Encodings of Bounded Synthesis*

Peter Faymonville[1], Bernd Finkbeiner[1], Markus N. Rabe[2], and Leander Tentrup[1]

[1] Saarland University
[2] University of California, Berkeley

Abstract. The reactive synthesis problem is to compute a system satisfying a given specification in temporal logic. Bounded synthesis is the approach to bound the maximum size of the system that we accept as a solution to the reactive synthesis problem. As a result, bounded synthesis is decidable whenever the corresponding verification problem is decidable, and can be applied in settings where classic synthesis fails, such as in the synthesis of distributed systems. In this paper, we study the constraint solving problem behind bounded synthesis. We consider different reductions of the bounded synthesis problem of linear-time temporal logic (LTL) to constraint systems given as boolean formulas (SAT), quantified boolean formulas (QBF), and dependency quantified boolean formulas (DQBF). The reductions represent different trade-offs between conciseness and algorithmic efficiency. In the SAT encoding, both inputs and states of the system are represented explicitly; in QBF, inputs are symbolic and states are explicit; in DQBF, both inputs and states are symbolic. We evaluate the encodings systematically using benchmarks from the reactive synthesis competition (SYNTCOMP) and state-of-the-art solvers. Our key, and perhaps surprising, empirical finding is that QBF clearly dominates both SAT and DQBF.

1 Introduction

There has been a recent surge of new algorithms and tools for the synthesis of reactive systems from temporal specifications [5,9,14,15,19]. Roughly, these approaches can be classified into two categories: *game-based synthesis* [8] translates the specification into an deterministic automaton and subsequently determines the winner in a game played on the state graph of this automaton; *bounded synthesis* [25] constructs a constraint system that characterizes all systems, up to a fixed bound on the size of the system, that satisfy the specification.

The success of game-based synthesis is largely due to the fact that it is often possible to represent and analyze the game arena symbolically, in particular with BDDs (cf. [19]). As a result, it has been possible to scale synthesis to realistic benchmarks such as the AMBA bus protocol [3]. However, because the deterministic automaton often contains many more states than are needed by the

* Supported by the European Research Council (ERC) Grant OSARES (No. 683300).

implementation, the synthesized systems are often unnecessarily (and impractically) large (cf. [11]). This problem is addressed by bounded synthesis, where an iteratively growing bound can ensure that the synthesized system is actually the smallest possible realization of the specification. However, bounded synthesis has not yet reached the same scalability as game-based synthesis. A likely explanation for the phenomenon is that the encoding of bounded synthesis into the constraint system is "less symbolic" than the BDD-based representation of the game arena. Even though bounded synthesis tools typically use powerful SMT solvers, a careful study of the standard encoding shows that both the states of the synthesized system and its inputs are enumerated explicitly [14].

The question arises whether it is the encodings that need to be improved, or whether the poor scalability points to a more fundamental flaw in the underlying solver technology. To answer this question, we reduce the bounded synthesis problem of linear-time temporal logic (LTL) to constraint systems given as boolean formulas (SAT), quantified boolean formulas (QBF), and dependency quantified boolean formulas (DQBF). The reductions are landmarks on the spectrum of symbolic vs. explicit encodings. All encodings represent the synthesized system in terms of its transition function, which identifies the successor state in terms of the current state and the input, and additionally in terms of an output function, which identifies the output signals in terms of the current state and the input, and annotation functions, which relate the states of the system to the states of a universal automaton representing the specification.

In the SAT encoding of the transition function, a separate boolean variable is used for every combination of a source state, an input signal, and a target state. The encoding is thus explicit in both the state and the input. In the QBF encoding, a universal quantification over the inputs is added, so that the encoding becomes symbolic in the inputs, while staying explicit in the states. Quantifying universally over the states, just like over the input signals, is not possible in QBF because the states occur twice in the transition function, as source and as target. Separate quantifiers over sources and targets would allow for models where, for example, the value of the output function differs, even though both the source state and the input are the same. In DQBF we can avoid such artifacts and obtain a "fully symbolic" encoding in both the states and the input.

We evaluate the encodings systematically using benchmarks from the reactive synthesis competition (SYNTCOMP) and state-of-the-art solvers. Our empirical finding is that QBF clearly dominates both SAT and DQBF. While the dominance of QBF over SAT fits with our intuition that a more symbolic encoding provides opportunities for optimization in the solver, the dominance of QBF over DQBF is surprising. This indicates that with the currently available solvers, the most symbolic encoding (DQBF) is *not* the best choice. Of course, with better DQBF solvers, this may change: our benchmarks identify opportunities for improvement for current DQBF solvers.

Related Work. The game-based approach to the synthesis of reactive systems dates back to Büchi and Landweber's seminal 1969 paper [8]. Modern implementations of this approach exploit symbolic representations of the game arena, using BDDs (cf. [19]) or decision procedures for the satisfiability of Boolean formulas (SAT-, QBF- and DQBF-solvers). We refer to [4] for a detailed comparison of the different methods.

Bounded synthesis belongs to the class of *Safraless decision procedures* [22]. Safraless synthesis algorithms avoid the translation of the specification into an equivalent deterministic automaton via Safra's determinization procedure. Instead, the specification is first translated into an equivalent universal co-Büchi automaton, whose language is then approximated in a sequence of deterministic safety automata, obtained by bounding the number of visits to rejecting states [25]. Most synthesis tools for full LTL, including Unbeast [9], and Acacia+ [5], are based on this idea.

Bounded synthesis [25] limits not only the number of visits to rejecting states, but also the number of states of the synthesized system itself. As a result, the bounded synthesis problem can be represented as a decidable constraint system, even in settings where the classic synthesis problem is undecidable, such as the synthesis of asynchronous and distributed systems (cf. [14]). There have been several proposals for encodings of bounded synthesis. The first encoding [13, 25] was based on first-order logic modulo finite integer arithmetic. Improvements to the original encoding include the representation of transition systems that are not necessarily input-preserving, and, hence, often significantly smaller [14], the lazy generation of the constraints from model checking runs [11], and specification rewriting and modular solving [21]. Recently, a SAT-based encoding was proposed [27]. Another SAT-based encoding [12] bounds, in addition to the number of states, also the number of loops. A QBF-based encoding has been used in the related problem of solving Petri games [10]. Petri games can be used to solve certain distributed synthesis problems. They have, however, a significantly simpler winning condition than the games resulting from LTL specifications.

This paper presents the first encodings of bounded synthesis based on QBF and DQBF, and the first comprehensive evaluation of the spectrum of encodings from SAT to DQBF with state-of-the-art solvers. The encodings are significantly more concise than the previous SAT-based encodings and provide opportunities for solvers to exploit the symbolic representation of inputs and states. The empirical evidence shows that, with current solvers, the QBF encoding is superior to the SAT and DQBF encodings. A further contribution of the paper are the benchmarks themselves, which pinpoint opportunities for the improvement of the solvers, in particular for DQBF.

2 Preliminaries

Given a finite set of variables V, we identify boolean assignments $\alpha : V \to \mathbb{B}$ as elements from the powerset of V, i.e., given V and α, then $\boldsymbol{v} = \{v \mid \alpha(v) = \top\} \in$

2^V is a representation of α. We use $\mathbb{B}(V)$ to denote the set of propositional boolean formulas over the variables V.

LTL. Linear-time temporal logic (LTL) is the standard specification language for linear-time properties. Let Σ be a finite alphabet, i.e., a finite set of atomic propositions. The grammar of LTL is given by

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\psi \mid \varphi\,\mathcal{R}\,\psi \ ,$$

where $p \in \Sigma$ is an atomic proposition. The abbreviations $\mathit{true} \coloneqq p \vee \neg p$, $\mathit{false} \coloneqq \neg\mathit{true}$, $\diamondsuit\varphi = \mathit{true}\,\mathcal{U}\,\varphi$, and $\square\varphi = \mathit{false}\,\mathcal{R}\,\varphi$ are defined as usual. We assume standard semantics and write $\sigma \vDash \varphi$ if $\sigma \in (2^\Sigma)^\omega$ satisfies φ. The language of φ, written $\mathcal{L}(\varphi)$, is the set of ω-words that satisfy φ.

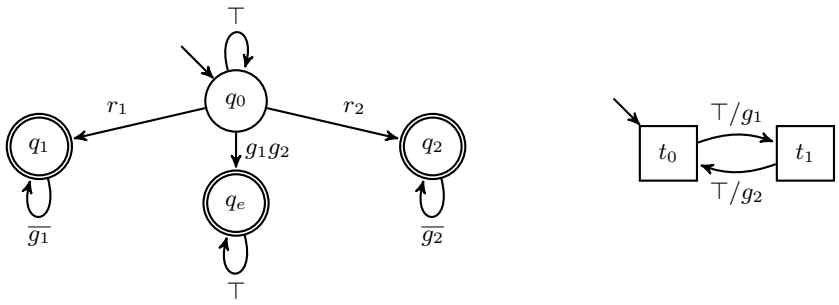
Automata. A universal co-Büchi automaton $\mathcal{A}$ over finite alphabet Σ is a tuple $\langle Q, q_0, \delta, F\rangle$, where Q is a finite set of states, $q_0 \in Q$ the designated initial state, $\delta : Q \times 2^\Sigma \times Q$ is the transition relation, and $F \subseteq Q$ is the set of rejecting states. Given an infinite word $\sigma \in (2^\Sigma)^\omega$, a run of σ on $\mathcal{A}$ is an infinite path $q_0q_1q_2\cdots \in Q^\omega$ where for all $i \geq 0$ it holds that $(q_i, \sigma_i, q_{i+1}) \in \delta$. A run is accepting, if it contains only finitely many rejecting states. $\mathcal{A}$ accepts a word σ, if *all* runs of σ on $\mathcal{A}$ are accepting. The language of $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set $\{\sigma \in (2^\Sigma)^\omega \mid \mathcal{A} \text{ accepts } \sigma\}$.

We represent automata as directed graphs with vertex set Q and a symbolic representation of the transition relation δ as propositional boolean formulas $\mathbb{B}(\Sigma)$. The rejecting states in F are marked by double lines.

Lemma 1. *Given an LTL formula φ, we can construct a universal co-Büchi automaton $\mathcal{A}_\varphi$ with $\mathcal{O}(2^{|\varphi|})$ states that accepts the language $\mathcal{L}(\varphi)$.*

Example 1. Consider the LTL formula $\psi = \square(r_1 \rightarrow \bigcirc\diamondsuit g_1) \wedge \square(r_2 \rightarrow \bigcirc\diamondsuit g_2) \wedge \square\neg(g_1 \wedge g_2)$. Whenever there is a request r_i, the corresponding grant g_i must be set eventually. Further, it is disallowed to set both grants simultaneously. The universal co-Büchi automaton $\mathcal{A}_\psi$ that accepts the same language as ψ is shown in Fig. 1(a).

Transition Systems. In the following, we partition the set of atomic propositions into a set I that contains propositions controllable by the environment and a set O that contains propositions controllable by the system. A transition system $\mathcal{T}$ is a tuple $\langle T, t_0, \tau\rangle$ where T is a finite set of states, $t_0 \in T$ is the designated initial state, and $\tau : T \times 2^I \rightarrow 2^O \times T$ is the transition function. The transition function τ maps a state t and a valuation of the inputs $\boldsymbol{i} \in 2^I$ to a valuation of the outputs, also called *labeling*, and a next state t'. If the labeling produced by $\tau(t, \boldsymbol{i})$ is independent of $\boldsymbol{i}$, we call $\mathcal{T}$ a state-labeled (or Moore) transition system and transition-labeled (or Mealy) otherwise. Formally, $\mathcal{T}$ is a state-labeled transition system if, given a state $t \in T$ and any $\boldsymbol{i} \neq \boldsymbol{i'} \in 2^I$ with $\tau(t, \boldsymbol{i}) = (\boldsymbol{o}, \_)$ and $\tau(t, \boldsymbol{i'}) = (\boldsymbol{o'}, \_)$ it holds that $\boldsymbol{o} = \boldsymbol{o'}$.



(a) Universal co-Büchi automaton $\mathcal{A}_\psi$ (b) Transition system $\mathcal{T}_{arb}$

Fig. 1: A specification automaton over inputs r_1, r_2 and outputs g_1, g_2 and a realizing transition system.

Given an infinite word $\boldsymbol{i_0 i_1} \cdots \in (2^I)^\omega$ over the inputs, $\mathcal{T}$ produces an infinite trace $(\{t_0\} \cup \boldsymbol{i}_0 \cup \boldsymbol{o_0})(\{t_1\} \cup \boldsymbol{i}_1 \cup \boldsymbol{o_1}) \cdots \in (2^{T \cup I \cup O})^\omega$ where $\tau(t_j, \boldsymbol{i_j}) = (\boldsymbol{o_j}, t_{j+1})$ for every $j \geq 0$. A path $w \in (2^{I \cup O})^\omega$ is the projection of a trace to the atomic propositions. We denote the set of all paths generated by a transition system $\mathcal{T}$ as $\mathit{Paths}(\mathcal{T})$. A transition system realizes an LTL formula if $\mathit{Paths}(\mathcal{T}) \subseteq \mathcal{L}(\varphi)$.

Example 2. Figure 1(b) depicts the two-state (state-labeled) transition system $\mathcal{T}_{arb} = \langle \{t_0, t_1\}, t_0, \tau \rangle$ with $\tau(t_0, \boldsymbol{i}) = (\{g_1\}, t_1)$ and $\tau(t_1, \boldsymbol{i}) = (\{g_2\}, t_0)$ for every $\boldsymbol{i} \in 2^I$. The set of paths is $\mathit{Paths}(\mathcal{T}) = (\{g_1\}\{g_2\})^\omega \cup (2^{\{i_1, i_2\}})^\omega$.

3 Bounded Synthesis

Bounded synthesis [14] is a synthesis procedure for LTL specifications that produces size-optimal transition systems. A given LTL formula φ is translated into a universal co-Büchi automaton $\mathcal{A}$ that accepts the language $\mathcal{L}(\varphi)$. A transition system $\mathcal{T}$ realizes specification φ if, and only if, every trace generated by $\mathcal{T}$ is in the language $\mathcal{L}(\varphi)$. $\mathcal{T}$ is accepted by $\mathcal{A}$ if every path of the unique run graph, that is the product of $\mathcal{T}$ and $\mathcal{A}$, has only finitely many visits to rejecting states. This acceptance is witnessed by a bounded annotation on this product.

The bounded synthesis approach is to synthesize a transition system of bounded size n, by solving a constraint system that asserts the existence of a transition system and labeling function of $\mathcal{T}$ as well as a valid annotation. In this section we discuss how to construct a formula that represents that a *given* annotation is correct. We will use this formula as a building block for different bounded synthesis constraint systems in Section 4.

The product of a transition system $\mathcal{T} = \langle T, t_0, \tau \rangle$ and a universal co-Büchi automaton $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$ is a *run graph* $\mathcal{G} = \langle V, E \rangle$, where $V = T \times Q$ is the set of vertices and $E \subseteq V \times V$ is the edge relation with

$$((t, q), (t', q')) \in E \text{ iff } \exists \boldsymbol{i} \in 2^I. \exists \boldsymbol{o} \in 2^O. \tau(t, \boldsymbol{i}) = (\boldsymbol{o}, t') \text{ and } (q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta \ .$$

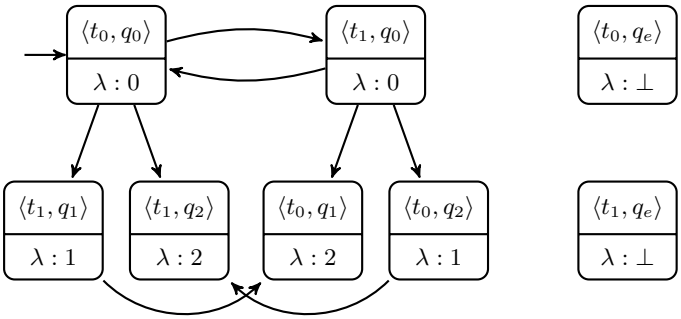


Fig. 2: Run graph of the automaton $\mathcal{A}_\psi$ and the two-state transition system $\mathcal{T}_{arb}$ from the earlier example (Fig. 1). The bottom node part displays a valid λ-annotation of the run graph.

An annotation $\lambda : T \times Q \to \{\bot\} \cup \mathbb{N}$ is a function that maps nodes from the run graph to either unreachable $\bot$ or a natural number k. An annotation is valid if it satisfies the following conditions:

- the pair of initial states (t_0, q_0) is labeled by a natural number ($\lambda(t_0, q_0) \neq \bot$), and
- if a pair of states (t, q) is annotated with a natural number ($\lambda(t, q) = k \neq \bot$) then for every $\boldsymbol{i} \in 2^I$ and $\boldsymbol{o} \in 2^O$ with $\tau(t, \boldsymbol{i}) = (\boldsymbol{o}, t')$ and $(q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$, the successor pair (t', q') is annotated with a greater number, which needs to be strictly greater if $q' \in F$ is rejecting. That is, $\lambda(t', q') \rhd_{q'} k$ where $\rhd_{q'} \coloneqq >$ if $q' \in F$ and $\geq$ otherwise.

Example 3. Figure 2 shows the run graph of $\mathcal{T}_{arb}$ and $\mathcal{A}_\psi$ from our earlier example (Fig. 1). Additionally, a valid annotation λ is provided at the second component of every node. One can verify that the annotation is correct by checking every edge individually. For example, the annotation has to increase from $\langle t_0, q_0\rangle \to \langle t_1, q_2\rangle$ and from $\langle t_0, q_2\rangle \to \langle t_1, q_2\rangle$ as q_2 is rejecting. As $\lambda(\langle t_0, q_0\rangle) = 0$ and $\lambda(\langle t_0, q_2\rangle) = 1$, it holds that $\lambda(\langle t_1, q_2\rangle)$ must be at least 2.

Given $\mathcal{T}$, $\mathcal{A}$, and λ, we want to derive a propositional constraint that is satisfiable if, and only if, the annotation is valid. First, by the characterization above, we know that we can verify the annotation by local checks, i.e., we have to consider only one step in the product graph. To derive a propositional encoding, we encode $\mathcal{T}$, $\mathcal{A}$, and λ:

- $\mathcal{T} = \langle T, t_0, \tau\rangle$. We represent the transition function τ by one variable $o_{t,\boldsymbol{i}}$ for every output proposition $o \in O$ and one variable $\tau_{t,\boldsymbol{i},t'}$ representing a transition form t to t'. Given $(t, t') \in T \times T$ and $\boldsymbol{i} \in 2^I$, it holds that (1) $\tau_{t,\boldsymbol{i},t'}$ is true if, and only if, $\tau(t, \boldsymbol{i}) = (\_, t')$, and (2) $o_{t,\boldsymbol{i}}$ is true if, and only if, $\tau(t, \boldsymbol{i}) = (\boldsymbol{o}, \_)$ and $o \in \boldsymbol{o}$.

– $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$. We represent $\delta : (Q \times 2^{I \cup O} \times Q)$ as propositional formulas $\delta_{t,q,\boldsymbol{i},q'}$ over the output variables $o_{t,\boldsymbol{i}}$. That is, an assignment $\boldsymbol{o}$ to the variables $o_{t,\boldsymbol{i}}$ satisfies $\delta_{t,q,\boldsymbol{i},q'}$ iff $(q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$.
– We first split the annotation λ into two parts: The first part $\lambda^{\mathbb{B}} : T \times Q \to \mathbb{B}$ represents the reachability constraint and the second part $\lambda^{\#} : T \times Q \to \mathbb{N}$ represents the bound. For every $t \in T$ and $q \in Q$ we introduce variables $\lambda^{\mathbb{B}}_{t,q}$ that we assign to be true iff the state pair is reachable from the initial state pair and a bit vector $\lambda^{\#}_{t,q}$ of length $\mathcal{O}(\log(|T| \cdot |Q|))$ that we assign the binary encoding of the value $\lambda(t, q)$.

Using the variables $o_{t,\boldsymbol{i}}$, $\tau_{t,\boldsymbol{i},t'}$, $\lambda^{\mathbb{B}}_{t,q}$, and $\lambda^{\#}_{t,q}$ (which have a unique assignment for a given $\mathcal{T}$, $\mathcal{A}$, and λ) as well as the propositional formulas $\delta_{t,q,\boldsymbol{i},q'}$, we construct a formula that represents that the annotation is valid:

$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \to \bigwedge_{q' \in Q} \bigwedge_{\boldsymbol{i} \in 2^I} \left(\delta_{t,q,\boldsymbol{i},q'} \to \bigwedge_{t' \in T} \left(\tau_{t,\boldsymbol{i},t'} \to \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

Theorem 1 ([14]). *Given $\mathcal{T}$, $\mathcal{A}$, and an annotation λ. If the propositional encoding of $\mathcal{T}$, $\mathcal{A}$, and λ satisfy the constraint system, then λ is a valid annotation.*

4 Encodings

Using the constraints developed in the last section for checking the validity of a given annotation, we now consider the problem of finding a transition system with a valid annotation.

This section introduces four encodings, starting with the most explicit encoding and moving first to an input-symbolic variant, then to a input- and state-symbolic variant and then further to a "fully symbolic" variant which treats inputs, transition systems states and the specification automaton symbolically. The first encoding can be solved using a SAT solver, the second requires a QBF solver, and the remaining two encodings require a DQBF solver. We will indicate for each encoding the difficulty to switch from the decision variant of the problem (realizability) to the constructive variant of the problem (synthesis).

4.1 SAT: The Basic Encoding

The *basic encoding* of bounded synthesis follows almost immediately from the last section. Instead of checking that for given $\mathcal{T}$, $\mathcal{A}$, and λ, the unique assignment to the variables satisfies the formula, we existentially quantify over the variables to find an assignment. We only have to add constraints that assert that the reachability information, represented in the variables $\lambda^{\mathbb{B}}_{t,q}$, is consistent, and that the transition relation, represented in the variables $\tau_{t,\boldsymbol{i},t'}$, provides at least one transition for every source state and every input. The consistency of

the reachability annotation is given once we assert $\lambda^{\mathbb{B}}_{t_0,q_0}$, as the formula itself asserts that the $\lambda^{\mathbb{B}}_{t,q}$ annotations are consistent with the transition relation.

$$\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$
$$\exists\{\tau_{t,\boldsymbol{i},t'} \mid (t,t') \in T \times T, \boldsymbol{i} \in 2^I\}$$
$$\exists\{o_{t,\boldsymbol{i}} \mid o \in O, t \in T, \boldsymbol{i} \in 2^I\}$$
$$\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigwedge_{\boldsymbol{i} \in 2^I} \bigvee_{t' \in T} \tau_{t,\boldsymbol{i},t'}$$
$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \bigwedge_{\boldsymbol{i} \in 2^I} \left(\delta_{t,q,\boldsymbol{i},q'} \rightarrow \bigwedge_{t' \in T} \left(\tau_{t,\boldsymbol{i},t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

Theorem 2. *The size of the constraint system is in* $\mathcal{O}(nm^2 \cdot 2^{|I|} \cdot (|\delta_{q,q'}| + n\log(nm)))$ *and the number of variables is in* $\mathcal{O}(n(m\log(nm) + 2^{|I|} \cdot (|O| + n)))$, *where* $n = |T|$ *and* $m = |Q|$.

Since we only quantify existentially over propositional variables, the encoding can be solved by a SAT solver. The synthesized transition system can be directly extracted from the satisfying assignment of the solver. For each state and each input, there is at least one true variable, indicating a possible successor. The variables $o_{t,\boldsymbol{i}}$ indicate whether output o is given at state t for input i.

4.2 QBF: The Input-Symbolic Encoding

One immediate drawback of the encoding above is the explicit handling of the inputs in the existential quantifiers representing the transition relation τ and the outputs o, which introduces several variables for each possible input $\boldsymbol{i} \in 2^I$. This leads to a constraint system that is exponential in the number of inputs, both in the size of the constraints and in the number of variables. Also, since all variables are quantified on the same level, some of the inherent structure of the problem is lost and the solver will have to assign a value to each propositional variable, which may lead to non-minimal solutions of τ and o due to unnecessary interdependencies.

By adding a universal quantification over the input variables, we obtain a quantified boolean formula (QBF) and avoid this exponential blow-up. In this encoding, the variables representing the λ-annotation remain in the outer existential quantifier - they cannot depend on the input. We then universally quantify over the valuations of the input propositions I (interpreted as variables in this encoding) before we existentially quantify over the remaining variables.

By the semantics of QBF, the innermost quantified variables, representing the transition function τ of $\mathcal{T}$, can be seen as boolean functions (Skolem functions) whose domain is the set of assignments to I. Indicating the dependency on the inputs in the quantifier hierarchy, we can now drop the indices $\boldsymbol{i}$ from the variables $\tau_{t,\boldsymbol{i},t'}$ and $o_{t,\boldsymbol{i}}$. Further, we now represent $\delta : (Q \times 2^{I \cup O} \times Q)$ as

propositional formulas $\delta_{t,q,q'}$ over the inputs I and output variables $o_{\boldsymbol{t}}$ (which depend on I) with the following property: an assignment $\boldsymbol{i} \cup \boldsymbol{o}$ satisfies $\delta_{t,q,q'}$ iff $(q, \boldsymbol{i} \cup \boldsymbol{o}, q') \in \delta$. We obtain the following formula for the input-symbolic encoding. (The gray box highlights the changes in the quantifier prefix compared to the previous encoding.)

$$\exists\{\lambda^{\mathbb{B}}_{t,q}, \lambda^{\#}_{t,q} \mid t \in T, q \in Q\}$$

$$\boxed{\forall I}$$

$$\exists\{\tau_{t,t'} \mid (t,t') \in T \times T\}$$

$$\exists\{o_t \mid o \in O, t \in T\}$$

$$\lambda^{\mathbb{B}}_{t_0,q_0} \wedge \bigwedge_{t \in T} \bigvee_{t' \in T} \tau_{t,t'}$$

$$\bigwedge_{q \in Q} \bigwedge_{t \in T} \left(\lambda^{\mathbb{B}}_{t,q} \rightarrow \bigwedge_{q' \in Q} \left(\delta_{t,q,q'} \rightarrow \bigwedge_{t' \in T} \left(\tau_{t,t'} \rightarrow \lambda^{\mathbb{B}}_{t',q'} \wedge \lambda^{\#}_{t',q'} \rhd_{q'} \lambda^{\#}_{t,q} \right) \right) \right)$$

Theorem 3. *Let $n = |T|$ and $m = |Q|$. The size of the input-symbolic constraint system is in $\mathcal{O}(nm^2(|\delta_{q,q'}| + n\log(nm)))$. The number of existential and universal variables is in $\mathcal{O}(n(m\log(nm) + |O| + n))$ and $\mathcal{O}(|I|)$, respectively.*

The input-symbolic encoding is not only exponentially smaller (in $|I|$) than the basic encoding, but also enables the solver to exploit the dependency between I and the transition function τ. An additional property of this encoding that we use in the implementation is the following: If we fix the values of the λ-annotation, the resulting 2QBF query represents all transition systems that are possible with respect to the λ-annotation. Since the outermost variables are existentially quantified, their assignments (in case the formula is satisfiable) can be extracted easily, even from non-certifying QBF solvers. For synthesis, we thus employ a two-step approach. We first solve the complete encoding and, if the formula was satisfiable, extract the assignment of the annotation variables $\lambda^{\mathbb{B}}_{t,q}$, and $\lambda^{\#}_{t,q}$. In the second step we instantiate the formula by the satisfiable λ-annotation and solve the remaining formula with a certifying solver to generate boolean functions for the inner existential variables. Those can be then be translated into a realizing transition system.

4.3 DQBF/EPR: The State- and Input-Symbolic Encoding

The previous encoding shows how to describe the functional dependency between the inputs I and the transition function τ and outputs o as a quantifier alternation. The reactive synthesis problem, however, contains more functional dependencies that we can exploit.

In the following we describe an encoding that also treats the states of the system to generate symbolically. First, we change the definition of T slightly. Where before, T was the set of states of the transition system, we now consider T as the set of *state bits* of the transition system. Consequently, the state space

of $\mathcal{T}$ is now 2^T and we consider the initial state to be the all-zero assignment to the variables T.

Since all variables depend on the state, we no longer have propositional variables. Instead, we quantify over the existence of boolean functions. Candidate logics for solving this query are dependency-quantified boolean formulas (DQBF) and the effective propositional fragment of first-order logic (EPR). While the existential quantification over functions is not immediately available in DQBF, we can encode them in a quadratic number of constraints, which is known as Ackermannization [7].

$$\exists\{\lambda_q^{\mathbb{B}} \colon 2^T \to \mathbb{B}, \lambda_q^{\#} \colon 2^T \to \mathbb{B}^b \mid q \in Q\}$$
$$\exists \tau : 2^T \times 2^I \to 2^T$$
$$\exists\{o \colon 2^T \times 2^I \to \mathbb{B} \mid o \in O\}$$
$$\forall I. \forall T, T'.$$
$$(T = 0 \to \lambda_{q_0}^{\mathbb{B}}(T))$$
$$\bigwedge_{q \in Q} \left(\lambda_q^{\mathbb{B}}(T) \to \bigwedge_{q' \in Q} \left(\delta_{q,q'} \wedge (\tau(T, I) \Rightarrow T') \to \lambda_{q'}^{\mathbb{B}}(T') \wedge \lambda_{q'}^{\#}(T') \rhd_{q'} \lambda_q^{\#}(T) \right) \right)$$

Theorem 4. *Let $n = |T|$ and $m = |Q|$. The size of the state-symbolic constraint system is in $\mathcal{O}(m^2(|\delta_{q,q'}| + \log(nm)))$. The number of existential and universal variables is in $\mathcal{O}(n + m \log(nm) + |O|)$ and $\mathcal{O}(n + |I|)$, respectively.*

Encoding the states of the specification automaton. The last dependency that we consider here is the dependency on the state space of the specification automaton. As a precondition, we need a symbolic representation $\mathscr{A} = \langle Q, q_{\text{init}}, \delta, q_{\text{reject}} \rangle$ of a universal co-Büchi automaton over alphabet $I \cup O$, where Q is a set of variables whose valuations represent the state space, $q_{\text{init}} \in \mathbb{B}(Q)$ is a propositional formula representing the initial state, $\delta \in \mathbb{B}(Q, I \cup O, Q')$ is the transition relation ($\boldsymbol{q} \cup \boldsymbol{i} \cup \boldsymbol{o} \cup \boldsymbol{q'}$ satisfies δ iff $\boldsymbol{q} \xrightarrow{\boldsymbol{i} \cup \boldsymbol{o}} \boldsymbol{q'}$), and $q_{\text{reject}} \in \mathbb{B}(Q)$ is a formula representing the rejecting states.

$$\exists \lambda^{\mathbb{B}} \colon 2^T \times 2^Q \to \mathbb{B}, \lambda^{\#} \colon 2^T \times 2^Q \to \mathbb{B}^b$$
$$\exists \tau : 2^T \times 2^I \to 2^T$$
$$\exists\{o \colon 2^T \times 2^I \to \mathbb{B} \mid o \in O\}$$
$$\forall I. \forall T, T'. \forall Q, Q'.$$
$$(t_{\text{init}} \wedge q_{\text{init}} \to \lambda^{\mathbb{B}}(T, Q)) \wedge$$
$$\left(\lambda^{\mathbb{B}}(T, Q) \to \left(\delta \wedge (\tau(T, I) \Rightarrow T') \to \lambda^{\mathbb{B}}(T', Q') \wedge \lambda^{\#}(T', Q') \rhd_{q'_{\text{reject}}} \lambda^{\#}(T, Q) \right) \right)$$

Theorem 5. *Let $n = |T|$ and $m = |Q|$. The size of the state-symbolic constraint system is in $\mathcal{O}(n + m + |\delta| + \log(nm))$. The number of existential and universal variables is in $\mathcal{O}(\log n + |O|)$ and $\mathcal{O}(n + m + |I|)$, respectively.*

Table 1: The table compares the encodings with respect to the number of variables and the size of the constraint system. We indicate the number of states of the transition system and the automaton by n and m, respectively.

	# existentials	# universals	constraint size
basic	$n(m\log(nm) + 2^{\lvert I\rvert} \cdot (\lvert O\rvert + n))$	-	$nm^2 \cdot 2^{\lvert I\rvert}$ $\cdot(\lvert\delta_{q,q'}\rvert + n\log(nm))$
input-symbolic	$n(m\log(nm) + \lvert O\rvert + n)$	$\lvert I\rvert$	$nm^2(\lvert\delta_{q,q'}\rvert + n\log(nm))$
state-symbolic	$n + m\log(nm) + \lvert O\rvert$	$n + \lvert I\rvert$	$m^2(\lvert\delta_{q,q'}\rvert + \log(nm))$
symbolic	$\log n + \lvert O\rvert$	$n + m + \lvert I\rvert$	$n + m + \lvert\delta\rvert + \log(nm)$

Table 2: Implementation matrix

	basic	input-symbolic	state-symbolic	symbolic
fragment	SAT	QBF	DQBF/EPR	DQBF/EPR
Mealy/Moore	●/●	●/●	●/●	●/●
solution extraction	●	●	○	○

4.4 Comparison

Table 1 compares the sizes of the encodings presented in this paper. From the basic propositional encoding, we developed more symbolic encodings by making dependencies explicit and employing Boolean functions. This conciseness, however, comes with the price of higher solving complexity. In the following section we study this tradeoff empirical.

5 Experimental Evaluation

5.1 Implementation

We implemented the encodings described in this paper in a tool called *BoSy*[3]. The LTL to automaton conversion is provided by the tool ltl3ba [1]. We reduce the number of counters and their size by only keeping them for automaton states within a rejecting strongly connected component, as proposed in [21]. The tool searches for a system implementation and a counter-strategy for the environment in parallel. An exponential search strategy is employed for the bound on the size of the transition system. In synthesis mode, we apply as a post-processing step circuit minimization provided by ABC [6].

For solving the non-symbolic encoding, we translate the propositional query to the DIMACS file format and solve it using the CryptoMiniSat SAT solver in version 5. The satisfying assignment is used to construct the realizing transition system.

The input-symbolic encoding is translated to the QDIMACS file format and is solved by a combination of the QBF preprocessor Bloqqer [2] and QBF solver

[3] The tool is available at https://react.uni-saarland.de/tools/bosy/

Table 3: Experimental results on selected scalable instances. Reported is the maximal parameter value k for which the instance could be solved and the cumulative solving time t (in seconds) up to this point.

	basic		input-sym		state-sym		Acacia		Party	
instance	max k	sum t	max k	sum t	max k	sum t	max k	sum t	max k	sum t
simple-arbiter	7	1008.7	8	2.7	3	100.5	8	59.2	6	902.7
full-arbiter	4	2994.5	3	0.6	2	13.3	5	2683.4	3	111.7
roundrob-arbiter	4	143.1	4	227.0	2	11.0	4	345.6	4	19.2
loadfull	5	268.7	8	44.2	2	25.1	4	83.7	4	213.5
prio-abiter	4	176.5	4	1.6	2	0.4	6	701.2	3	69.0
loadcomp	5	36.9	6	639.4	3	432.1	5	387.8	5	212.7
genbuf	2	1840.3	2	2711.8	0	–	5	159.3	0	–
generalized-buffer	2	2093.8	2	3542.8	0	–	6	3194.8	2	792.5
load-balancer	5	1148.8	8	83.2	2	75.3	5	270.8	0	–
detector	6	1769.0	8	1010.7	3	239.4	8	261.6	5	370.3

RAReQS [18]. The solution extraction is implemented in two steps. For satisfiable queries, we first derive a top level (λ) assignment [26] and instantiate the QBF query using this assignment which results in a 2QBF query that represents transition systems that satisfy the specification. This is then solved using a certifying QBF solver, such as QuAbS [28], CADET [23], or CAQE [24]. Among those, QuAbS performed best and was used in the evaluation. The resulting resulting Skolem functions, represented as AIGER circuit, are transformed into a representation of the transition system.

The symbolic encodings are translated to DQDIMACS file format and solved by the DQBF solver iDQ [16]. Due to limited solver support, we have not implemented solution extraction.

For comparison, we also implemented an SMT version using the classical encoding [14]. We also tested the state-symbolic and symbolic encoding with state-of-the art EPR solvers, but the solving times were not competitive. Table 2 gives an overview over the capabilities of the implemented encodings.

5.2 Setup & Benchmarks

For our experiments, we used a machine with a 3.6 GHz quad-core Intel Xeon processor and 32 GB of memory. The timeout and memout were set to 1 hour and 8 GB, respectively. We use the LTL benchmark sets from the latest reactive synthesis competition (SYNTCOMP 2016) [17]. The benchmarks include a variety of arbiter specifications of increasing complexity, load balancers, buffers, detectors as well as benchmark suites from previously existing tools. Some of

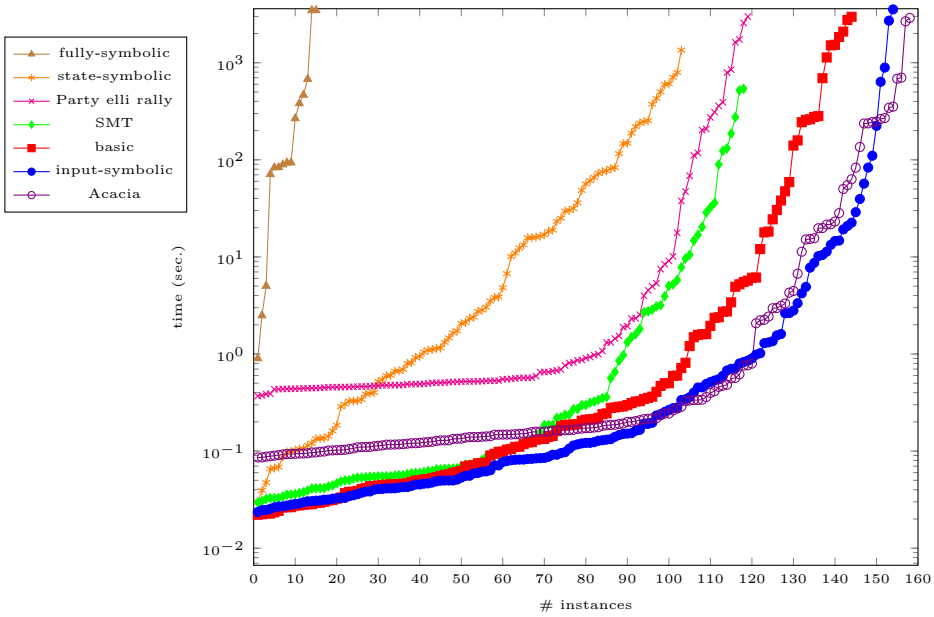


Fig. 3: Number of solved instances within 1 hour among the 201 instances from SYNTCOMP 2016. The time axis has logarithmic scale.

the benchmark classes are parameterized in the number of clients or masters, which allows scaling them for experimental purposes. In total, the realizability benchmark suite of SYNTCOMP 2016 consists of 195 benchmarks. We have additionally added six instances from scalable benchmark classes of this set to cover larger parameter values, resulting in a total size of 201 benchmarks for our benchmark set.

For comparison, we run the other two solves that participated in the SYNTCOMP 2016, that is Acacia [5], a game-based solver, and Party [20], a variant of the SMT bounded synthesis.

5.3 Realizability

In Table 3, we report results on realizability for all scalable instances from the aforementioned competition. We have omitted the results of our fully symbolic encoding from the table, since it could not solve a single instance of the selected benchmarks. The results from our own SMT encoding are also omitted, since they are very close to the results of the tool Party. Highlighted are those entries which reach the highest parameter value among the solvers and the best cumulative runtime within the class of instances.

An overall comparison of all realizability solvers on the full benchmark set is provided in Figure 3. For the individual solvers, we track the number of instances solved by this solver within a a certain time bound.

5.4 Synthesis

To evaluate the different encodings in terms of their solutions to the synthesis problem and to compare with other competing tools, we measure the size of the provided solutions. In line with the rules of SYNTCOMP, the synthesized transition system is encoded as an AIGER circuit. The size of the result is measured in terms of the number of AND gates. In the comparisons, we only consider instances where both solvers in the comparison had a result. All resulting circuits have been minimized using ABC.

First, we compare in the scatter plot of Figure 4 the propositional, non-symbolic encoding to the input-symbolic encoding. Since most points are below the diagonal and are therefore smaller than their counterparts, the input-symbolic solutions are better in size compared to the non-symbolic encoding.

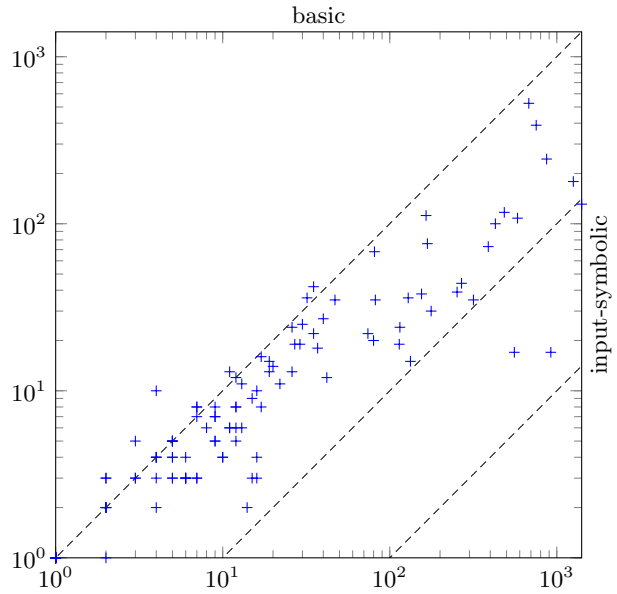


Fig. 4: Scatter plot comparing the size of the synthesized strategies between the basic (Sec. 4.1) and input-symbolic (Sec. 4.2) encoding. Both axes have logarithmic scale.

In Figure 5, we compare our input-symbolic encoding against two competing tools. On the left, we observe that the solution sizes of our input-symbolic encoding are significantly better (observe the log-log scale) than the solutions provided by Acacia. The reason for the size difference is that the strategies of Acacia may depend on the current state of the specification automaton, as they are extracted from the resulting safety game. When comparing to the SMT-

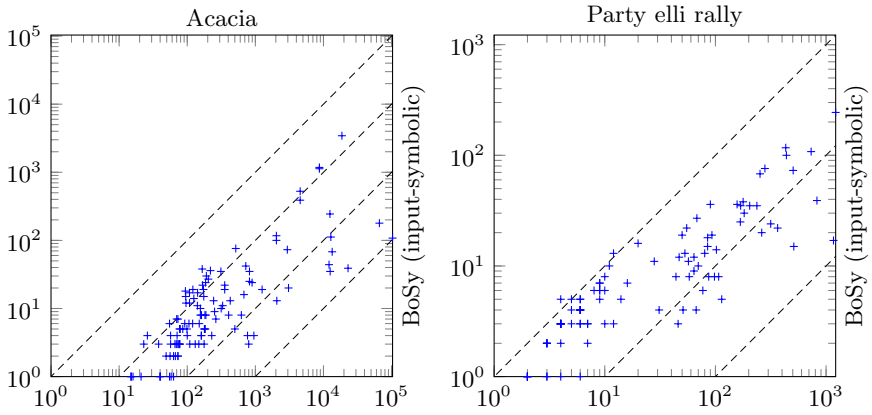


Fig. 5: Scatter plot comparing the size of the synthesized strategies of BoSy, Acacia, and Party elli rally. Both axes have logarithmic scale.

based Party tool, we again see a strict improvement in terms of strategy size, but not as significant as for Acacia.

We thus observe that the ability to universally quantify over the inputs and extract the transition system from the functional descriptions leads to advantages in terms of the size of the solution strategies.

6 Conclusion

We have revisited the bounded synthesis problem [14] and presented alternative encodings into boolean formulas (SAT), quantified boolean formulas (QBF), and dependency-quantified boolean formulas (DQBF). Our evaluation shows that the QBF approach clearly dominates the SAT approach and the DQBF approach, and also previous approaches to bounded synthesis – both in terms of the number of instances solved and in the size of the solutions. This demonstrates that, while modern QBF-solvers effectively exploit the input-symbolic representation, current DQBF solvers cannot yet take similar advantage of the state-symbolic representation. The benchmarks obtained from the encodings of bounded synthesis problems should therefore be useful in improving current solvers, in particular for DQBF.

References

1. Babiak, T., Kretínský, M., Rehák, V., Strejcek, J.: LTL to Büchi automata translation: Fast and more deterministic. In: Proceedings of TACAS. LNCS, vol. 7214, pp. 95–109. Springer (2012)
2. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Proceedings of CADE-23. LNCS, vol. 6803, pp. 101–115. Springer (2011)
3. Bloem, R., Galler, S.J., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Interactive presentation: Automatic hardware synthesis from specifications: a case study. In: Proceedings of DATE. pp. 1188–1193. EDA Consortium, San Jose, CA, USA (2007)
4. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: Proceedings of VMCAI. LNCS, vol. 8318, pp. 1–20. Springer (2014)
5. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: Proceedings of CAV. LNCS, vol. 7358, pp. 652–657. Springer (2012)
6. Brayton, R.K., Mishchenko, A.: ABC: an academic industrial-strength verification tool. In: Proceedings of CAV. LNCS, vol. 6174, pp. 24–40. Springer (2010)
7. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Santuari, A., Sebastiani, R.: To ackermann-ize or not to ackermann-ize? on efficiently handling uninterpreted function symbols in $SMT(EUF \grave{e} T)$. In: Proceedings of LPAR. LNCS, vol. 4246, pp. 557–571. Springer (2006)
8. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the American Mathematical Society 138 (1969), http://www.jstor.org/stable/1994916
9. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Proceedings of TACAS. LNCS, vol. 6605, pp. 272–275. Springer (2011)
10. Finkbeiner, B.: Bounded synthesis for Petri games. In: Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings. LNCS, vol. 9360, pp. 223–237. Springer (2015)
11. Finkbeiner, B., Jacobs, S.: Lazy synthesis. In: Proceedings of VMCAI. LNCS, vol. 7148, pp. 219–234. Springer (2012)
12. Finkbeiner, B., Klein, F.: Bounded cycle synthesis. In: Proceedings of CAV. LNCS, vol. 9779, pp. 118–135. Springer (2016)
13. Finkbeiner, B., Schewe, S.: SMT-based synthesis of distributed systems. In: Proceedings of AFM (2007)
14. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT 15(5-6), 519–539 (2013)
15. Finkbeiner, B., Tentrup, L.: Detecting unrealizable specifications of distributed systems. In: Proceedings of TACAS. LNCS, vol. 8413, pp. 78–92. Springer (2014)
16. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Proceedings of POS@SAT. EPiC Series in Computing, vol. 27, pp. 103–116. EasyChair (2014)
17. Jacobs, S., Bloem, R., Brenguier, R., Khalimov, A., Klein, F., Könighofer, R., Kreber, J., Legg, A., Narodytska, N., Pérez, G.A., Raskin, J., Ryzhyk, L., Sankur, O., Seidl, M., Tentrup, L., Walker, A.: The 3rd reactive synthesis competition (SYNTCOMP 2016): Benchmarks, participants & results. In: Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016. EPTCS, vol. 229, pp. 149–177 (2016)
18. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. Artif. Intell. 234, 1–25 (2016)

19. Jobstmann, B., Galler, S.J., Weiglhofer, M., Bloem, R.: Anzu: A tool for property synthesis. In: Proceedings of CAV. LNCS, vol. 4590, pp. 258–262. Springer (2007)
20. Khalimov, A., Jacobs, S., Bloem, R.: PARTY parameterized synthesis of token rings. In: Proceedings of CAV. LNCS, vol. 8044, pp. 928–933. Springer (2013)
21. Khalimov, A., Jacobs, S., Bloem, R.: Towards efficient parameterized synthesis. In: Proceedings of VMCAI. LNCS, vol. 7737, pp. 108–127. Springer (2013)
22. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Proceedings of FOCS. pp. 531–542. IEEE Computer Society (2005)
23. Rabe, M.N., Seshia, S.A.: Incremental determinization. In: Proceedings of SAT. LNCS, vol. 9710, pp. 375–392. Springer (2016)
24. Rabe, M.N., Tentrup, L.: CAQE: A certifying QBF solver. In: Proceedings of FM-CAD. pp. 136–143. IEEE (2015)
25. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Proceedings of ATVA. LNCS, vol. 4762, pp. 474–488. Springer (2007)
26. Seidl, M., Könighofer, R.: Partial witnesses from preprocessed quantified boolean formulas. In: Proceedings of DATE. pp. 1–6. European Design and Automation Association (2014)
27. Shimakawa, M., Hagihara, S., Yonezaki, N.: Reducing bounded realizability analysis to reachability checking. In: Proceedings of RP. LNCS, vol. 9328, pp. 140–152. Springer (2015)
28. Tentrup, L.: Solving QBF by abstraction. CoRR abs/1604.06752 (2016)