SMT-Based Synthesis of Distributed Systems*

Bernd Finkbeiner
Universität des Saarlandes
finkbeiner@uni-sb.de

Sven Schewe
Universität des Saarlandes
schewe@uni-sb.de

ABSTRACT

We apply SMT solving to synthesize distributed systems from specifications in linear-time temporal logic (LTL). The LTL formula is translated into an equivalent universal co-Büchi tree automaton. The existence of a finite transition system in the language of the automaton is then specified as a quantified formula in the theory $(\mathbb{N}, <)$ of the ordered natural numbers with uninterpreted function symbols. While our experimental results indicate that the resulting satisfiability problem is generally out of reach for the currently available SMT solvers, the problem immediately becomes tractable if we fix an upper bound on the number of states in the distributed system. After replacing each universal quantifier by an explicit conjunction, the SMT solver Yices solves simple single-process synthesis problems within a few seconds, and distributed synthesis problems, such as a two-process distributed arbiter, within one minute.

1. INTRODUCTION

Synthesis automatically *derives* correct implementations from specifications. Compared to verification, which only *proves* that a given implementation is correct, this has the advantage that there is no need to manually write and debug the code.

For temporal logics, the synthesis problem has been studied in several variations, including the synthesis of *closed* and *single-process* systems [2, 12, 6, 7], pipeline and ring architectures [9, 8, 11], as well as general *distributed* architectures [4]. Algorithms for synthesizing distributed systems typically reduce the synthesis problem in a series of automata transformations to the non-emptiness problem of a tree automaton. Unfortunately, the transformations are expensive: for example, in a pipeline architecture, each process requires a powerset construction and therefore causes an exponential blow-up in the number of states.

Inspired by the success of bounded model checking [3, 1], we recently proposed an alternative approach based on a reduction of the synthesis problem to a satisfiability problem [10]. Our starting point is the representation of the LTL specification as a universal co-Büchi tree automaton. 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. We define a constraint system that specifies 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 global transition system 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 global transition system.

In this paper, we report on preliminary experience applying the new approach with the SMT solver Yices. The result of the reduction is a quantified formula in the theory $(\mathbb{N}, <)$ of the ordered natural numbers with uninterpreted function symbols. The formula contains only a single quantifier (over the states of the implementation, represented as natural numbers).

While our experimental results indicate that proving the satisfiability of the quantified formulas is currently not possible (Yices reports "unknown"), the problem immediately becomes tractable if we fix an upper bound on the number of states. After replacing each universal quantifier by an explicit conjunction, Yices solves simple single-process synthesis problems within a few seconds, and distributed synthesis problems, such as a two-process distributed arbiter, within one minute.

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 determine whether there is an implementation for each system process in A, such that the composition of the implementations satisfies φ.

*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).

AFM'07, November 6, Atlanta, GA, USA.

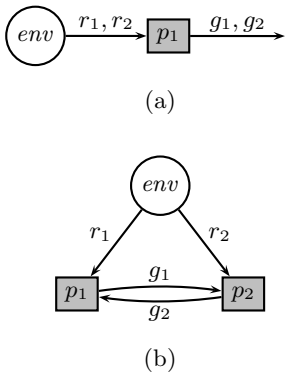

Figure 1: Example architectures: (a) single-process arbiter (b) two-process arbiter

2.1 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.

Figure 1 shows two example architectures, a single-process arbiter and a two-process arbiter. In the architecture in Figure 1a, the arbiter is a single process (p_1), which receives requests (r_1, r_2) from the environment (env) and reacts by sending grants (g_1, g_2). In the architecture in Figure 1b, the arbiter is split into two processes (p_1, p_2), which each receive one type of request (p_1 receives r_1; p_2 receives r_2) and react by sending the respective grant (p_1 sends g_1; p_2 sends g_2).

2.2 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.

2.3 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.

2.4 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.

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

A parity automaton is called a *Büchi* automaton if the image of α is contained in $\{1,2\}$ and a *co-Büchi* automaton iff the image of α is contained in $\{0,1\}$. 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. 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.

3. ANNOTATED TRANSITION SYSTEMS

In this section, we discuss 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.

THEOREM 1. *[5] 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.* □

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 $\{\_\} \cup \{0,\dots,c\}$, and *bounded* if it is c-bounded for some $c \in \mathbb{N}$. An annotation is *valid* if it satisfies the following conditions:

1. the pair (q_0, t_0) of initial states is annotated with a natural number ($\lambda(q_0,t_0) \neq \_$), and
2. 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, $\lambda(q',\tau(t,\upsilon)) \triangleright_{q'} n$ where $\triangleright_{q'}$ is $>$ for $q' \in F$ and $\geq$ otherwise.

THEOREM 2. *[10] 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.*

4. SINGLE-PROCESS SYNTHESIS

Using the annotation function, we reduce the non-emptiness problem of the universal co-Büchi tree automaton to an SMT problem. This solves the synthesis problem for single-process systems.

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 defined in Section 5, have a likewise compact representation.

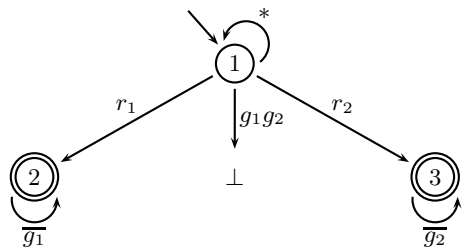

Figure 2: Specification of a simple arbiter, represented as a universal co-Büchi automaton. The states depicted as double circles (2 and 3) are the rejecting states in F.

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 $\lambda^{\mathbb{B}}_q$ and a unary function symbol $\lambda^{\#}_q$. Intuitively, $\lambda^{\mathbb{B}}_q$ 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 *progress* constraints (modulo finite integer arithmetic):
$\forall t.\, \lambda^{\mathbb{B}}_q(t) \;\wedge\; \underline{(q',\upsilon) \in \delta(q,\overrightarrow{a}(t))} \;\rightarrow\; \lambda^{\mathbb{B}}_{q'}(\tau_\upsilon(t)) \;\wedge\; \lambda^{\#}_{q'}(\tau_\upsilon(t)) \triangleright_q \lambda^{\#}_q(t)$, 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 the *initiality constraint* $\lambda^{\mathbb{B}}_{q_0}(0)$, 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 *global consistency* constraint $\forall t.\, a(\tau_\upsilon(t))$ if $a \in \upsilon$ and $\forall t.\, \neg a(\tau_\upsilon(t))$ if $a \notin \upsilon$. Additionally, we require the *root constraint* that the initial state is labeled with the root direction.

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.

Figure 3 shows the constraint system, resulting from the specification of an arbiter by the universal co-Büchi automaton depicted in Figure 2, implemented as a single process as required by the architecture of Figure 1a.

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.

5. 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.

The input visible to p consists of the fragment $I_p^{env} = O_{env} \cap I_p$ of environment variables visible to p, and the set $I_p^{sys} = I_p \setminus O_{env}$ of system variables visible to p. To encode the transition function τ_p, we introduce a $|I_p^{sys}| + 1$-ary function symbol τ_p^υ for every $\upsilon \subseteq I_p^{env}$. Intuitively, τ_p^υ maps the visible input $\upsilon' \subseteq I_p^{sys}$ of the system and a local position l of the transition system $\mathcal{T}_p$ to the $\upsilon \cup \upsilon'$-successor $\tau_p(l, \upsilon \cup \upsilon') = \tau_p^\upsilon(\upsilon', l)$ of l. This is formalized by the following *local consistency* constraints:

> $\forall t.\, \tau_p^\upsilon(a_1(d_{q_1}(t)), \ldots, a_n(d_{q_n}(t)); d_p(t)) = d_p(\tau_{\upsilon'}(t))$
> for all decisions $\upsilon' \subseteq O_{env}$ of the environment and their fragment $\upsilon = \upsilon' \cap I_p$ visible to p, where the variables $a_1, \ldots, a_n$ form the elements of I_p^{sys}.

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.

Example. As an example for the reduction of the distributed synthesis problem to SMT, 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 4 shows the resulting constraint system. Constraints 1–3, 5, and 6 are the same as in the fully informed case (Figure 3). The consistency constraints 9–10 guarantee that processes p_1 and p_2 show the same behavior on all input histories they cannot distinguish.

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. $\forall t.\, \lambda_3^{\mathbb{B}}(t) \wedge \neg g_2(t) \rightarrow \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda_3^{\#}(t) \wedge \lambda_3^{\mathbb{B}}(\tau_{\overline{r}_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{\overline{r}_1 r_2}(t)) > \lambda_3^{\#}(t) \wedge \lambda_3^{\mathbb{B}}(\tau_{r_1 \overline{r}_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 \overline{r}_2}(t)) > \lambda_3^{\#}(t) \wedge \lambda_3^{\mathbb{B}}(\tau_{r_1 r_2}(t)) \wedge \lambda_3^{\#}(\tau_{r_1 r_2}(t)) > \lambda_3^{\#}(t)$

Figure 3: Example of a constraint system for the synthesis of a single-process system. The figure shows the constraint system for the arbiter example (Figure 2). The arbiter is to be implemented as a single process as shown in Figure 1a.

6. EDGE-BASED ACCEPTANCE

A variation of our construction is to start with a tree automaton that has an edge-based acceptance condition instead of the standard state-based acceptance condition of the automata of Theorem 1. Since the progress constraints refer to edges rather than states, this often leads to a significant reduction in the size of the constraint system.

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. $\forall t.\, \tau_1^{r_1}\big(g_2(d_2(t)), d_1(t)\big) = d_1(\tau_{r_1 r_2}(t)) = d_1(\tau_{r_1\overline{r}_2}(t))$
$\wedge\ \tau_1^{\overline{r}_1}\big(g_2(d_2(t)), d_1(t)\big) = d_1(\tau_{\overline{r}_1 r_2}(t)) = d_1(\tau_{\overline{r}_1\overline{r}_2}(t))$

10. $\forall t.\, \tau_2^{r_2}\big(g_1(d_1(t)), d_2(t)\big) = d_2(\tau_{r_1 r_2}(t)) = d_2(\tau_{\overline{r}_1 r_2}(t))$
$\wedge\ \tau_2^{\overline{r}_2}\big(g_1(d_1(t)), d_2(t)\big) = d_2(\tau_{r_1\overline{r}_2}(t)) = d_2(\tau_{\overline{r}_1\overline{r}_2}(t))$

Figure 4: Example of a constraint system for distributed synthesis. The figure shows modifications and extensions to the constraint system from Figure 3 for the arbiter example (Figure 2) in order to implement the arbiter in the distributed architecture shown in Figure 1b.

For universal automata, the transition function δ can be described as a set of edges $E_\delta \subseteq Q \times \Sigma \times Q \times \Upsilon$ with

$$e = (q, \sigma, q', \upsilon) \in E_\delta \Leftrightarrow (q', \upsilon) \text{ is a conjunct of } \delta(q, \sigma).$$

For an edge-based universal co-Büchi automaton $\mathcal{E} = (\Sigma, \Upsilon, Q, E, F)$, the acceptance is defined by a finite set $F \subseteq E$ of rejecting edges, and $\mathcal{E}$ accepts an input tree if all paths in the run graph contain only finitely many rejecting edges. A state-based acceptance condition can be viewed as a special case of an edge-based acceptance condition, where an edge is rejecting iff it originates from a rejecting state, and edge-based acceptance can be translated into state-based acceptance by splitting the states with outgoing accepting and rejecting edges. For an edge-based universal co-Büchi automaton $\mathcal{E}$, we only need to adjust the definition of valid annotations slightly to

2. if a pair (q, t) is annotated with a natural number $(\lambda(q,t) = n \neq \_)$ and $(q, o(t), q', \upsilon) = e \in E$ is an edge of $\mathcal{E}$, then $(q', \tau(t, \upsilon))$ is annotated with a greater number, which needs to be strictly greater if $e \in F$ is rejecting. That is, $\lambda(q', \tau(t, \upsilon)) \triangleright_e n$ where $\triangleright_e$ is $>$ for $e \in F$ and $\geq$ otherwise.

Example. Figure 5 shows an example of a universal co-Büchi word automaton with edge-based acceptance condition. The automaton extends the specification of the simple arbiter such that the arbiter may not withdraw a grant while the environment upholds the request. Nonstarvation is required whenever the grant is not kept forever by the other process. Describing the same property with a state-based acceptance conditions requires 40% more states.

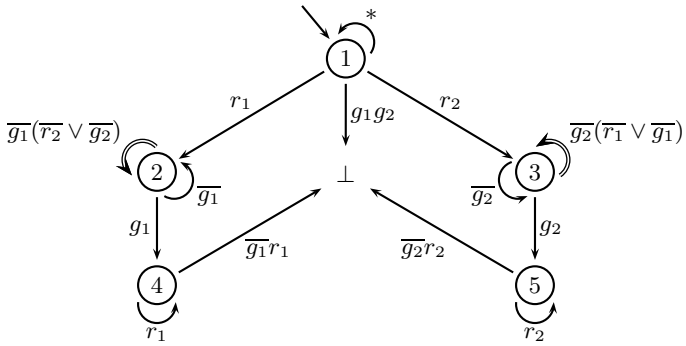

Figure 5: Extended specification of an arbiter, represented as a universal co-Büchi automaton with edge-based acceptance. The edges depicted as double-line arrows are the rejecting edges in F.

7. EXPERIMENTAL RESULTS

Using the reduction described in the previous sections, we considered five benchmarks; we synthesized implementations for simple arbiter specification from Figure 2 and the two architectures from Figure 1, and for a full arbiter specification and the two architectures from Figure 1, and we synthesized a strategy for dining philosophers to satisfy the specification from Figure 6. The arbiter examples are parameterized in the size of the transition system(s), the dining philosophers benchmark is additionally parameterized in the number of philosopher. As the SMT solver, we used Yices version 1.0.9 on a 2.6 Ghz Opteron system.

In all benchmarks, Yices is unable to directly determine the satisfiability of the quantified formulas. (For example the formulas from Figure 3 and Figure 4, respectively, for the monolithic and distributed synthesis in the simple arbiter example.) However, after replacing the universal quantifiers with explicit conjunctions (for a given upper bound on the number of states in the implementation), Yices solved all satisfiability problems quickly.

A single-process implementation of the arbiter needs 8 states. Table 1 shows the time and memory consumption of Yices when solving the SMT problem from Figure 3 with the quantifiers unravelled for different upper bounds on the number of states. The correct implementation with 8 states is found in 8 seconds.

7.1 Arbiter

Table 2 shows the time and memory consumption for the distributed synthesis problem. The quantifiers in the formula from Figure 4 were unravelled for different bounds on the size of the global transition system and for different bounds (shown in parentheses) on the size of the processes. A correct solution with 8 global states is found by Yices in 71 seconds if the number of process states is left unconstrained. Restricting the process states explicitly to 2 leads to an acceleration by a factor of two (36 seconds).

Table 3 and Table 4 show the time and memory consumption of Yices when solving the SMT problem resulting from the arbiter specification of Figure 5. The correct monolithic im-

bound	4	5	6	7	8	9
result	unsatisfiable	unsatisfiable	unsatisfiable	unsatisfiable	satisfiable	satisfiable
# decisions	3957	13329	23881	68628	72655	72655
# conflicts	209	724	1998	15859	4478	4478
# boolean variables	1011	2486	4169	9904	5214	5214
memory (MB)	16.9102	18.1133	20.168	27.4141	26.4375	26.4414
time (seconds)	0.05	0.28	1.53	35.99	7.53	7.31

Table 1: Experimental results from the synthesis of a single-process arbiter using the specification from Figure 2 and the architecture from Figure 1a. The table shows the time and memory consumption of Yices 1.0.9 when solving the SMT problem from Figure 3, with all quantifiers replaced by explicit conjunctions for different bounds on the number of states in the transition system.

bound	4	5	6	7	8	9	8 (1)	8 (2)
result	unsatisfiable	unsatisfiable	unsatisfiable	unsatisfiable	satisfiable	satisfiable	unsatisfiable	satisfiable
# decisions	6041	15008	35977	89766	197150	154315	178350	71074
# conflicts	236	929	2954	30454	33496	24607	96961	18263
# boolean variables	1269	2944	5793	9194	7766	8533	12403	6382
memory (MB)	17.0469	18.4766	22.1992	33.1211	37.4297	36.2734	39.4922	29.1992
time (seconds)	0.06	0.35	3.3	120.56	70.97	58.43	200.07	36.38

Table 2: Experimental results from the synthesis of a two-process arbiter using the specification from Figure 2 and the architecture from Figure 1b. The table shows the time and memory consumption of Yices 1.0.9 when solving the SMT problem from Figure 4, with all quantifiers replaced by explicit conjunctions for different bounds on the number of states in the global transition system and on the number of states in the individual processes (shown in parentheses).

bound	4	5	6	7	8
result	unsatisfiable	satisfiable	satisfiable	satisfiable	satisfiable
# decisions	17566	30011	52140	123932	161570
# conflicts	458	800	1375	2614	3987
# boolean variables	1850	2854	3734	5406	6319
memory (MB)	18.3008	20.0586	22.5781	27.5000	35.7148
time (seconds)	0.21	0.63	1.72	5.15	12.38

Table 3: Experimental results from the synthesis of a single-process arbiter using the specification from Figure 5 and the architecture from Figure 1a. The table shows the time and memory consumption of Yices 1.0.9 when solving the resulting SMT problem, with all quantifiers replaced by explicit conjunctions for different bounds on the number of states in the transition system.

bound	4	5	6	7	8	9	8 (1)	8 (2)	8 (3)	8 (4)
result	unsat	unsat	unsat	unsat	sat	sat	unsat	unsat	sat	sat
# decisions	16725	47600	91480	216129	204062	344244	309700	1122755	167397	208255
# conflicts	326	1422	8310	61010	11478	16347	92712	775573	13086	13153
# boolean variables	1890	7788	5793	13028	8330	10665	15395	25340	8240	7806
memory (MB)	18.0273	22.2109	28.5312	43.8594	42.2344	61.9727	54.1641	120.0160	42.1484	42.7188
time (seconds)	0.16	1.72	14.84	208.78	32.47	72.97	263.44	5537.68	31.12	30.36

Table 4: Experimental results from the synthesis of a two-process arbiter using the specification from Figure 5 and the architecture from Figure 1b. The table shows the time and memory consumption of Yices 1.0.9 when solving the resulting SMT problem, with all quantifiers replaced by explicit conjunctions for different bounds on the number of states in the global transition system and on the number of states in the individual processes (shown in parentheses).

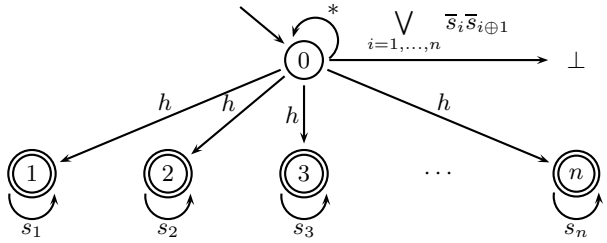

Figure 6: Specification of a dining philosopher problem with n philosophers. The environment can cause the philosophers to become hungry (by setting h to true). The states depicted as double circles (1 through n) are the rejecting states in F; state i refers to the situation where philosopher i is hungry and starving (s_i). The fail state is reached when two adjacent philosophers try to reach for their common chopstick; the fail state refers to the resulting eternal philosophical quarrel that keeps the affected philosophers from eating.

	3 states			4 states			6 states		
# philosophers	time (s)	memory (MB)	result	time (s)	memory (MB)	result	time (s)	memory (MB)	result
125	1.52	23.2695	unsat	23.84	36.2305	unsat	236.5	87.7852	sat
250	5.41	29.2695	unsat	130.07	52.0859	sat	141.36	91.1328	sat
375	22.81	38.9727	unsat	128.83	58.1992	unsat	890.58	154.355	sat
500	17.98	39.9297	unsat	15.84	52.9336	sat	237.04	119.309	sat
625	35.57	49.5586	unsat	417.05	94.7188	unsat	486.5	130.977	sat
750	22.25	52.3359	unsat	20.85	69.1562	sat	82.63	99.707	sat
875	51.98	56.0859	unsat	628.84	119.363	unsat	2546.88	255.965	sat
1000	168.17	70.3906	unsat	734.74	117.703	sat	46.18	124.691	sat
1125	67.14	70.1133	unsat	1555.18	165.922	unsat	1854.77	246.848	sat
1250	165.59	76.2227	unsat	122.8	107.645	sat	596.8	203.012	sat
1375	104.27	75.4531	unsat	3518.85	191.113	unsat	8486.18	490.566	sat
1500	187.25	82.8867	unsat	85.52	129.215	sat	232.81	214.68	sat
1625	85.83	88.8047	unsat	2651.82	246.734	unsat	1437.45	281.203	sat
1750	169.93	97.543	unsat	107.14	126.477	sat	257.77	185.887	sat
1875	174.03	105.25	unsat	3629.18	234.527	unsat	4641.03	405.781	sat
2000	25.86	102.125	unsat	242.55	157.734	sat	811.78	269.375	sat
2125	163.39	113.27	unsat	5932.24	315.711	unsat	6465.75	424.121	sat
2250	412.37	115.438	unsat	523.87	162.391	sat	5034.83	456.316	sat
2375	201.95	120.047	unsat	7311.03	313.168	unsat	4887.76	451.332	sat
2500	375.29	135.535	unsat	235.17	202.59	sat	319.78	253.781	sat
2625	544.03	135.379	unsat	6560.53	312.355	unsat	23990.5	808.633	sat
2750	559.35	139.137	unsat	817.41	226.082	sat	632.28	349.992	sat
2875	308.36	151.727	unsat	7273.89	299.016	unsat	8638.96	551.5	sat
3000	666.18	155.57	unsat	533.23	228.961	sat	3158.26	493.617	sat
3125	235.52	141.93	unsat	12596.6	377.328	unsat	10819.7	693.133	sat
3250	869.53	153.633	unsat	2089.72	308.719	sat	21298.8	889.285	sat
3375	260.88	145.918	unsat	11581.7	379.949	unsat	21560	741.09	sat
3500	308.23	169.348	unsat	897.6	270.676	sat	829.52	398.008	sat
5000	982.68	240.273	unsat	3603.7	421.832	sat	1357.48	582.457	sat
7000	2351.87	313.277	unsat	7069.55	535.98	sat	6438.73	1081.68	sat
10000	4338.83	448.648	unsat	4224.28	761.008	sat	10504.6	1121.58	sat

Table 5: Experimental results from the synthesis of a strategy for the dining philosophers using the specification from Figure 6. The table shows the time and memory consumption of Yices 1.0.9 when solving the resulting SMT problem, with all quantifiers replaced by explicit conjunctions for different bounds on the number of states in the transition system.

plementation with 5 states is found in less than one second, and Yices needs only half a minute to construct a correct distribute implementation. The table also shows that borderline cases like the fruitless search for an implementation with 8 states, but only 2 local states, can become very expensive; in the example, Yices needed more than $1\frac{1}{2}$ hours to determine unsatisfiability. Compromising on optimality, by slightly increasing the bounds, greatly improves the performance. Searching for an implementation with 8 states and 3 local states takes about 30 seconds.

7.2 Dining Philosophers

Table 5 shows the time and memory consumption for synthesizing a strategy for the dining philosophers to satisfy the specification shown in Figure 6. In the dining philosophers benchmark, the size of the specification grows linearly with the number of philosophers; for 10.000 philosophers this results in systems of hundreds of thousands constraints. In spite of the large size of the resulting constraint system, the synthesis problem remains tractable; Yices solves all resulting constraint systems within a few hours, and within a minutes for small constraint systems with up to 1000 philosophers.

8. CONCLUSIONS

Our experimental results suggest that the synthesis problem can be solved efficiently using satisfiability checking as long as a reasonable bound on the size of the implementation can be set in advance. In general, distributed synthesis is undecidable. By iteratively increasing the bound, our approach can be used as a semi-decision procedure.

Bounded synthesis thus appears to be a promising new application domain for SMT solvers. Clearly, there is a lot of potential for improving the performance. For example, Yices is not able to determine the satisfiability of the quantified formula directly. After applying a preprocessing step that replaces universal quantification by explicit conjunctions, Yices solves the resulting satisfiability problem within seconds. Developing specialized quantifier elimination heuristics could be an important step in bringing synthesis to practice.

9. REFERENCES

[1] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[2] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. IBM Workshop on Logics of Programs*, pages 52–71. Springer-Verlag, 1981.

[3] F. Copty, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. of CAV*, LNCS. Springer Verlag, 2001.

[4] B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. LICS*, pages 321–330. IEEE Computer Society Press, June 2005.

[5] O. Kupferman and M. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, Pittsburgh, October 2005.

[6] O. Kupferman and M. Y. Vardi. Synthesis with incomplete informatio. In *Proc. ICTL*, pages 91–106, Manchester, July 1997.

[7] O. Kupferman and M. Y. Vardi. μ-calculus synthesis. In *Proc. MFCS*, pages 497–507. Springer-Verlag, 2000.

[8] O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. LICS*, pages 389–398. IEEE Computer Society Press, July 2001.

[9] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.

[10] S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th International Symposium on Automated Technology for Verification and Analysis (ATVA 2007)*. Springer Verlag, 2007.

[11] I. Walukiewicz and S. Mohalik. Distributed games. In *Proc. FSTTCS'03*, pages 338–351. Springer-Verlag, 2003.

[12] P. Wolper. *Synthesis of Communicating Processes from Temporal-Logic Specifications*. PhD thesis, Stanford University, 1982.