Bounded Synthesis*

Sven Schewe and Bernd Finkbeiner

Universität des Saarlandes, 66123 Saarbrücken, Germany

Abstract. The bounded synthesis problem is to construct an implementation that satisfies a given temporal specification and a given bound on the number of states. We present a solution to the bounded synthesis problem for linear-time temporal logic (LTL), based on a novel emptiness-preserving translation from LTL to safety tree automata. For distributed architectures, where standard unbounded synthesis is in general undecidable, we show that bounded synthesis can be reduced to a SAT problem. As a result, we obtain an effective algorithm for the bounded synthesis from LTL specifications in arbitrary architectures. By iteratively increasing the bound, our construction can also be used as a semi-decision procedure for the unbounded synthesis problem.

1 Introduction

Verification and synthesis both provide a formal guarantee that a system is implemented correctly. The difference between the two approaches is that while verification proves that a *given* implementation satisfies the specification, synthesis automatically *derives* one such implementation. Synthesis thus has the obvious advantage that it completely eliminates the need for manually writing and debugging code.

Unfortunately, the synthesis problem is undecidable even for simple distributed architectures. Consider, for example, the typical 2-process arbiter architecture shown in Figure 1b: the environment (env) sends requests (r_1, r_2) for access to a critical resource to two processes p_1 and p_2, which react by sending out grants (g_1, g_2). As shown by Pnueli and Rosner [1], the synthesis problem is undecidable for this architecture, because both p_1 and p_2 have access to information (r_1 and r_2, respectively) that is hidden from the other process. For system architectures without such *information forks* [2], like pipeline architectures (Figure 1a shows a pipeline of length 3), the synthesis problem is decidable, but has nonelementary complexity.

The high complexity of synthesis is explained by the fact that, as pointed out by Rosner [3], a small LTL formula of size n which refers to m different processes already suffices to specify a system that cannot be implemented with less than m-exp(n) states. From a practical point of view, however, it is questionable

* This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS).

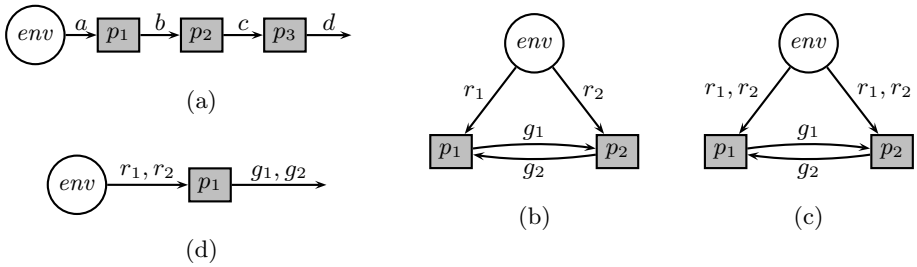

Fig. 1. Distributed architectures: (a) pipeline architecture, (b) 2-process arbiter architecture, (c) 2-process arbiter architecture with complete information, (d) single-process architecture.

whether such huge implementations should be considered by the synthesis algorithm, because they are likely to violate other design considerations (such as the available memory). In this paper, we therefore study a variation of the synthesis problem, which we call the *bounded synthesis problem*, where an upper limit on the size of the implementation is set in advance. The bound may either be an explicit design constraint or the result of iteratively increasing the limit in the search for a solution of minimal size.

Our starting point is the representation of the LTL specification as a universal co-Büchi tree automaton. We show that the acceptance of a finite-state transition system by a universal co-Büchi automaton can be characterized by the existence of an annotation that maps each pair of a state of the automaton and a state of the transition system to a natural number. The advantage of this characterization is that the acceptance condition can be simplified to a simple safety condition: we show that the universal co-Büchi automaton can be translated to an (emptiness-equivalent) deterministic *safety automaton* that implicitly builds a valid annotation. The emptiness of the safety automaton can then be determined in a simple two-player game, where player *accept* represents the system implementation and wins the game if the specification is satisfied; the opponent, player *reject*, wins the game if the specification is violated.

If the system architecture consists of a single process, as in Figure 1d, then a victory for player *accept* means that the specification is realizable. Any winning strategy for player *accept* immediately defines a correct implementation for the process. If the architecture consists of more than one process, as in the arbiter architecture of Figure 1b, then a victory for player *accept* only means that the specification can be implemented in the slightly modified architecture (shown for the arbiter example in Figure 1c), where all processes have the same information. An implementation for the architecture with incompletely informed processes must additionally satisfy a consistency requirement: if a process cannot distinguish between two different computation paths, it must react in the same way.

Inspired by the success of bounded model checking [4, 5], we show that the bounded synthesis problem for *distributed architectures* can be effectively

reduced to a SAT problem. We define a constraint system that describes the existence of a valid annotation and, additionally, ensures that the resulting implementation is consistent with the limited information available to the distributed processes. For this purpose, we introduce a mapping that decomposes the states of the safety game into the states of the individual processes: because the reaction of a process only depends on its local state, the process is forced to give the same reaction whenever it cannot distinguish between two paths in the safety game. The satisfiability of the constraint system can be checked using standard SAT solvers [6, 7]. As a result, we obtain an effective algorithm for the bounded synthesis from LTL specifications in arbitrary distributed architectures. By iteratively increasing the bound, our construction can also be used as a semi-decision procedure for the standard (unbounded) synthesis problem.

Related work. The *synthesis of distributed reactive systems* was pioneered by Pnueli and Rosner [1], who showed that the synthesis problem is undecidable in general and has nonelementary complexity for pipeline architectures. An automata-based synthesis algorithm for pipeline and ring architectures is due to Kupferman and Vardi [8]; Walukiewicz and Mohalik provided an alternative game-based construction [9]. We recently showed that the synthesis problem is decidable if and only if the architecture does not contain an information fork [2]. Madhusudan and Thiagarajan [10] consider the special case of *local* specifications (each property refers only to the variables of a single process). Among the class of acyclic architectures (without broadcast) this synthesis problem is decidable for exactly the doubly-flanked pipelines. Castellani, Mukund and Thiagarajan [11] consider *transition systems* as the specification language: an implementation is correct if the product of the processes is bisimilar to the specification. In this case, the synthesis problem is decidable independently of the architecture.

Our translation of LTL formulas to tree automata is based on Kupferman and Vardi's *Safraless decision procedures* [12]. We use their idea of avoiding Safra's determinization using universal co-Büchi automata. Our construction improves on [12] in that it produces deterministic safety automata instead of nondeterministic Büchi automata.

2 Preliminaries

We consider the synthesis of distributed reactive systems that are specified in linear-time temporal logic (LTL). Given an architecture A and an LTL formula φ, we decide whether there is an implementation for each system process in A, such that the composition of the implementations satisfies φ.

Architectures. An *architecture* A is a tuple (P, env, V, I, O), where P is a set of processes consisting of a designated environment process $env \in P$ and a set of system processes $P^- = P \setminus \{env\}$. V is a set of boolean system variables (which also serve as atomic propositions), $I = \{I_p \subseteq V \mid p \in P^-\}$ assigns a set

I_p of input variables to each system process $p \in P^-$, and $O = \{O_p \subseteq V \mid p \in P\}$ assigns a set O_p of output variables to each process $p \in P$ such that $\bigcup_{p \in P} O_p = V$. While the same variable $v \in V$ may occur in multiple sets in I to indicate broadcasting, the sets in O are assumed to be pairwise disjoint. If $O_{env} \subseteq I_p$ for every system process $p \in P^-$, we say the architecture is *fully informed*. Since every process in a fully informed architecture has enough information to simulate every other process, we can assume w.l.o.g. that a fully informed architecture contains only a single system process p, and that the input variables of p are the output variables of the environment process $I_p = O_{env}$.

Implementations. We represent implementations as labeled transition systems. For a given finite set Υ of directions and a finite set Σ of labels, a Σ-labeled Υ-*transition system* is a tuple $\mathcal{T} = (T, t_0, \tau, o)$, consisting of a set of states T, an initial state $t_0 \in T$, a transition function $\tau : T \times \Upsilon \to T$, and a labeling function $o : T \to \Sigma$. $\mathcal{T}$ is a *finite-state* transition system iff T is finite.

Each system process $p \in P^-$ is implemented as a 2^{O_p}-labeled 2^{I_p}-transition system $\mathcal{T}_p = (T_p, t_p, \tau_p, o_p)$. The specification φ refers to the composition of the system processes, which is the 2^V-labeled $2^{O_{env}}$-transition system $\mathcal{T}_A = (T, t_0, \tau, o)$, defined as follows: the set $T = \bigotimes_{p \in P^-} T_p \times 2^{O_{env}}$ of states is formed by the product of the states of the process transition systems and the possible values of the output variables of the environment. The initial state t_0 is formed by the initial states t_p of the process transition systems and a designated *root direction* $\subseteq O_{env}$. The transition function updates, for each system process p, the T_p part of the state in accordance with the transition function τ_p, using (the projection of) o as input, and updates the $2^{O_{env}}$ part of the state with the output of the environment process. The labeling function o labels each state with the union of its $2^{O_{env}}$ part with the labels of its T_p parts.

With respect to the system processes, the combined transition system thus simulates the behavior of all process transition systems; with respect to the environment process, it is *input-preserving*, i.e., in every state, the label accurately reflects the input received from the environment.

Synthesis. A specification φ is (finite-state) *realizable* in an architecture $A = (P, V, I, O)$ iff there exists a family of (finite-state) implementations $\{\mathcal{T}_p \mid p \in P^-\}$ of the system processes, such that their composition $\mathcal{T}_A$ satisfies φ.

Bounded Synthesis. We introduce bounds on the size of the process implementations and on the size of the composition. Given an architecture $A = (P, V, I, O)$, a specification φ is *bounded realizable* with respect to a family of bounds $\{b_p \in \mathbb{N} \mid p \in P^-\}$ on the size of the system processes and a bound $b_A \in \mathbb{N}$ on the size of the composition T_A, if there exists a family of implementations $\{\mathcal{T}_p \mid p \in P^-\}$, where, for each process $p \in P$, $\mathcal{T}_p$ has at most b_p states, such that the composition $\mathcal{T}_A$ satisfies φ and has at most b_A states.

Alternating Automata. An *alternating parity tree automaton* is a tuple $\mathcal{A} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$, where Σ denotes a finite set of labels, Υ denotes a finite

set of directions, Q denotes a finite set of states, $q_0 \in Q$ denotes a designated initial state, δ denotes a transition function, and $\alpha : Q \to C \subset \mathbb{N}$ is a coloring function. The transition function $\delta : Q \times \Sigma \to \mathbb{B}^+(Q \times \Upsilon)$ maps a state and an input letter to a positive boolean combination of states and directions. In our setting, the automaton runs on Σ-labeled Υ-transition systems. The acceptance mechanism is defined in terms of run graphs.

A *run graph* of an automaton $\mathcal{A} = (\Sigma, \Upsilon, Q, q_0, \delta, \alpha)$ on a Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is a minimal directed graph $\mathcal{G} = (G, E)$ that satisfies the following constraints:

- The vertices $G \subseteq Q \times T$ form a subset of the product of Q and T.
- The pair of initial states $(q_0, t_0) \in G$ is a vertex of $\mathcal{G}$.
- For each vertex $(q, t) \in G$, the set $\{(q', \upsilon) \in Q \times \Upsilon \mid ((q, t), (q', \tau(t, \upsilon))) \in E\}$ satisfies $\delta(q, o(t))$.

A run graph is *accepting* if every infinite path $g_0 g_1 g_2 \ldots \in G^\omega$ in the run graph satisfies the *parity condition*, which requires that the highest number occurring infinitely often in the sequence $\alpha_0 \alpha_1 \alpha_2 \in \mathbb{N}$ with $\alpha_i = \alpha(q_i)$ and $g_i = (q_i, t_i)$ is even. A transition system is accepted if it has an accepting run graph.

The set of transition systems accepted by an automaton $\mathcal{A}$ is called its *language* $\mathcal{L}(\mathcal{A})$. An automaton is empty iff its language is empty.

The acceptance of a transition system can also be viewed as the outcome of a game, where player *accept* chooses, for a pair $(q, t) \in Q \times T$, a set of atoms satisfying $\delta(q, o(t))$, and player *reject* chooses one of these atoms, which is executed. The transition system is accepted iff player *accept* has a strategy enforcing a path that fulfills the parity condition.

A *nondeterministic* automaton is a special alternating automaton, where the image of δ consists only of such formulas that, when rewritten in disjunctive normal form, contain exactly one element of $Q \times \{\upsilon\}$ for all $\upsilon \in \Upsilon$ in every disjunct. The emptiness of a nondeterministic automaton can be checked with a variation of the acceptance game called the *emptiness game*, where, in each step, player *accept* additionally chooses the label from Σ. A nondeterministic automaton is empty iff the emptiness game is won by player *reject*.

An alternating automaton is called *universal* if, for all states q and input letters σ, $\delta(q, \sigma)$ is a conjunction. A universal and nondeterministic automaton is called *deterministic*.

A parity automaton is called a *Büchi* automaton if the image of α is contained in $\{1, 2\}$, a *co-Büchi* automaton iff the image of α is contained in $\{0, 1\}$, and a *safety* automaton if the image of α is $\{0\}$. Büchi and co-Büchi automata are denoted by $(\Sigma, \Upsilon, Q, q_0, \delta, F)$, where $F \subseteq Q$ denotes the states with the higher color. Safety automata are denoted by $(\Sigma, \Upsilon, Q, q_0, \delta)$. A run graph of a Büchi automaton is thus accepting if, on every infinite path, there are infinitely many visits to F; a run graph of a co-Büchi automaton is accepting if, on every path, there are only finitely many visits to F. For safety automata, every run graph is accepting.

3 Annotated Transition Systems

In this section, we introduce an annotation function for transition systems. The annotation function has the useful property that a finite-state transition system satisfies the specification if and only if it has a valid annotation.

Our starting point is a representation of the specification as a universal co-Büchi automaton. Since the automaton is universal, every transition system in the language of the automaton has a unique run graph. The annotation assigns to each pair (q, t) of a state q of the automaton and a state t of the transition system either a natural number or a blank sign. The natural number indicates the maximal number of rejecting states that occur on some path to (q, t) in the run graph.

We show that the finite-state transition systems accepted by the automaton are exactly those transition systems for which there is an annotation that assigns only natural numbers to the vertices of the run graph. We call such annotations *valid*.

3.1 Universal Co-Büchi Automata

We translate a given LTL specification φ into an equivalent universal co-Büchi automaton $\mathcal{U}_\varphi$. This can be done with a single exponential blow-up by first negating φ, then translating $\neg\varphi$ into an equivalent nondeterministic Büchi word automaton, and then constructing a universal co-Büchi automaton that simulates the Büchi automaton along each path: if each path is co-Büchi accepting (i.e., it violates the Büchi condition), then the specification φ must hold along every path.

Theorem 1. *[12] Given an LTL formula φ, we can construct a universal co-Büchi automaton $\mathcal{U}_\varphi$ with $2^{O(|\varphi|)}$ states that accepts a transition system $\mathcal{T}$ iff $\mathcal{T}$ satisfies φ.* □

3.2 Bounded Annotations

An *annotation* of a transition system $\mathcal{T} = (T, t_0, \tau, o)$ on a universal co-Büchi automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, \delta, F)$ is a function $\lambda : Q \times T \to \{\_\} \cup \mathbb{N}$. We call an annotation *c-bounded* if its mapping is contained in $\{\_\} \cup \{0, \ldots, c\}$, and *bounded* if it is c-bounded for some $c \in \mathbb{N}$. An annotation is *valid* if it satisfies the following conditions:

- the pair (q_0, t_0) of initial states is annotated with a natural number ($\lambda(q_0, t_0) \neq \_$), and
- if a pair (q, t) is annotated with a natural number ($\lambda(q, t) = n \neq \_$) and $(q', \upsilon) \in \delta(q, o(t))$ is an atom of the conjunction $\delta(q, o(t))$, then $(q', \tau(t, \upsilon))$ is annotated with a greater number, which needs to be strictly greater if $q' \in F$ is rejecting. That is, $\lambda(q', \tau(t, \upsilon)) \rhd_{q'} n$ where $\rhd_{q'}$ is $>$ for $q' \in F$ and $\geq$ otherwise.

Theorem 2. *A finite-state Σ-labeled Υ-transition system $\mathcal{T} = (T, t_0, \tau, o)$ is accepted by a universal co-Büchi automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, \delta, F)$ iff it has a valid $(|T| \cdot |F|)$-bounded annotation.*

Proof. Since $\mathcal{U}$ is universal, $\mathcal{U}$ has a unique run graph $\mathcal{G} = (G, E)$ on $\mathcal{T}$. Since $\mathcal{T}$ and $\mathcal{U}$ are finite, $\mathcal{G}$ is finite, too.

If $\mathcal{G}$ contains a lasso with a rejecting state in its loop, i.e., a path $(q_0, t_0)(q_1, t_1) \dots (q_n, t_n) = (q'_0, t'_0)$ and a path $(q'_0, t'_0)(q'_1, t'_1) \dots (q'_m, t'_m) = (q'_0, t'_0)$ such that q'_i is rejecting for some $i \in \{1, \dots, m\}$, then, by induction, any valid annotation λ satisfies $\lambda(q_j, t_j) \in \mathbb{N}$ for all $j \in \{0, \dots, n\}$, $\lambda(q'_j, t'_j) \in \mathbb{N}$ for all $j \in \{0, \dots, m\}$, $\lambda(q'_{j-1}, t'_{j-1}) \leq \lambda(q'_j, t'_j)$ for all $j \in \{1, \dots, m\}$, and $\lambda(q'_{i-1}, t'_{i-1}) < \lambda(q'_i, t'_i)$. $\lightning$

If, on the other hand, $\mathcal{G}$ does not contain a lasso with a rejecting state in its loop, we can easily infer a valid $(|T| \cdot |F|)$-bounded annotation by assigning to each vertex $(q, t) \in G$ of the run graph the highest number of rejecting states occurring on some path $(q_0, t_0)(q_1, t_1) \dots (q, t)$, and by assigning $\_$ to every pair of states $(q, t) \notin G$ not in $\mathcal{G}$. □

3.3 Estimating the Bound

Since the distributed synthesis problem is undecidable, it is in general not possible to estimate a sufficient bound c that guarantees that a transition system with a valid c-bounded annotation exists if the specification is realizable.

For fully informed architectures, however, such an estimate is possible. If a universal co-Büchi automaton is non-empty, then the size of a smallest accepted transition system can be estimated by the size of an equivalent deterministic parity automaton.

Theorem 3. *[13] Given a universal co-Büchi automaton $\mathcal{U}$ with n states, we can construct an equivalent deterministic parity automaton $\mathcal{P}$ with n^{2n+2} states and $2n$ colors.* □

A solution to the synthesis problem is required to be input-preserving, i.e., in every state, the label must accurately reflect the input. Input preservation can be checked with a deterministic safety automaton $\mathcal{D}_\mathcal{I}$, whose states are formed by the possible inputs $\mathcal{I} = 2^{O_{env}}$. In every state $i \in \mathcal{I}$, $\mathcal{D}_\mathcal{I}$ checks if the label agrees with the input i, and sends the successor state $i' \in \mathcal{I}$ into the direction i'. If $\mathcal{U}$ accepts an input-preserving transition system, then we can construct a *finite* input-preserving transition system, which is accepted by $\mathcal{U}$, by evaluating the emptiness game of the product automaton of $\mathcal{P}$ and $\mathcal{D}_\mathcal{I}$. The minimal size of such an input-preserving transition system can be estimated by the size of $\mathcal{P}$ and $\mathcal{I}$.

Corollary 1. *If a universal co-Büchi automaton $\mathcal{U}$ with n states and m rejecting states accepts an input-preserving transition system, then $\mathcal{U}$ accepts a finite input-preserving transition system $\mathcal{T}$ with $n^{2n+2} \cdot |\mathcal{I}|$ states, where $\mathcal{I} = 2^{O_{env}}$. $\mathcal{T}$ has a valid $m \cdot n^{2n+2} \cdot |\mathcal{I}|$-bounded annotation for $\mathcal{U}$.* □

4 Automata-Theoretic Bounded Synthesis

Using the annotation function, we can reduce the synthesis problem for fully informed architectures to a simple emptiness check on safety automata. The following theorem shows that there is a deterministic safety automaton that, for a given parameter value c, accepts a transition system iff it has a valid c-bounded annotation. This leads to the following automata-theoretic synthesis procedure for fully informed architectures:

Given a specification, represented as a universal co-Büchi automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, q_0, \delta, F)$, we construct a sequence of safety automata that check for valid bounded annotations up to the bound $c = |F| \cdot b$, where b is either the predefined bound b_A on the size of the transition system, or the sufficient bound $n^{2n+2} \cdot |\mathcal{I}|$ from Corollary 1. If the intersection of $\mathcal{D}_\mathcal{I}$ with one of these automata is non-empty, then the specification is realizable; if the intersection with the safety automaton for the largest parameter value c is empty, then the specification is unrealizable. The emptiness of the automata can be checked by solving their emptiness games.

Theorem 4. *Given a universal co-Büchi automaton $\mathcal{U} = (\Sigma, \Upsilon, Q, q_0, \delta, F)$, we can construct a family of deterministic safety automata $\{\mathcal{D}_c = (\Sigma, \Upsilon, S_c, s_0, \delta_c) \mid c \in \mathbb{N}\}$ such that $\mathcal{D}_c$ accepts a transition system iff it has a valid c-bounded annotation.*

Construction: We choose the functions from Q to the union of $\mathbb{N}$ and a blank sign ($S = Q \to \{\_\} \cup \mathbb{N}$) as the state space of an abstract deterministic safety automaton $\mathcal{D} = (\Sigma, \Upsilon, S, s_0, \delta_\infty)$. Each state of $\mathcal{D}$ indicates how many times a rejecting state may have been visited in some trace of the run graph that passes the current position in the transition system. The initial state of $\mathcal{D}$ maps the initial state of $\mathcal{U}$ to 0 ($s_0(q_0) = 0$) and all other states of $\mathcal{U}$ to blank ($\forall q \in Q \setminus \{q_0\}.\, s_0(q) = \_$).

Let $\delta^+_\infty(s, \sigma) = \{((q', s(q') + f(q')), \upsilon) \mid q, q' \in Q,\, s(q) \neq \_, \text{ and } (q', \upsilon) \in \delta(q, \sigma)\}$, where $f(q) = 1\, \forall q \in F$, and $f(q) = 0\, \forall q \notin F$, be the function that collects the transitions of $\mathcal{U}$. The transition function δ_∞ is defined as follows: $\delta_\infty(s, \sigma) = \bigwedge_{\upsilon \in \Upsilon}(s_\upsilon, \upsilon)$ with $s_\upsilon(q) = max\{n \in \mathbb{N} \mid ((q, n), \upsilon \in \delta^+_\infty(s, \sigma)\}$ (where $max\{\emptyset\} = \_$). $\mathcal{D}_c$ is formed by restricting the states of $\mathcal{D}$ to $S_c = Q \to \{\_\} \cup \{0, \ldots, c\}$.

Proof. Let λ be a valid c-bounded annotation of $\mathcal{T} = (T, t_0, \tau, o)$ for $\mathcal{U}$, and let λ_t denote the function with $\lambda_t(q) = \lambda(q, t)$. For two functions $s, s' : Q \to \{\_\} \cup \mathbb{N}$, we write $s \leq s'$ if $s(q) \leq s'(q)$ holds for all $q \in Q$, where $\_$ is the minimal element ($\_ < n$ for all $n \in \mathbb{N}$). We show by induction that $\mathcal{D}_c$ has a run graph $\mathcal{G} = (G, E)$ for $\mathcal{T}$, such that $s \leq \lambda_t$ holds true for all vertices $(s, t) \in G$ of the run graph. For the induction basis, $s_0 \leq \lambda_{t_0}$ holds by definition. For the induction step, let $(s, t) \in G$ be a vertex of $\mathcal{G}$. By induction hypothesis, we have $s \leq \lambda_t$. With the definition of δ^+_∞ and the validity of λ, we can conclude that $((q', n), \upsilon) \in \delta^+_\infty(s, o(t))$ implies $n \leq \lambda_{\tau(t,\upsilon)}(q')$, which immediately implies $s' \leq \lambda_{t'}$ for all successors (s', t') of (s, t) in $\mathcal{G}$.

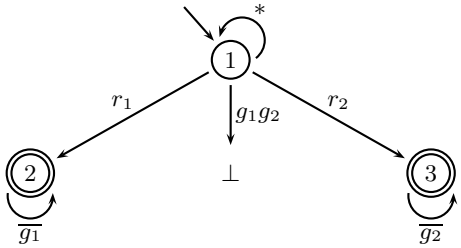

Fig. 2. Specification of a simple arbiter, represented as a universal co-Büchi automaton. The states depicted as double circles (2 and 3) are the rejecting states in F.

Let now $\mathcal{G} = (G, E)$ be an accepting run graph of $\mathcal{D}_c$ for $\mathcal{T}$, and let $\lambda(q,t) = max\{s(q) \mid (s,t) \in G\}$. Then λ is obviously a c-bounded annotation. For the validity of λ, $\lambda(q_0, t_0) \in \mathbb{N}$ holds true since $s_0(q_0) \in \mathbb{N}$ is a natural number and $(s_0, t_0) \in G$ is a vertex of $\mathcal{G}$. Also, if a pair (q,t) is annotated with a natural number $\lambda(q,t) = n \neq \_$, then there is a vertex $(s,t) \in G$ with $s(q) = n$. If now $(q', \upsilon) \in \delta(q, o(t))$ is an atom of the conjunction $\delta(q, o(t))$, then $((q', n + f(q')), \upsilon) \in \delta^+_\infty(s, o(t))$ holds true, and the υ-successor $(s', \tau(t, \upsilon))$ of (s,t) satisfies $s'(q') \triangleright_{q'} n$. The validity of λ now follows with $\lambda(q', \tau(t, \upsilon) \geq s'(q')$. □

Remark 1. Since $\mathcal{U}$ may accept transition systems where the number of rejecting states occurring on a path is unbounded, the union of the languages of all $\mathcal{D}_c$ is, in general, a strict subset of the language of $\mathcal{U}$. Every finite-state transition system in the language of $\mathcal{U}$, however, is accepted by almost all $\mathcal{D}_c$.

Example. Consider the specification of a simple arbiter, depicted as a universal co-Büchi automaton in Figure 2. The specification requires that globally (1) at most one process has a grant and (2) each request is eventually followed by a grant. The emptiness game for $\mathcal{D}_1$ intersected with $\mathcal{D}_\mathcal{I}$ is depicted in Figure 3.

5 Constraint-Based Bounded Synthesis

We now develop an alternative synthesis method for fully informed architectures that uses a SAT solver to determine an input-preserving transition system with a valid annotation. The constraint system defined in this section will provide the foundation for the synthesis method for general distributed architectures in Section 6.

We represent the (unknown) transition system and its annotation by uninterpreted functions. The existence of a valid annotation is thus reduced to the satisfiability of a constraint system in first-order logic modulo finite integer arithmetic. The advantage of this representation is that the size of the constraint system is small (bilinear in the size of $\mathcal{U}$ and the number of directions). Furthermore, the additional constraints needed for distributed synthesis, which will be

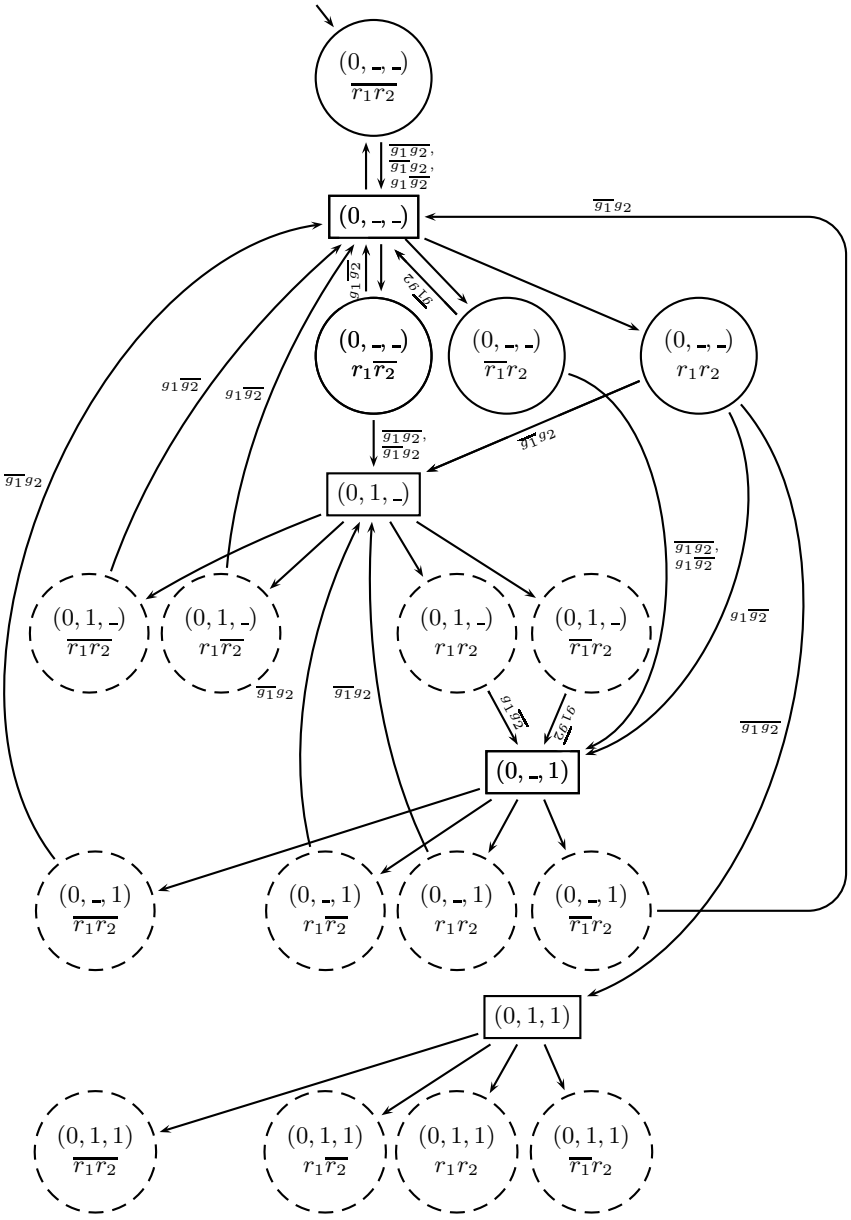

Fig. 3. Example of a safety game for synthesis in a fully informed architecture. The figure shows the emptiness game for the intersection of $\mathcal{D}_1$ and $\mathcal{D}_{\mathcal{I}}$ in the arbiter example (Figure 2). Circles denote game positions for player *accept*, rectangles denote game positions for player *reject*. Game positions that are not completely expanded (i.e., that have more successors if the parameter is increased) are dashed. The starting position specifies $\overline{r_1 r_2}$ as the (arbitrarily chosen) root direction. Player *accept* wins the game by avoiding the move to $(0, 1, 1)$.

defined in Section 6, have a compact representation as well (logarithmic in the number of directions of the individual processes).

The constraint system specifies the existence of a finite input-preserving 2^V-labeled $2^{O_{env}}$-transition system $\mathcal{T} = (T, t_0, \tau, o)$ that is accepted by the universal co-Büchi automaton $\mathcal{U}_\varphi = (\Sigma, \Upsilon, Q, q_0, \delta, F)$ and has a valid annotation λ.

To encode the transition function τ, we introduce a unary function symbol τ_υ for every output $\upsilon \subseteq O_{env}$ of the environment. Intuitively, τ_υ maps a state t of the transition system $\mathcal{T}$ to its υ-successor $\tau_\upsilon(t) = \tau(t, \upsilon)$.

To encode the labeling function o, we introduce a unary predicate symbol a for every variable $a \in V$. Intuitively, a maps a state t of the transition system $\mathcal{T}$ to *true* iff it is part of the label $o(t) \ni a$ of $\mathcal{T}$ in t.

To encode the annotation, we introduce, for each state q of the universal co-Büchi automaton $\mathcal{U}$, a unary predicate symbol $\lambda_q^{\mathbb{B}}$ and a unary function symbol $\lambda_q^{\#}$. Intuitively, $\lambda_q^{\mathbb{B}}$ maps a state t of the transition system $\mathcal{T}$ to *true* iff $\lambda(q, t)$ is a natural number, and $\lambda_q^{\#}$ maps a state t of the transition system $\mathcal{T}$ to $\lambda(q, t)$ if $\lambda(q, t)$ is a natural number and is unconstrained if $\lambda(q, t) = \_$.

We can now formalize that the annotation of the transition system is valid by the following first order constraints (modulo finite integer arithmetic):
$\forall t.\, \lambda_q^{\mathbb{B}}(t) \wedge \underline{(q', \upsilon) \in \delta(q, \overrightarrow{a}(t))} \rightarrow \lambda_{q'}^{\mathbb{B}}(\tau_\upsilon(t)) \wedge \lambda_{q'}^{\#}(\tau_\upsilon(t)) \triangleright_q \lambda_q^{\#}(t)$, where $\overrightarrow{a}(t)$ represents the label $o(t)$, $\underline{(q', \upsilon) \in \delta(q, \overrightarrow{a}(t))}$ represents the corresponding propositional formula, and $\triangleright_q$ stands for $\triangleright_q \equiv >$ if $q \in F$ and $\triangleright_q \equiv \geq$ otherwise. Additionally, we require $\lambda_{q_0}^{\mathbb{B}}(0)$, i.e., we require the pair of initial states to be labeled by a natural number (w.l.o.g. $t_0 = 0$).

To guarantee that the resulting transition system is input-preserving, we add, for each $a \in O_{env}$ and each $\upsilon \subseteq O_{env}$, a constraint $\forall t.\, a(\tau_\upsilon(t))$ if $a \in \upsilon$, and a constraint $\forall t.\, \neg a(\tau_\upsilon(t))$ if $a \notin \upsilon$. Additionally, we require that the initial state is labeled with the root direction.

As an obvious implication of Theorem 2, this constraint system is satisfiable if and only if $\mathcal{U}$ accepts a finite input-preserving transition system.

Theorem 5. *For fully informed architectures, the constraint system inferred from the specification, represented as the universal co-Büchi automaton $\mathcal{U}$, is satisfiable modulo finite integer arithmetic iff the specification is finite-state realizable.* □

Lemma 1. *For a specification represented as a universal co-Büchi automaton $\mathcal{U} = (2^V, 2^{O_{env}}, Q, q_0, \delta, F)$, the inferred constraint system has size $O(|\delta| \cdot |V| + |O_{env}| \cdot |2^{O_{env}}|)$.* □

The main parameter of the constraint system is the bound b_A on the size of the transition system $\mathcal{T}_A$. If we use b_A to unravel the constraint system completely (i.e., if we resolve the universal quantification explicitly), the size of the resulting constraint system is linear in b_A.

Theorem 6. *For a specification, represented as a universal co-Büchi automaton $\mathcal{U} = (2^V, 2^{O_{env}}, Q, q_0, \delta, F)$, and a given bound b_A on the size of the transition system $\mathcal{T}_A$, the unraveled constraint system has size $O(b_A \cdot (|\delta| \cdot |V| + |O_{env}| \cdot |2^{O_{env}}|))$.*

1. $\forall t.\, r_1(\tau_{r_1 r_2}(t)) \wedge r_2(\tau_{r_1 r_2}(t)) \wedge r_1(\tau_{r_1 \overline{r}_2}(t)) \wedge \neg r_2(\tau_{r_1 \overline{r}_2}(t))$
   $\wedge\, \neg r_1(\tau_{\overline{r}_1 r_2}(t)) \wedge r_2(\tau_{\overline{r}_1 r_2}(t)) \wedge \neg r_1(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \neg r_2(\tau_{\overline{r}_1 \overline{r}_2}(t))$
2. $\lambda_1^{\mathbb{B}}(0) \wedge \neg r_1(0) \wedge \neg r_2(0)$
3. $\forall t.\, \lambda_1^{\mathbb{B}}(t) \rightarrow \lambda_1^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_1^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \geq \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_1^{\#}(\tau_{\overline{r}_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_1^{\#}(\tau_{r_1 \overline{r}_2}(t)) \geq \lambda_1^{\#}(t)$
   $\wedge\, \lambda_1^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_1^{\#}(\tau_{r_1 r_2}(t)) \geq \lambda_1^{\#}(t)$
4. $\forall t.\, \lambda_1^{\mathbb{B}}(t) \rightarrow \neg g_1(t) \vee \neg g_2(t)$
5. $\forall t.\, \lambda_1^{\mathbb{B}}(t) \wedge r_1(t) \rightarrow \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 \overline{r}_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 r_2}(t)) > \lambda_1^{\#}(t)$
6. $\forall t.\, \lambda_1^{\mathbb{B}}(t) \wedge r_2(t) \rightarrow \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 \overline{r}_2}(t)) > \lambda_1^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 r_2}(t)) > \lambda_1^{\#}(t)$
7. $\forall t.\, \lambda_2^{\mathbb{B}}(t) \wedge \neg g_1(t) \rightarrow \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 \overline{r}_2}(t)) > \lambda_2^{\#}(t)$
   $\wedge\, \lambda_2^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 r_2}(t)) > \lambda_2^{\#}(t)$
8. $\forall t.\, \lambda_3^{\mathbb{B}}(t) \wedge \neg g_2(t) \rightarrow \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda_3^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_3^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 \overline{r}_2}(t)) > \lambda_3^{\#}(t)$
   $\wedge\, \lambda_3^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 r_2}(t)) > \lambda_3^{\#}(t)$

Fig. 4. Example of a constraint system for synthesis in a fully informed architecture. The figure shows the constraint system for the arbiter example (Figure 2). The arbiter is to be implemented in the fully informed architecture shown in Figure 1d.

It is satisfiable if and only if the specification is bounded realizable in the fully informed architecture $(\{env, p\}, V, \{I_p = O_{env}\}, \{O_{env}, O_p = V \setminus O_{env}\})$ *with bound* b_A. □

Example. Figure 4 shows the constraint system, resulting from the specification of an arbiter by the universal co-Büchi automaton depicted in Figure 2, implemented on the single process architecture of Figure 1d (or, likewise, on the distributed but fully informed architecture of Figure 1c).

The first constraint represents the requirement that the resulting transition system must be input-preserving, the second requirement represents the initialization (where $\neg r_1(0) \wedge \neg r_2(0)$ represents an arbitrarily chosen root direction), and the requirements 3 through 8 each encode one transition of the universal automaton of Figure 2. Following the notation of Figure 2, r_1 and r_2 represent the requests and g_1 and g_2 represent the grants.

6 Distributed Synthesis

To solve the distributed synthesis problem for a given architecture $A = (P, V, I, O)$, we need to find a family of (finite-state) transition systems $\{\mathcal{T}_p = (T_p, t_0^p, \tau_p, o_p) \mid p \in P^-\}$ such that their composition to $\mathcal{T}_A$ satisfies the specification. The constraint system developed in the previous section can be adapted to distributed synthesis by explicitly decomposing the global state space of the combined transition system $\mathcal{T}_A$: we introduce a unary function symbol d_p for each process $p \in P^-$, which, intuitively, maps a state $t \in T_A$ of the product state space to its p-component $t_p \in T_p$.

The value of an output variable $a \in O_p$ may only depend on the state of the process transition system $\mathcal{T}_p$. We therefore replace every occurrence of $a(t)$ in the constraint system of the previous section by $a(d_p(t))$. Additionally, we require that every process p acts consistently on any two histories that it cannot distinguish. The update of the state of $\mathcal{T}_p$ may thus only depend on the state of $\mathcal{T}_p$ and the input visible to p. This is formalized by the following constraints:

1. $\forall t.\, d_p(\tau_\upsilon(t)) = d_p(\tau_{\upsilon'}(t))$ for all decisions $\upsilon, \upsilon' \subseteq O_{env}$ of the environment that are indistinguishable for p (i.e., $\upsilon \cap I_p = \upsilon' \cap I_p$).
2. $\forall t, u.\, d_p(t) = d_p(u) \wedge \bigwedge_{a \in I_p \smallsetminus O_{env}} \big(a(d_{p_a}(t)) \leftrightarrow a(d_{p_a}(t))\big) \rightarrow d_p(\tau_\upsilon(t)) = d_p(\tau_\upsilon(u))$ for all decisions $\upsilon \subseteq O_{env} \cap I_p$ (picking one representative for each class of environment decisions that p can distinguish). $p_a \in P^-$ denotes the process controlling the output variable $a \in O_{p_a}$.

Since the combined transition system $\mathcal{T}_A$ is finite-state, the satisfiability of this constraint system modulo finite integer arithmetic is equivalent to the distributed synthesis problem.

Theorem 7. *The constraint system inferred from the specification, represented as the universal co-Büchi automaton $\mathcal{U}$, and the architecture A is satisfiable modulo finite integer arithmetic iff the specification is finite-state realizable in the architecture A.* □

Lemma 2. *For a specification, represented as a universal co-Büchi automaton $\mathcal{U} = (2^V, 2^{O_{env}}, Q, q_0, \delta, F)$, and an architecture A, the inferred constraint system for distributed synthesis has size $O(|\delta| \cdot |V| + |O_{env}| \cdot |2^{O_{env}}| + \sum_{p \in P^-} |I_p \smallsetminus O_{env}|)$.* □

The main parameters of the constraint system for distributed synthesis are the bound b_A on the size of the transition system $\mathcal{T}_A$ and the family $\{b_p \mid p \in P^-\}$ of bounds on the process transition systems $\{\mathcal{T}_p \mid p \in P^-\}$. If we use these parameters to unravel the constraint system completely (i.e., if we resolve the universal quantification explicitly), the resulting transition system is linear in b_A, and quadratic in b_p.

Theorem 8. *For a given specification, represented as a universal co-Büchi automaton $\mathcal{U} = (2^V, 2^{O_{env}}, Q, q_0, \delta, F)$, an architecture $A = (P, V, I, O)$, a bound b_A*

4. $\forall t.\, \lambda_1^{\mathbb{B}}(t) \rightarrow \neg g_1(d_1(t)) \vee \neg g_2(d_2(t))$
7. $\forall t.\, \lambda_2^{\mathbb{B}}(t) \wedge \neg g_1(d_1(t)) \rightarrow \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1\overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda_2^{\#}(t)$
$\wedge\ \lambda_2^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_2^{\#}(t)$
$\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1\overline{r}_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1\overline{r}_2}(t)) > \lambda_2^{\#}(t)$
$\wedge\ \lambda_2^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_2^{\#}(\tau_{r_1 r_2}(t)) > \lambda_2^{\#}(t)$
8. $\forall t.\, \lambda_3^{\mathbb{B}}(t) \wedge \neg g_2(d_2(t)) \rightarrow \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1\overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda_3^{\#}(t)$
$\wedge\ \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_3^{\#}(t)$
$\wedge\ \lambda_3^{\mathbb{B}}(\tau_{r_1\overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1\overline{r}_2}(t)) > \lambda_3^{\#}(t)$
$\wedge\ \lambda_3^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 r_2}(t)) > \lambda_3^{\#}(t)$
9. $\forall t.\, d_1(\tau_{r_1 r_2}(t)) = d_1(\tau_{r_1\overline{r}_2}(t)) \wedge d_1(\tau_{\overline{r}_1 r_2}(t)) = d_1(\tau_{\overline{r}_1\overline{r}_2}(t))$
$\wedge\ d_2(\tau_{r_1 r_2}(t)) = d_2(\tau_{\overline{r}_1 r_2}(t)) \wedge d_2(\tau_{r_1\overline{r}_2}(t)) = d_2(\tau_{\overline{r}_1\overline{r}_2}(t))$
10. $\forall t, u.\, d_1(t) = d_1(u) \wedge \big(g_2(d_2(t)) \leftrightarrow g_2(d_2(u))\big) \rightarrow d_1(\tau_{r_1 r_2}(t)) = d_1(\tau_{r_1 r_2}(u))$
$\wedge\ d_1(\tau_{\overline{r}_1 r_2}(t)) = d_1(\tau_{\overline{r}_1 r_2}(u))$
11. $\forall t, u.\, d_2(t) = d_2(u) \wedge \big(g_1(d_1(t)) \leftrightarrow g_1(d_1(u))\big) \rightarrow d_2(\tau_{r_1 r_2}(t)) = d_2(\tau_{r_1 r_2}(u))$
$\wedge\ d_2(\tau_{r_1\overline{r}_2}(t)) = d_2(\tau_{r_1\overline{r}_2}(u))$

Fig. 5. Example of a constraint system for distributed synthesis. The figure shows modifications and extensions to the constraint system from Figure 4 for the arbiter example (Figure 2) in order to implement the arbiter in the distributed architecture shown in Figure 1b.

on the size of the input-preserving transition system $\mathcal{T}_A$, and a family $\{b_p \mid p \in P^-\}$ of bounds on the process transition systems $\{\mathcal{T}_p \mid p \in P^-\}$, the unraveled constraint system has size $O(b_A \cdot (|\delta| \cdot |V| + |O_{env}| \cdot |2^{O_{env}}|) + \sum_{p \in P^-} {b_p}^2 |I_p \smallsetminus O_{env}|))$. It is satisfiable if and only if the specification is bounded realizable in A for the bounds b_A and $\{b_p \mid p \in P^-\}$. □

Example. As an example for the reduction of the distributed synthesis problem to SAT, we consider the problem of finding a distributed implementation to the arbiter specified by the universal automaton of Figure 2 in the architecture of Figure 1b. The functions d_1 and d_2 are the mappings to the processes p_1 and p_2, which receive requests r_1 and r_2 and provide grants g_1 and g_2, respectively. Figure 5 shows the resulting constraint system. Constraints 1–3, 5, and 6 are the same as in the fully informed case (Figure 4). The consistency constraints 9–11 guarantee that processes p_1 and p_2 show the same behavior on all input histories they cannot distinguish.

7 Conclusions

Despite its obvious advantages, synthesis has been less popular than verification. While the complexity of verification is determined by the size of the implementation under analysis, standard synthesis algorithms [1, 8, 9, 2] suffer from the daunting complexity determined by the theoretical upper bound on the smallest implementation, which, as shown by Rosner [3], increases by an extra exponent with each additional process in the architecture.

By introducing a bound on the size of the implementation, we have levelled the playing field for synthesis and verification. We have shown that the bounded synthesis problem can be solved effectively with a reduction to SAT.

Our solution for the bounded synthesis problem can be extended to the standard (unbounded) synthesis problem by iteratively increasing the bound. The advantage of this approach is that the complexity is determined by the size of the smallest actual implementation. Typically, this is far less than the exploding upper bound.

References

1. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. FOCS, IEEE Computer Society Press (1990) 746–757
2. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. LICS, IEEE Computer Society Press (2005) 321–330
3. Rosner, R.: Modular Synthesis of Reactive Systems. PhD thesis, Weizmann Institute of Science, Rehovot, Israel (1992)
4. Copty, F., Fix, L., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.: Benefits of bounded model checking at an industrial setting. In: Proc. of CAV. LNCS, Springer Verlag (2001)
5. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers **58** (2003) 118–149
6. Gu, J., Purdom, P.W., Franco, J., Wah, B.W.: Algorithms for the satisfiability (SAT) Problem: A survey. In Du, D.Z., Gu, J., Pardalos, P., eds.: Satisfiability Problem: Theory and applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society (1997) 19–152
7. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an Efficient SAT Solver. In: Proceedings of the 38th Design Automation Conference (DAC'01). (2001)
8. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Proc. LICS, IEEE Computer Society Press (2001) 389–398
9. Walukiewicz, I., Mohalik, S.: Distributed games. In: Proc. FSTTCS'03, Springer-Verlag (2003) 338–351
10. Madhusudan, P., Thiagarajan, P.S.: Distributed controller synthesis for local specifications. In: Proc. ICALP, Springer-Verlag (2001) 396–407
11. Castellani, I., Mukund, M., Thiagarajan, P.S.: Synthesizing distributed transition systems from global specification. In: Proc. FSTTCS. (1999) 219–231
12. Kupferman, O., Vardi, M.: Safraless decision procedures. In: Proc. 46th IEEE Symp. on Foundations of Computer Science, Pittsburgh (2005) 531–540
13. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. LICS, IEEE Computer Society (2006) 255–264