The automaton from Figure 1 has two pairs of states that can be merged. However, if the rule $(\mathsf{G}\beta) \wedge \beta \rightsquigarrow \mathsf{G}\beta$ is applied during the construction, then the occurrence of $(\mathsf{GF}o_2) \wedge (\mathsf{F}o_2)$ will already be replaced by $\mathsf{GF}o_2$, producing the simplified automaton without requiring any merging.

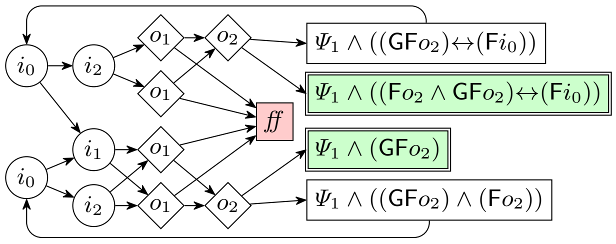
i_0
i_2
o_1
o_2
o_1
ff
i_1
o_1
i_0
i_2
o_1
o_2
$\Psi_1 \wedge ((\mathsf{GF}o_2) \leftrightarrow (\mathsf{F}i_0))$
$\Psi_1 \wedge ((\mathsf{F}o_2 \wedge \mathsf{GF}o_2) \leftrightarrow (\mathsf{F}i_0))$
$\Psi_1 \wedge (\mathsf{GF}o_2)$
$\Psi_1 \wedge ((\mathsf{GF}o_2) \wedge (\mathsf{F}o_2))$

Fig. 2. Interpretation of the MTDFA of Figure 1 as a game with $\mathcal{I} = \{i_0, i_1, i_2\}$, $\mathcal{O} = \{o_1, o_2\}$. Each MTBDD *node* of the MTDFA is viewed as a *vertex* of the game, with terminal of the form $(\alpha, \bot)$ looping back to $\Delta(\alpha)$. Player O decides where to go from diamond and rectangular vertices and wants to reach the green vertices corresponding to accepting terminals. Player I decides where to go from round vertices and wants to reach ff or avoid green vertices.

4 Deciding $\mathsf{LTL_f}$ Realizability

$\mathsf{LTL_f}$ realizability (Def. 3) is solved by reducing the problem to a two-player reachability game where one player decides the input assignments and the other player decides the output assignments [23]. Section 4.1 presents reachability games and how to interpret the MTDFA as a reachability game, and Section 4.2 shows how we can solve the game on-the-fly while constructing it.

4.1 Reachability Games & Backpropagation

Definition 5 (Rechability Game). *A Reachability Game is* $\mathcal{G} = \langle \mathcal{V} = \mathcal{V}_{\mathrm{O}} \uplus \mathcal{V}_{\mathrm{I}}, \mathcal{E}, \mathcal{F}_{\mathrm{O}} \rangle$*, where* $\mathcal{V}$ *is a finite set of* vertices *partitioned to player* output *(abbreviated* O*) and player* input *(abbreviated* I*),* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ *is a finite set of* edges*, and* $\mathcal{F}_{\mathrm{O}} \subseteq \mathcal{V}$ *is the set of target states. Let* $\mathcal{E}(v) = \{(v, v') \mid (v, v') \in \mathcal{E}\}$*. This graph is also referred to as the game* arena*.*

A strategy for player O *is a cycle-free subgraph* $\langle W, \sigma \rangle \subseteq \langle \mathcal{V}, \mathcal{E} \rangle$ *such that (a) for every* $v \in W$ *we have* $v \in \mathcal{F}_{\mathrm{O}}$ *or* $\mathcal{E}(v) \cap \sigma \neq \emptyset$ *and (b) if* $v \in W \cap \mathcal{V}_{\mathrm{I}}$ *then* $\mathcal{E}(v) \subseteq \sigma$*. A vertex* v *is winning for* O *if* $v \in W$ *for some strategy* $\langle W, \sigma \rangle$*.*

Such a reachability game can be solved by backpropagation identifying the maximal set W in a strategy. Namely, start from $W = \mathcal{F}_{\mathrm{O}}$. Then W is iteratively augmented with every vertex in $\mathcal{V}_{\mathrm{O}}$ that has some edge to W, and every (non dead-end) vertex in $\mathcal{V}_{\mathrm{I}}$ whose edges all lead to W. At the end of this backpropagation, which can be performed in linear time [35, Theorem 3.1.2], every vertex in W is winning for O, and every vertex outside W is losing for O. Notice that every dead-end that is not in $\mathcal{F}_{\mathrm{O}}$ cannot be winning. It follows that we can identify some (but not necessarily all) vertices that are losing by setting L as the set of all dead-ends and adding to L every $\mathcal{V}_{\mathrm{I}}$ vertex that has some edge to L and every $\mathcal{V}_{\mathrm{O}}$ vertex whose edges all lead to L.

Let $\mathcal{A}_\varphi = \langle \mathcal{Q}, \mathcal{I} \uplus \mathcal{O}, \iota, \Delta \rangle$ be a translation of $\varphi \in \mathsf{LTL_f}(\mathcal{I} \uplus \mathcal{O})$ (per Th. 1) such that variables of $\mathcal{I}$ appear before $\mathcal{O}$ in the MTBDD encoding of Δ.

Definition 6 (Realizability Game). *We define the reachability game $\mathcal{G}_\varphi = \langle \mathcal{V} = \mathcal{V}_\text{I} \uplus \mathcal{V}_\text{O}, \mathcal{E}, \mathcal{F}_\text{O}\rangle$ in which $\mathcal{V} \subseteq \mathsf{MTBDD}(\mathcal{I} \uplus \mathcal{O})$ corresponds the set of nodes that appear in the MTBDD encoding of Δ. $\mathcal{V}_\text{O}$ contains all nodes (p, ℓ, h) such that $p \in \mathcal{O}$ or $p = \infty$ (terminals), and $\mathcal{V}_\text{I}$ contains those with $p \in \mathcal{I}$. The edges $\mathcal{E}$ follows the structure of Δ, i.e., if $\mathcal{A}_\varphi$ has a node $r = (p, \ell, h)$, then $\{(r, \ell), (r, h)\} \subseteq \mathcal{E}$. Additionally, for any terminal $t = (\infty, (\alpha, \bot), \infty)$ such that $\alpha \neq f\!f$, $\mathcal{E}$ contains the edge $(t, \Delta(\alpha))$. Finally, $\mathcal{F}_\text{O}$ is the set of accepting terminals, i.e., nodes of the form $(\infty, (\alpha, \top), \infty)$.*

Theorem 2. *Vertex $\Delta(\iota)$ is winning for* O *in $\mathcal{G}_\varphi$ iff φ is Mealy-realizable.*

Moore realizability can be checked similarly by changing the order of $\mathcal{I}$ and $\mathcal{O}$ in the MTBDD encoding of Δ.

Example 7. Figure 2 shows how to interpret the MTDFA of Figure 1 as a game, by turning each MTBDD node into a game vertex. The player owning each vertex is chosen according to the variable that labels it. Vertices corresponding to accepting terminals become winning targets for the output player, so the game stops once they are reached. Solving this game will find every internal node as winning for O, so the corresponding formula is Mealy-realizable.

The difference with DFA games [23,20,28,51] is that instead of having player I select all input signals at once, and then player O select all output signals at once, our game proceeds by selecting one signal at a time. Sharing nodes that represent identical partial assignments contributes to the scalability of our approach.

4.2 Solving Realizability On-the-fly

We now show how to construct and solve $\mathcal{G}_\varphi$ on-the-fly, for better efficiency. The construction is easier to study in two parts: (1) the on-the-fly solving of reachability games, based on backpropagation, and (2) the incremental construction of $\mathcal{G}_\varphi$, done with a forward exploration of a subset of the MTDFA for φ.

Algorithm 1 presents the first part: a set of functions for constructing a game arena incrementally, while performing the linear-time backpropagation algorithm on-the-fly. At all points during this construction, the winning status of a vertex (*winner*$[x]$) will be one of O (player O can force the play to reach $\mathcal{F}_\text{O}$, i.e., the vertex belongs to W), I (player I can force the play to avoid $\mathcal{F}_\text{O}$, i.e., the vertex belongs to L), or U (undetermined yet), and the algorithm will backpropagate both O and I. At the end of the construction, all vertices with status U will be considered as winning for I. Like in the standard algorithm for solving reachability games [35, Th. 3.1.2] each state uses a counter (*count*, lines 7,15) to track the number of its undeterminated successors. When a vertex x is marked as winning for player w by calling `set_winner(`*x,w*`)`, an undeterminated predecessor p has its counter decreased (line 15), and p can be marked as winning for w (line 16) if either vertex p is owned by w (player w can choose to go to x) or the counter dropped to 0 (meaning that all choices at p were winning for w).

To solve the game while it is constructed, we *freeze* vertices. A vertex should be frozen after all its successors have been introduced with `new_edge`. The

```
   var: $owner[]$;                      // map each vertex to one of $\{\texttt{O}, \texttt{I}\}$
   var: $pred[]$;           // map vertices to sets of predecessor vertices
   var: $count[]$;        // map vertices to # of undeterminated successors
   var: $winner[]$;                     // map vertices to one of $\{\texttt{O}, \texttt{I}, \texttt{U}\}$
   var: $frozen[]$;       // map vertices to their frozen status (a Boolean)
 1 Function new_vertex($x \in \mathcal{V}$, $own \in \{\texttt{O}, \texttt{I}\}$) // new vertex owned by $own$
 2     $owner[x] \leftarrow own$; $pred[x] \leftarrow \emptyset$; $count[x] \leftarrow 0$; $frozen[x] \leftarrow \bot$;
 3     $winner[x] \leftarrow \texttt{U}$; // undeterminated winner
 4 Function new_edge($src \in \mathcal{V}$, $dst \in \mathcal{V}$)
 5     assert ($frozen[src] = \bot$);
 6     if $winner[dst] = \texttt{U}$ then
 7         $count[src] \leftarrow count[src] + 1$; $pred[dst] \leftarrow pred[dst] \cup \{src\}$;
 8     else if $winner[dst] = owner[src]$ then set_winner($src$, $owner[src]$);
       // ignore the edge otherwise, it will never be used
 9 Function freeze_vertex($x \in \mathcal{V}$) // promise not to add more successors
10     $frozen[x] \leftarrow \top$;              // next line, we assume $\neg\texttt{I} = \texttt{O}$ and $\neg\texttt{O} = \texttt{I}$
11     if $winner[x] = \texttt{U} \land count[n] = 0$ then set_winner($x$, $\neg owner[x]$) ;
12 Function set_winner($x \in \mathcal{V}$, $w \in \{\texttt{O}, \texttt{I}\}$) // with linear backprop.
13     assert ($winner[x] = \texttt{U}$); $winner[x] \leftarrow w$;
14     foreach $p \in pred[x]$ such that $winner[p] = \texttt{U}$ do
15         $count[p] \leftarrow count[p] - 1$;
16         if $owner[p] = w \lor (count[p] = 0 \land frozen[p])$ then set_winner($p$, $w$);
```

Algorithm 1: API for solving a reachability game on-the-fly. Construct the game arena with `new_vertex` and `new_edge`. Once all successors of a vertex have been connected, call `freeze_vertex`. Call `set_winner` at any point to designate vertices winning for one player.

counter dropping to 0 is only checked on frozen vertices (lines 11, 16) since it is only meaningful if all successors of a vertex are known.

Algorithm 2 is the second part. It shows how to build $\mathcal{G}_\varphi$ incrementally. It translates the states α of the corresponding MTDFA one at a time, and uses the functions of Algorithm 1 to turn each node of $\mathsf{tr}(\alpha)$ into a vertex of the game. Since the functions of Algorithm 1 update the winning status of the states as soon as possible, Algorithm 2 can use that to cut parts of the exploration.

Instead of using $\Delta(\varphi) = \mathsf{tr}(\varphi)$ as initial vertex of the game, as in Theorem 2, we consider $init = \mathsf{term}(\varphi, \bot)$ as initial vertex (line 3): this makes no theoretical difference, since $\mathsf{term}(\varphi, \bot)$ has $\mathsf{tr}(\varphi)$ as unique successor. Lines 5,9–11,13, and 32 implements the exploration of all the $\mathsf{LTL_f}$ formulas α that would label the states of the MTDFA for φ (as needed to implement Theorem 1). The actual order in which formulas are removed from *todo* on line 11 is free. (We found out that handling *todo* as a queue to implement a BFS exploration worked marginally better than using it as a stack to do a DFS-like exploration, so we use a BFS in practice.)

Each α is translated into an MTBDD $\mathsf{tr}(\alpha)$ representing its possible successors. The constructed game should have one vertex per MTBDD node in $\mathsf{tr}(\alpha)$. Those vertices are created in the inner **while** loop (lines 19–31). Function `declare_vertex` is used to assign the correct owner to each new node according to its decision variable (as in Def. 6) as well as adding those nodes to the

```
1  Function realizability($\varphi \in \mathsf{LTL_f}(\mathcal{I} \uplus \mathcal{O})$)
2      configure the MTBDD library to put variables in $\mathcal{I}$ before those in $\mathcal{O}$;
3      $init \leftarrow \mathsf{term}(\varphi, \bot)$; new_vertex($init$, I);
4      $\mathcal{V} \leftarrow \{init\}$; // nodes created as game vertices
5      $\mathcal{Q} \leftarrow \emptyset$; // $\mathsf{LTL_f}$ formulas processed by main loop on line 10
6      Function declare_vertex($r \in \mathsf{MTBDD}(\mathcal{I} \uplus \mathcal{O}, \mathsf{LTL_f}(\mathcal{I} \uplus \mathcal{O}) \times \mathbb{B})$)
7          $(p, \ell, h) \leftarrow r$; if $p = \infty \lor p \in \mathcal{I}$ then $own \leftarrow$ I else $own \leftarrow$ O;
8          new_vertex($r$, $own$); $\mathcal{V} \leftarrow \mathcal{V} \cup \{r\}$; $to\_encode \leftarrow to\_encode \cup \{r\}$;
9      $todo \leftarrow \{\varphi\}$;
10     while $todo \neq \emptyset \land winner[init] =$ U do
11         $\alpha \leftarrow todo.\mathsf{pop\_any}()$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\alpha\}$;
12         [optional: add one-step (un)realizability check here, see Sec. 5];
13         $a \leftarrow \mathsf{term}(\alpha, \bot)$; $m \leftarrow \mathsf{tr}(\alpha)$;
14         if $m \in \mathcal{V}$ then // $m$ has already been encoded
15             new_edge($a, m$); freeze_vertex($a$); continue to line 10;
16         $to\_encode \leftarrow \emptyset$; $leaves \leftarrow \emptyset$;
17         declare_vertex($m$); new_edge($a, m$); freeze_vertex($a$);
18         if $winner[a] \neq$ U then continue to line 10;
19         while $to\_encode \neq \emptyset$ do
20             $r \leftarrow to\_encode.\mathsf{pop\_any}()$;
21             $(p, \ell, h) \leftarrow r$;
22             if $p = \infty$ then // this is a terminal labeled by $\ell$
23                 $(\beta, b) \leftarrow \ell$;
24                 if $b$ then set_winner($r$, O);
25                 else if $\beta = f\!f$ then set_winner($r$, I);
26                 else if $\beta \notin \mathcal{Q}$ then $leaves \leftarrow leaves \cup \{\beta\}$;
27             else
28                 if $\ell \notin \mathcal{V}$ then declare_vertex($\ell$);
29                 if $h \notin \mathcal{V}$ then declare_vertex($h$);
30                 new_edge($r, \ell$); new_edge($r, h$); freeze_vertex($r$);
31             if $winner[a] \neq$ U then continue to line 10;
32         $todo \leftarrow todo \cup leaves$;
33     return $winner[init] =$ O;
```

Algorithm 2: On-the-fly realizability check with Mealy semantics (for Moore semantics, swap the order of $\mathcal{I}$ and $\mathcal{O}$ on the first line).

to_encode set processed by this inner loop. Terminal nodes are either marked as winning for one of the players (lines 24–25) or stored in *leaves* (line 26).

Since connecting game vertices may backpropagate their winning status, the encoding loop can terminate early whenever the vertex associated to $\mathsf{term}(\alpha, \bot)$ becomes determined (lines 18 and 31). If that vertex is not determined, the *leaves* of α are added to *todo* (line 32) for further exploration.

The entire construction can also stop as soon as the initial vertex is determined (line 10). However, if the algorithm terminates with $winning[init] =$ U, it still means that O cannot reach its targets. Therefore, as tested by line 33, formula φ is realizable iff $winning[init] =$ O in the end.

Theorem 3. *Algorithm 2 returns tt iff φ is Mealy-realizable.*

5 Implementation and Evaluation

Our algorithms have been implemented in Spot [25], after extending its fork of BuDDy [40] to support MTBDDs. The release of Spot 2.14 distributes two new command-line tools: `ltlf2dfa` and `ltlfsynt`, implementing translation from $\mathsf{LTL_f}$ to MTDFA, and solving $\mathsf{LTL_f}$ synthesis. We describe and evaluate `ltlfsynt` in the following.

Preprocessing Before executing Algorithm 2, we use a few preprocessing techniques to simplify the problem. We remove variables that always have the same polarity in the specification (a simplification used also by Strix [48]), and we decompose the specifications into output-disjoint sub-specifications that can be solved independently [31]. A specification such as $\Psi_1 \wedge \Psi_2$, from Example 1, is not solved directly as demonstrated here, but split into two output-disjoint specifications Ψ_1 and Ψ_2 that are solved separately. Finally, we also simplify $\mathsf{LTL_f}$ formulas using very simple rewriting rules such as $\mathsf{X}(\alpha) \wedge \mathsf{X}(\beta) \rightsquigarrow \mathsf{X}(\alpha \wedge \beta)$ that reduce the number of MTBDD operations required during translation.

One-step (un)realizability checks An additional optimization consists in performing one-step realizability and one-step unrealizability checks in Algorithm 2. The principle is to transform the formula α into two smaller Boolean formulas α_r and α_u, such that if α_r is realizable it implies that α is realizable, and if α_u is unrealizable it implies that α is unrealizable [50, Theorems 2–3]. Those Boolean formulas can be translated to BDDs for which realizability can be checked by quantification. On success, it avoids the translation of the larger formula α. The simple formula $\Psi_1 \wedge \Psi_2$ of our running example is actually one-step realizable.

Synthesis After deciding realizability, `ltlfsynt` is able to extract a strategy from the solved game in the form of a Mealy machine, and encode that into an And-Invert Graph (AIG) [10]: the expected output of the Synthesis Competition for the $\mathsf{LTL_f}$ synthesis tracks. The conversion from Mealy to AIG reuses prior work [46,47] developed for Spot's LTL (not $\mathsf{LTL_f}$) synthesis tool. We do not detail nor evaluate these extra steps here due to lack of space.

Evaluation We evaluated the task of deciding $\mathsf{LTL_f}$ reachability over specifications from the Synthesis Competition [38]. We took all `tlsf-fin` specifications from SyntComp's repository, excluded some duplicate specifications as well as some specifications that were too large to be solved by any tool, and converted the specifications from TLSF v1.2 [37] to $\mathsf{LTL_f}$ using `syfco` [37].

We used BenchExec 3.22 [9] to track time and memory usage of each tool. Tasks were run on a Core i7-3770 with *Turbo Boost* disabled, and frequency scaled down to 1.6GHz to prevent CPU throttling. The computer has 4 physical cores and 16GB of memory. BenchExec was configured to run up to 3 tasks in parallel with a memory limit of 4GB per task, and a time limit of 15 minutes.

Figure 3 compares five configurations of `ltlfsynt` against seven other tools. We verified that all tools were in agreement. Lydia 0.1.3 [19], SyftMax (or Syft 2.0) [52] and LydiaSyft 0.1.0-alpha [29] are all using Mona to construct a DFA by composition; they then solve the resulting game symbolically after encoding

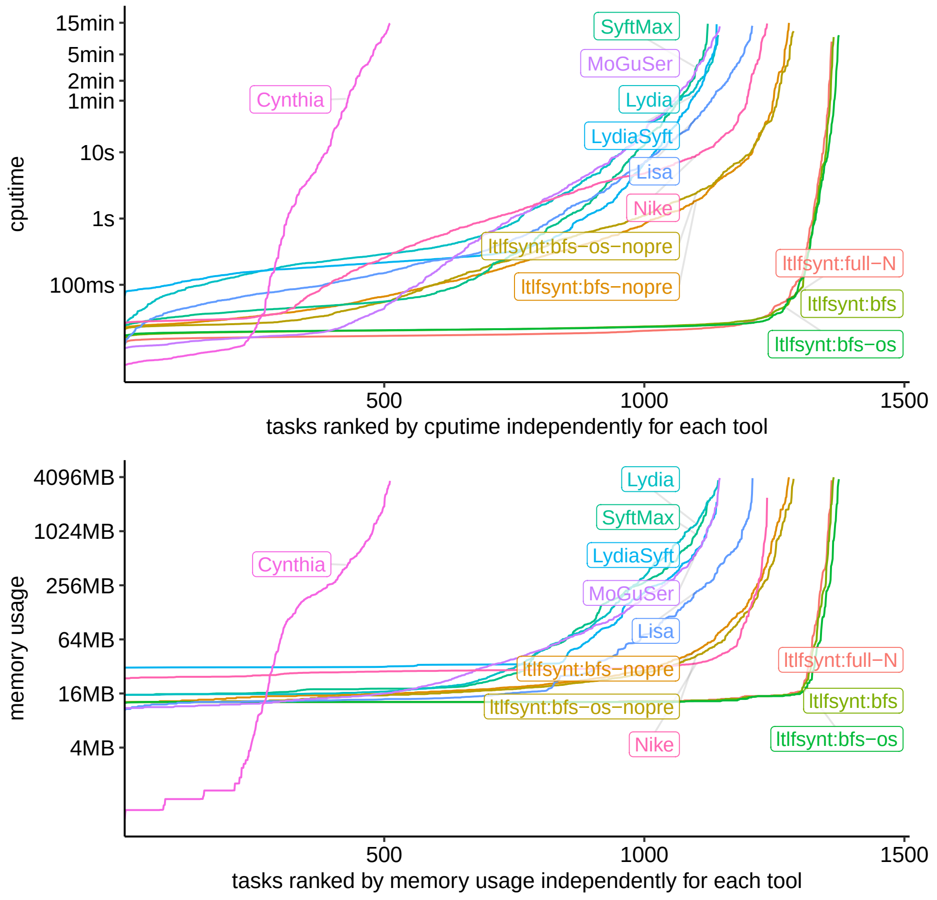

Fig. 3. Cactus plots comparing time and memory usage of different configurations.

it using BDDs. Lisa [8] uses a hybrid compositional construction, mixing explicit compositions (using Spot), with symbolic compositions (using BuDDy), solving the game symbolically in the end. Cynthia 0.1.0 [20], Nike 0.1.0 [28], and MoGuSer [51] all use an on-the-fly construction of a DFA game that they solve via forward exploration with backpropagation, but they do not implement backpropagation in linear time, as we do. Yet, the costly part of synthesis is game generation, not solving. Cynthia uses SDDs [17] to compute successors and represent states, while Nike and MoGuSer use SAT-based techniques to compute successors and BDDs to represent states. Nike, Lisa, and LydiaSyft were the top-3 contenders of the $\mathsf{LTL_f}$ track of SyntComp in 2023 and 2024.

Configuration `ltlfsynt:bfs-nopre` corresponds to Algorithm 2 were *todo* is a queue: it already solves more cases than all other tested tools. Suffix `-nopre` indicates that preprocessings of the specification are disabled (this makes comparison fairer, since other tools have no such preprocessings). The version with preprocessings enabled is simply called `ltlfsynt:bfs`. Variants with "`-os`" adds the one-step (un)realizability checks that LydiaSyft, Cynthia, and Nike also perform.

We also include a configuration `ltlfsynt:full-N` that corresponds to first translating the specification into a MTDFA using Theorem 1, and then solving the game by linear propagation. The difference between `ltlfsynt:full` and `ltlfsynt:bfs` shows the gain obtained with the on-the-fly translation: although that look small in the cactus plot, it is important in some specifications.

Data Availability Statement Implementation, supporting scripts, detailed analysis of this benchmark, and additional examples are archived on Zenodo [24].

6 Conclusion

We have presented the implementation of `ltlfsynt`, and evaluated it to be faster at deciding $\mathsf{LTL_f}$ realizability than seven existing tools, including the winners of SyntComp'24. The implementation uses a direct and efficient translation from $\mathsf{LTL_f}$ to DFA represented by MTBDDs, which can then be solved as a game played directly on the structure of the MTBDDs. The two constructions (translation and game solving) are performed together on-the-fly, to allow early termination.

Although `ltlsynt` also includes a preliminary implementation of $\mathsf{LTL_f}$ synthesis of And-Inverter graphs, we leave it as future work to document it and ensure its correctness.

Finally, the need for solving a reachability game while it is discovered also occurs in other equivalent contexts such as HornSAT, where linear algorithms that do not use "counters" and "predecessors" (unlike ours) have been developed [41]. Using such algorithms might improve our solution by saving memory.

References

1. Andersen, H.R.: An introduction to binary decision diagrams. Lecture notes for Efficient Algorithms and Programs, Fall 1999 (1999), https://web.archive.org/web/20090530154634/http://www.itu.dk:80/people/hra/bdd-eap.pdf
2. Antimirov, V.: Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science **155**(2), 291–319 (Mar 1996). https://doi.org/10.1016/0304-3975(95)00182-4
3. Bacchus, F., Kabanza, F.: Planning for temporally extended goals. Annals of Mathematics and Artificial Intelligence **22**, 5–27 (1998). https://doi.org/10.1023/A:1018985923441
4. Bacchus, F., Kabanza, F.: Using temporal logics to express search control knowledge for planning. Artificial Intelligence **116**(1–2), 123–191 (2000). https://doi.org/10.1016/S0004-3702(99)00071-5
5. Bahar, R.I., Frohm, E.A., Gaona, C.M., Hachtel, G.D., Macii, E., Pardo, A., Somenzi, F.: Algebraic decision diagrams and their applications. In: Proceedings of 1993 International Conference on Computer Aided Design (ICCAD'93). pp. 188–191. IEEE Computer Society Press (Nov 1993). https://doi.org/10.1109/ICCAD.1993.580054
6. Baier, J.A., Fritz, C., McIlraith, S.A.: Exploiting procedural domain control knowledge in state-of-the-art planners. In: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'07). pp. 26–33. AAAI (2007), https://aaai.org/papers/icaps-07-004

7. Baier, J.A., McIlraith, S.A.: Planning with first-order temporally extended goals using heuristic search. In: Proceedings of the 21st national conference on Artificial intelligence (AAAI'06). pp. 788–795. AAAI Press (2006). https://doi.org/10.5555/1597538.1597664
8. Bansal, S., Li, Y., Tabajara, L.M., Vardi, M.Y.: Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In: Proceedings of the 34th national conference on Artificial intelligence (AAAI'20). pp. 9766–9774. AAAI Press (2020). https://doi.org/10.1609/AAAI.V34I06.6528
9. Beyer, D., Löwe, S., Wendler, P.: Reliable benchmarking: requirements and solutions. International Journal on Software Tools for Technology Transfer **21**, 1–29 (Feb 2019). https://doi.org/10.1007/s10009-017-0469-y
10. Biere, A., Heljanko, K., Wieringa, S.: AIGER 1.9 and beyond. Tech. Rep. 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria (2011), https://fmv.jku.at/aiger/
11. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **35**(8), 677–691 (Aug 1986). https://doi.org/10.1109/TC.1986.1676819
12. Bryant, R.E.: Symbolic boolean manipulation with ordered binary-decision diagrams. ACM Comput. Surv. **24**(3), 293–318 (Sep 1992). https://doi.org/10.1145/136035.136043
13. Brzozowski, J.A.: Derivatives of regular expressions. Journal of the ACM **11**(4), 481–494 (Oct 1964). https://doi.org/10.1145/321239.321249
14. Calvanese, D., De Giacomo, G., Vardi, M.Y.: Reasoning about actions and planning in LTL action theories. In: Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning (KR'02). pp. 593–602. Morgan Kaufmann (2002). https://doi.org/10.5555/3087093.3087142
15. Camacho, A., Bienvenu, M., McIlraith, S.A.: Towards a unified view of AI planning and reactive synthesis. In: Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'19). pp. 58–67. AAAI Press (2019). https://doi.org/10.1609/icaps.v29i1.3460
16. Cimatti, A., Pistore, M., Roveri, M., Traverso, P.: Weak, strong, and strong cyclic planning via symbolic model checking. Artificial Intelligence **147**(1–2), 35–84 (2003). https://doi.org/10.1016/S0004-3702(02)00374-0
17. Darwiche, A.: SDD: A new canonical representation of propositional knowledge bases. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence. pp. 819–826. AAAI Press (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-143
18. De Giacomo, G., Favorito, M.: Compositional approach to translate LTL$_f$/LDL$_f$ into deterministic finite automata. In: Proceedings of the 31st International Conference on Automated Planning and Scheduling (ICAPS'21). pp. 122–130 (2021). https://doi.org/10.1609/icaps.v31i1.15954
19. De Giacomo, G., Favorito, M.: Compositional approach to translate LTLf/LDLf into deterministic finite automata. In: Biundo, S., Do, M., Goldman, R., Katz, M., Yang, Q., Zhuo, H.H. (eds.) Proceedings of the 31'st International Conference on Automated Planning and Scheduling (ICAPS'21). pp. 122–130. AAAI Press (Aug 2021). https://doi.org/10.1609/icaps.v31i1.15954
20. De Giacomo, G., Favorito, M., Li, J., Vardi, M.Y., Xiao, S., Zhu, S.: LTLf synthesis as AND-OR graph search: Knowledge compilation at work. In: Raedt, L.D. (ed.) Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI'22). pp. 2591–2598. International Joint Conferences on Artificial Intelligence Organization (Jul 2022). https://doi.org/10.24963/ijcai.2022/359

21. De Giacomo, G., Rubin, S.: Automata-theoretic foundations of fond planning for LTL$_f$/LDL$_f$ goals. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18). pp. 4729–4735 (2018). https://doi.org/10.24963/ijcai.2018/657
22. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI'13). pp. 854–860. IJCAI'13, AAAI Press (Aug 2013). https://doi.org/10.5555/2540128.2540252
23. De Giacomo, G., Vardi, M.Y.: Synthesis for LTL and LDL on finite traces. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 1558–1564. AAAI Press (2015). https://doi.org/10.5555/2832415.2832466
24. Duret-Lutz, A.: Supporting material for "Engineering an LTLf Synthetizer Tool" (2025). https://doi.org/10.5281/zenodo.15752968
25. Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.: From Spot 2.0 to Spot 2.10: What's new? In: Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22). Lecture Notes in Computer Science, vol. 13372, pp. 174–187. Springer (Aug 2022). https://doi.org/10.1007/978-3-031-13188-2_9
26. Ehlers, R., Lafortune, S., Tripakis, S., Vardi, M.Y.: Supervisory control and reactive synthesis: a comparative introduction. Discrete Event Dynamic Systems **27**(2), 209–260 (2017). https://doi.org/10.1007/s10626-015-0223-0
27. Esparza, J., Křetínský, J., Sickert, S.: One theorem to rule them all: A unified translation of LTL into ω-automata. In: Dawar, A., Grädel, E. (eds.) Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18). pp. 384–393. ACM (2018). https://doi.org/10.1145/3209108.3209161
28. Favorito, M.: Forward LTLf synthesis: DPLL at work. In: Benedictis, R.D., Castiglioni, M., Ferraioli, D., Malvone, V., Maratea, M., Scala, E., Serafini, L., Serina, I., Tosello, E., Umbrico, A., Vallati, M. (eds.) Proceedings of the 30th Workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion (RCRA'23). CEUR Workshop Proceedings, vol. 3585 (2023), https://ceur-ws.org/Vol-3585/paper7_RCRA4.pdf
29. Favorito, M., Zhu, S.: LydiaSyft: A compositional symbolic synthesis framework for LTLf specifications. In: Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'25). Lecture Notes in Computer Science, vol. 15696, pp. 295–302. Springer (May 2025). https://doi.org/10.1007/978-3-031-90643-5_15
30. Finkbeiner, B.: Synthesis of reactive systems. In: Javier Esparza, Orna Grumberg, S.S. (ed.) Dependable Software Systems Engineering, NATO Science for Peace and Security Series — D: Information and Communication Security, vol. 45, pp. 72–98. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-72
31. Finkbeiner, B., Geier, G., Passing, N.: Specification decomposition for reactive synthesis. In: Proceedings for the 13th NASA Formal Methods Symposium (NFM'21). Lecture Notes in Computer Science, vol. 12673, pp. 113–130. Springer (2021). https://doi.org/10.1007/978-3-030-76384-8_8
32. Fujita, M., McGeer, P.C., Yang, J.C.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal Methods in System Design **10**(2/3), 149–169 (1997). https://doi.org/10.1023/A:1008647823331

33. Gabbay, D., Pnueli, A., Shelah, S., Stavi, J.: On the temporal analysis of fairness. In: Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL'80). pp. 163–173. Association for Computing Machinery (1980). https://doi.org/doi.org/10.1145/567446.5674
34. Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artificial Intelligence **173**(5–6), 619–668 (2009). https://doi.org/10.1016/j.artint.2008.10.012
35. Grädel, E.: Finite model theory and descriptive complexity. In: Finite Model Theory and Its Applications, chap. 3, pp. 125–230. Texts in Theoretical Computer Science an EATCS Series, Springer Berlin Heidelberg, Berlin, Heidelberg (2007). https://doi.org/10.1007/3-540-68804-8_3
36. Henriksen, J.G., Jensen, J., Jørgensen, M., Klarlund, N., Paige, R., Rauhe, T., Sandholm, A.: Mona: Monadic second-order logic in practice. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) First International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95). pp. 89–110. Springer Berlin Heidelberg (1995). https://doi.org/10.1007/3-540-60630-0_5
37. Jacobs, S., Perez, G.A., Schlehuber-Caissier, P.: The temporal logic synthesis format TLSF v1.2. arXiV (2023). https://doi.org/10.48550/arXiv.2303.03839
38. Jacobs, S., Perez, G.A., Abraham, R., Bruyère, V., Cadilhac, M., Colange, M., Delfosse, C., van Dijk, T., Duret-Lutz, A., Faymonville, P., Finkbeiner, B., Khalimov, A., Klein, F., Luttenberger, M., Meyer, K.J., Michaud, T., Pommellet, A., Renkin, F., Schlehuber-Caissier, P., Sakr, M., Sickert, S., Staquet, G., Tamines, C., Tentrup, L., Walker, A.: The reactive synthesis competition (SYNTCOMP): 2018–2021. arXiV (Jun 2022). https://doi.org/10.48550/ARXIV.2206.00251
39. Klarlund, N., Møller, A.: MONA version 1.4, user manual. Tech. rep., BRICS (Jul 2001), https://www.brics.dk/mona/mona14.pdf
40. Lind-Nielsen, J.: BuDDy: A binary decision diagram package. User's manual. (1999), https://web.archive.org/web/20040402015529/http://www.itu.dk/research/buddy/
41. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) Proceedings of the 25th International Colloquium on Automata, Languages and Programming (ICALP'98). pp. 53–66. Springer Berlin Heidelberg (1998). https://doi.org/10.1007/BFb0055035
42. Long, D.: BDD library. source archive, https://www.cs.cmu.edu/~modelcheck/bdd.html
43. Minato, S.i.: Representation of Multi-Valued Functions, pp. 39–47. Springer US, Boston, MA (1996). https://doi.org/10.1007/978-1-4613-1303-8_4
44. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Proceedings of the 7th international conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'06). Lecture Notes in Computer Science, vol. 3855, pp. 364–380. Springer (2006). https://doi.org/10.1007/11609773_24
45. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL'89). Association for Computing Machinery (1989). https://doi.org/10.1145/75277.75293
46. Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., Pommellet, A.: Effective reductions of Mealy machines. In: Proceedings of the 42nd International Conference on Formal Techniques for Distributed Objects, Components, and Systems

(FORTE'22). Lecture Notes in Computer Science, vol. 13273, pp. 170–187. Springer (Jun 2022). https://doi.org/10.1007/978-3-031-08679-3_8

47. Renkin, F., Schlehuber-Caissier, P., Duret-Lutz, A., Pommellet, A.: Dissecting `ltlsynt`. Formal Methods in System Design (2023). https://doi.org/10.1007/s10703-022-00407-6
48. Sickert, S., Meyer, P.: Modernizing strix (2021), https://www7.in.tum.de/~sickert/publications/MeyerS21.pdf
49. Somenzi, F.: CUDD: CU Decision Diagram package release 3.0.0 (Dec 2015), https://web.archive.org/web/20171208230728/http://vlsi.colorado.edu/~fabio/CUDD/cudd.pdf
50. Xiao, S., Li, J., Zhu, S., Shi, Y., Pu, G., Vardi, M.: On-the-fly synthesis for LTL over finite traces. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI'21, Technical Track 7). pp. 6530–6537 (May 2021). https://doi.org/10.1609/aaai.v35i7.16809
51. Xiao, S., Li, Y., Huang, X., Xu, Y., Li, J., Pu, G., Strichman, O., Vardi, M.Y.: Model-guided synthesis for LTL over finite traces. In: Proceedings of the 25th International Conference on Verification, Model Checking, and Abstract Interpretation. Lecture Notes in Computer Science, vol. 14499, pp. 186–207. Springer (2024). https://doi.org/10.1007/978-3-031-50524-9_9
52. Zhu, S., De Giacomo, G.: Synthesis of maximally permissive strategies for LTLf specifications. In: Raedt, L.D. (ed.) Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI'22). pp. 2783–2789. ijcai.org (Jul 2022). https://doi.org/10.24963/IJCAI.2022/386
53. Zhu, S., Tabajara, L.M., Li, J., Pu, G., Vardi, M.Y.: Symbolic LTLf synthesis. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17). pp. 1362–1369 (2017). https://doi.org/10.24963/ijcai.2017/189