Parameterized Synthesis*

Swen Jacobs[1] and Roderick Bloem[2]

[1] École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
swen.jacobs@epfl.ch
[2] IAIK, Graz University of Technology, Austria
roderick.bloem@iaik.tugraz.at

Abstract. We study the synthesis problem for distributed architectures with a parametric number of finite-state components. Parameterized specifications arise naturally in a synthesis setting, but thus far it was unclear how to decide realizability and how to perform synthesis. Using a classical result from verification, we show that for specifications in LTL\X, parameterized synthesis of token ring networks is equivalent to distributed synthesis of a network consisting of a few copies of a single process. Adapting a result from distributed synthesis, we show that the latter problem is undecidable. We then describe a semi-decision procedure based on bounded synthesis and show applicability on a simple case study. Finally, we sketch a general framework for parameterized synthesis based on cut-off results for verification.

1 Introduction

Synthesis is the problem of turning a temporal logical specification into a reactive system [1, 2]. In synthesis, parameterized specifications occur very naturally. For instance, Piterman, Pnueli, and Sa'ar illustrate their GR(1) approach with two parameterized examples of an arbiter and an elevator controller [3]. Similarly, the case studies given in [4, 5] consist of a parameterized specification of the AMBA bus arbiter. A simple example of a parameterized specification may be

$$\forall i.\ \mathsf{G}(r_i \rightarrow \mathsf{F}\, g_i) \wedge \forall i \neq j.\ \mathsf{G}(\neg g_i \vee \neg g_j).$$

This specification describes an arbiter serving an arbitrary number of clients, say n. Client i receives an input r_i for requests and controls an output g_i for grants. The specification states that for each client i, a request r_i is eventually followed by a grant g_i, but grants never occur simultaneously.

Previous approaches have focused on the synthesis of such systems for a fixed n. The question whether such a specification is realizable for *any* n is natural: it occurs, for instance, in the work on synthesis of processes for the leader election problem by Katz and Peled [6]. Only an answer to this question can determine whether a parameterized specification is correct. A further natural question is

* This work was supported by the Austrian Science Fund (FWF) under the RiSE National Research Network (S11406) and by the Swiss NSF Grant #200021_132176.

how to construct a parameterized system, i.e., a recipe for quickly constructing a system for an arbitrary n. Such a construction would avoid the steep increase of runtime and memory use with n that current tools incur [4, 5, 25].

Parameterized systems have been studied extensively in the context of verification. It is well known that the verification of such systems is undecidable [7, 8], although it can be decided for some restricted cases. In particular, for restricted topologies, the problem of verifying a network of isomorphic processes of arbitrary size can be reduced to the verification of a small network [9, 10]. As a corollary, synthesis of a network of an arbitrary number of processes can be reduced to synthesis of a small network, as long as the restricted topology is respected. In this paper, we focus on token ring topologies [9].

The question of synthesis of token rings is thus equivalent to the synthesis of a small network of isomorphic processes. This question is closely related to that of distributed synthesis [11–13]. Distributed synthesis is undecidable for all systems in which processes are incomparable with respect to their information about the environment. Our problem is slightly different in that we only consider specifications in LTL\X and that our synthesis problem is *isomorphic*, i.e., processes have to be identical. Unfortunately, this problem, and thus the original problem of parameterized synthesis, is also undecidable.

Having obtained a negative decidability result, we turn our attention to a semi-decision procedure, namely bounded synthesis [14, 15], an approach that searches for systems with a bounded number of states. We modify this approach to deal with isomorphic token-passing systems. Bounded synthesis reduces the problem of realizability to an SMT formula, a model of which gives an implementation of the system. Using Z3 [16], we show that a simple parameterized arbiter can be synthesized in reasonable time. Finally, we sketch a framework that extends our approach to the more general topologies of [10], and other classes of systems and specifications, in particular those that allow a cut-off for the corresponding verification problem.

2 Preliminaries

We consider the synthesis problem for distributed systems, with specifications in (fragments of) LTL. Given a system architecture A and a specification φ, we want to find implementations of all system processes in A, such that their composition satisfies φ.

Architectures. An *architecture* A is a tuple $(P, \mathit{env}, V, I, O)$, where P is a finite set of processes, containing the environment process env and system processes $P^- = P \setminus \{\mathit{env}\}$, V is a set of boolean system variables, $I = \{I_i \subseteq V \mid i \in P^-\}$ assigns a set I_i of boolean input variables to each system process, and $O = \{O_i \subseteq V \mid i \in P\}$ assigns a set O_i of boolean output variables to each process, such that $\dot{\bigcup}_{i \in P} O_i = V$. In contrast to output variables, inputs may be shared between processes. Wlog., we use natural numbers to refer to system processes, and assume $P^- = \{1, \ldots, k\}$ for an architecture with k system processes.

Implementations. An *implementation* $\mathcal{T}_i$ of a system process i with inputs I_i and outputs O_i is a labeled transition system (LTS) $\mathcal{T}_i = (T_i, t_i, \rho_i, o_i)$, where T_i is a set of states including the initial state t_i, $\rho_i : T_i \times \mathcal{P}(I_i) \rightarrow T_i$ a transition function, and $o_i : T_i \rightarrow \mathcal{P}(O_i)$ a labeling function.

The *composition* of the set of system process implementations $\{\mathcal{T}_1, \ldots, \mathcal{T}_k\}$ is the LTS $\mathcal{T}_A = (T_A, t_0, \rho, o)$, where the states are $T_A = T_1 \times \cdots \times T_k$, the initial state $t_0 = (t_1, \ldots, t_k)$, the labeling function $o : T_A \rightarrow \mathcal{P}(\dot{\bigcup}_{1 \leq i \leq k} O_i)$ with $o(t_1, \ldots, t_k) = o_1(t_1) \cup \cdots \cup o_k(t_k)$, and finally the transition function $\rho : T_A \times \mathcal{P}(O_{env}) \rightarrow T_A$ with

$$\rho((t_1, \ldots, t_k), e) = (\rho_1(t_1, (o(t_1, \ldots, t_k) \cup e) \cap I_1), \ldots, \rho_k(t_k, (o(t_1, \ldots, t_k) \cup e) \cap I_k)),$$

i.e., every process advances according to its own transition function and input variables, where inputs from other system processes are interpreted according to the labeling of the current state.

A *run* of an LTS (T, t_0, ρ, o) is an infinite sequence $(t^0, e^0), (t^1, e^1), \ldots$, where $t^0 = t_0$, $e^i \subseteq O_{env}$ and $t^{i+1} = \rho(t^i, e^i)$. An LTS *satisfies* a formula φ if for every run, the sequence $o(t^0) \cup e^0, o(t^1) \cup e^1, \ldots$ is a model of φ.

Asynchronous Systems. An *asynchronous system* is an LTS such that in every transition, only a subset of the system processes changes their state. This is decided by a *scheduler*, which can choose in every step which of the processes (including the environment) is allowed to make a step. In our setting, we will assume that the environment is always scheduled, and consider the scheduler as a part of the environment.

Formally, O_{env} contains additional scheduling variables $s_1, \ldots, s_k$, and $s_i \in I_i$ for every i. We require $\rho_i(t, I) = t$ for any i and set of inputs I with $s_i \notin I$.

Token Rings. We consider a class of architectures called *token rings*, where the only communication between system processes is a token. At any time only one process can possess the token, and a process i which has the token can pass it to process $i + 1$ by raising an output $\mathsf{send}_i \in O_i \cap I_{i+1}$. For processes in token rings of size k, addition and subtraction is done modulo k.

We assume that token rings are implemented as asynchronous systems, where in every step only one system process may change its state, except for token-passing steps, in which both of the involved processes change their state.

Distributed Synthesis. The *distributed synthesis problem* for a given architecture A and a specification φ, is to find implementations for the system processes of A, such that the composition of the implementations $\mathcal{T}_1, \ldots, \mathcal{T}_k$ satisfies φ, written $A, (\mathcal{T}_1, \ldots, \mathcal{T}_k) \models \varphi$. A specification φ is *realizable* with respect to an architecture A if such implementations exist. Synthesis and checking realizability of LTL specifications have been shown to be undecidable for architectures in which not all processes have the same information wrt. environment outputs in the synchronous case [12], and even for all architectures with more than one system process in the asynchronous case [13].

Bounded Synthesis. The *bounded synthesis problem* for given architecture A, specification φ and a family of bounds $\{b_i \in \mathbb{N} \mid i \in P^-\}$ on the size of system processes as well as a bound b_A for the composition $\mathcal{T}_A$, is to find implementations $\mathcal{T}_i$ for the system processes such that their composition $\mathcal{T}_A$ satisfies φ, with $|T_i| \leq b_i$ for all process implementations, and $|T_A| \leq b_A$.

3 Parameterized Synthesis

In this section, we introduce the parameterized synthesis problem. Using a classical result for the verification of token rings by Emerson and Namjoshi [9], we show that parameterized synthesis for token ring architectures and specifications in LTL\X can be reduced to distributed synthesis of isomorphic processes in a ring of fixed size. We then show that for this class of architectures and specifications, the isomorphic distributed synthesis problem is still undecidable.

3.1 Definition

Parameterized Architectures and Specifications. Let $\mathcal{A}$ be the set of all architectures. A *parameterized architecture* is a function $\Pi : \mathbb{N} \to \mathcal{A}$. A *parameterized token ring* is a parameterized architecture R with $R(n) = (P_n, env, V_n, I_n, O_n)$, where

- $P_n = \{env, 1, \ldots, n\}$,
- I_n is such that all system processes are assigned isomorphic sets of inputs, consisting of the token-passing input send_{i-1} from process $i-1$ and a set of inputs from the environment, distinguished by indexing each input with i.
- Similarly, O_n assigns isomorphic, indexed sets of outputs to all system processes, with $\mathsf{send}_i \in O_n(i)$, and every output of env is indexed with all values from 1 to n.

A *parameterized specification* φ is an LTL specification with indexed variables, and universal quantification over indices. We say that a parameterized architecture Π and a process implementation $\mathcal{T}$ *satisfy* a parameterized specification (written $\Pi, \mathcal{T} \models \varphi$) if for any n, $\Pi(n), (\mathcal{T}, \ldots, \mathcal{T}) \models \varphi$.

Example 1. Consider the parameterized token ring R_{arb} with $R_{arb}(n) = (P_n, env, V_n, I_n, O_n)$, where

$$
\begin{aligned}
P_n &= \{env, 1, \ldots, n\} && (1)\\
V_n &= \{r_1, \ldots, r_n, g_1 \ldots, g_n, \mathsf{send}_1, \ldots, \mathsf{send}_n\} && (2)\\
I_n(i) &= \{r_i, \mathsf{send}_{i-1}\} && (3)\\
O_n(env) &= \{r_1, \ldots, r_n\} && (4)\\
O_n(i) &= \{g_i, \mathsf{send}_i\} && (5)
\end{aligned}
$$

The architecture $R(n)$ defines a token ring with n system processes, with each process i receiving an input r_i from the environment and another input send_{i-1} from the previous process in the ring, and an output send_i to the next process, as well as an output g_i to the environment.

An instance of this parameterized architecture for $n = 4$ is depicted in Fig. 1. Together with the parameterized specification from Section 1, we will use it in Section 5 to synthesize process implementations for a parameterized arbiter.

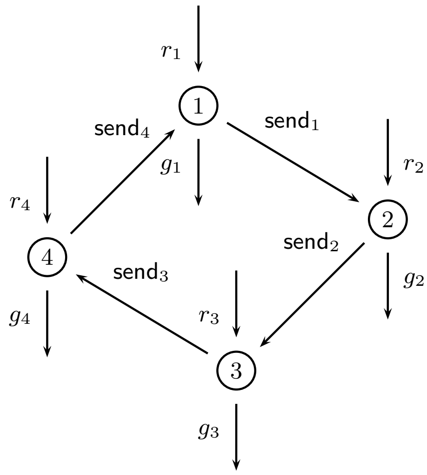

Fig. 1. Token ring architecture with 4 processes

Isomorphic and Parameterized Synthesis. The *isomorphic synthesis problem* for an architecture A and a specification φ is to find an implementation $\mathcal{T}$ for all system processes $(1, \ldots, k)$ such that $A, (\mathcal{T}, \ldots, \mathcal{T}) \models \varphi$. The *parameterized synthesis problem* for a parameterized architecture Π and a parameterized specification φ is to find an implementation $\mathcal{T}$ for all system processes such that $\Pi, \mathcal{T} \models \varphi$. The *parameterized (isomorphic) realizability problem* is the question whether such an implementation exists.

3.2 Reduction of Parameterized to Isomorphic Synthesis

Emerson and Namjoshi [9] have shown that verification of LTL\X properties for implementations of parameterized token rings can be reduced to verification of a small ring with up to five processes, depending on the form of the specification.

Theorem 1 ([9]). *Let R be a parameterized token ring, $\mathcal{T}$ an implementation of the isomorphic system processes that ensures fair token passing, and φ a parameterized specification. Then*

a) If $\varphi = \forall i.\ f_i$, where f_i is a formula that only refers to variables indexed by i, then $R, \mathcal{T} \models \varphi \iff R(2), \mathcal{T} \models \varphi$
b) If $\varphi = \forall i.\ f_{i,i+1}$, where $f_{i,i+1}$ is a formula that only refers to variables indexed by i and $i+1$, then $R, \mathcal{T} \models \varphi \iff R(3), \mathcal{T} \models \varphi$
c) If $\varphi = \forall i \neq j.\ f_{i,j}$, where $f_{i,j}$ is a formula that only refers to variables indexed by i and j, then $R, \mathcal{T} \models \varphi \iff R(4), \mathcal{T} \models \varphi$
d) If $\varphi = \forall i \neq j.\ f_{i,i+1,j}$, where $f_{i,i+1,j}$ is a formula that only refers to variables indexed by i, $i+1$, and j, then $R, \mathcal{T} \models \varphi \iff R(5), \mathcal{T} \models \varphi$

This theorem implies that verification of such structures is decidable. For synthesis, we obtain the following corollary:

Corollary 1. *For a given parameterized token ring R and parametric specification φ, parameterized synthesis can be reduced to isomorphic synthesis in rings of size 2 (3, 4, 5) for specifications of type a) (b, c, d, resp.).*

In the following, we will show that this reduction in general does not make the synthesis problem decidable.

3.3 Decidability

The parameterized synthesis problem is closely related to the distributed synthesis problem [11, 12]. We will use a modification of the original undecidability proof for distributed systems to show undecidability of isomorphic synthesis in token rings, which in turn implies undecidability of parameterized synthesis.

Theorem 2. *The isomorphic realizability problem is undecidable for token rings with 2 or more processes and specifications in LTL\X.*

Proof. The proof follows that of Pnueli and Rosner [11] (see also Finkbeiner and Schewe [12]). The original proof is for two synchronous processes, neither of which can observe the inputs or outputs of the other. The proof builds a specification that allows a single implementation, and forces the two processes to each simulate a Turing machine and halt. Thus, it is realizable iff the Turing machine halts, which shows undecidability. We will show that we can specify (in LTL\X) an asynchronous system in a token ring that simulates the behavior of these two synchronous processes. The proof works for rings of arbitrary size, if we assume that the specification is the same for all processes.

In the original proof, each process has a start signal that triggers the processes to output the next configuration of the Turing machine. The specification assumes that the number of start signals for the two processes is never different by more than one and requires that the configurations that are output by the two processes are either equal (if the number of start signals is equal) or that they are successors (if the number of start signals is off by one). This is easily specified because the processes are synchronized by a global clock.

We need to modify the original proof such that it works for asynchronous systems, and the specification can be written without the X operator.

This can be achieved by forcing the asynchronous system to simulate a synchronous system by using the token for synchronization: We augment the specification to assume that the token starts at a designated process, say 1. A clock cycle consists of a full cycle of the token, and we require that each process changes its output only once in each cycle. Thus, the asynchronous system simulates a synchronized system, where the synchronous states consist of the state of the asynchronous system immediately after the token passes to 1. Using tok_1 to identify these states, it is now possible to correlate the states of the simulated system: for instance, $\mathsf{X}\, q_i$ for the synchronous system corresponds to $\neg\mathsf{tok}_1\ \mathsf{W}\ (\mathsf{tok}_1 \wedge q_i)$ for the asynchronous system, and $\mathsf{G}\, q_i$ corresponds to $\mathsf{G}(\neg\mathsf{tok}_1 \implies \neg\mathsf{tok}_1\, \mathsf{W}\, \mathsf{tok}_1 \wedge q_i)$. This allows us to translate the construction in [11] to our setting, and remove all occurrences of X in the specification.

Finally, the token cannot be used to pass any additional information (beyond the synchronization): the only freedom a process has is *when* to pass the token, and by lack of a global clock and visibility of the input and output signals of the other processes, a given process cannot measure this time or observe any changes of the system during this time.

Thus, our asynchronous system simulates the synchronous system from [11] and is realizable iff the Turing machine halts. □

Combining Theorems 1 and 2, we obtain the following result.

Theorem 3. *The parametric realizability problem is undecidable for token rings and specifications of type (a), (b), (c), or (d).*

Proof. By Theorem 1, the isomorphic realizability problem for a specification of type (a) and two processes can be reduced to a parameterized realizability problem of type (a). Since the former problem is undecidable, so is the latter. The proof for cases (b)–(d) is analogous. □

4 Bounded Isomorphic Synthesis

The reduction from Section 3 allows us to reduce parameterized synthesis to isomorphic synthesis with a fixed number of processes. Still, the problem does not fall into a class for which the distributed synthesis problem is decidable.

For distributed architectures that do not fall into decidable classes, Finkbeiner and Schewe have introduced the semi-decision procedure of *bounded synthesis* [14, 15], which converts an undecidable distributed synthesis problem into a sequence of decidable synthesis problems, by bounding the size of the implementation. In the following, we will show how to adapt bounded synthesis for isomorphic synthesis in token rings, which by Corollary 1 amounts to parameterized synthesis in token rings.

4.1 Bounded Synthesis

The bounded synthesis procedure consists of three main steps:

Step 1: Automata translation. Following an approach by Kupferman and Vardi [17], the LTL specification φ (including fairness assumptions like fair scheduling) is translated into a universal co-Büchi-automaton $\mathcal{U}$ which accepts an LTS $\mathcal{T}$ iff $\mathcal{T}$ satisfies φ.

Step 2: SMT Encoding. Existence of an LTS which satisfies φ is encoded into a set of SMT constraints over the theory of integers and free function symbols. States of the LTS are represented by natural numbers, state labels as free functions of type $\mathbb{N} \to \mathbb{B}$, and the global transition function as a free function of type $\mathbb{N} \times \mathbb{B}^{|O_{env}|} \to \mathbb{N}$. Transition functions of individual processes are defined indirectly by introducing projections $d_i : \mathbb{N} \to \mathbb{N}$, mapping global to local states. To ensure that local transitions of process i only depend on inputs in I_i, we add a constraint

$$\forall i.\ \forall t, t'.\ \forall I, I'.\ d_i(t) = d_i(t') \wedge I \cap I_i = I' \cap I_i \ \to\ d_i(\tau(t, I)) = d_i(\tau(t', I')).$$

To obtain an interpretation of these symbols that satisfies the specification φ, additional annotations of states are introduced. This includes labels $\lambda_q^{\mathbb{B}} : \mathbb{N} \to \mathbb{B}$ and free functions $\lambda_q^{\#} : \mathbb{N} \to \mathbb{N}$, which are defined such that (i) $\lambda_q^{\mathbb{B}}(t)$ is true iff the product of $\mathcal{T}$ and $\mathcal{U}$ contains a path from an initial state to a state (t, q) with $q \in Q$, i.e., the product automaton can reach a state in which $\mathcal{U}$ is in q, among other states, and (ii) valuations of the $\lambda_q^{\#}$ must be increasing along paths of $\mathcal{U}$, and strictly increasing for transitions that enter a rejecting state of $\mathcal{U}$. Together, this ensures that an LTS satisfying these constraints cannot have runs which enter rejecting states infinitely often (and thus would be rejected by $\mathcal{U}$).

Step 3: Iteration for Increasing Bounds. To obtain a decidable problem, we restrict the number of states in the LTS that we are looking for, which allows us to instantiate all quantifiers over state variables t, t' explicitly with all values in the given range. If the constraints are unsatisfiable for a given bound, we increase it and try again. If they are satisfiable, we obtain a model, giving us an implementation for the system processes such that φ is satisfied.

4.2 Adaption to Token Rings

We adapt the bounded synthesis approach for synthesis in token rings, and introduce some optimizations we found vital for a good performance of the synthesis method.

Additional Constraints and Optimizations. We use some of the general modifications and optimizations mentioned in [15]:

- We use an additional constraint to ensure that the resulting system implementation is asynchronous. In general, we could directly add a constraint $\forall i.\ \forall I.\ s_i \notin I \to d_i(\tau(t, I)) = d_i(t)$ (where I is a set of inputs and s_i is the scheduling variable for process i). For the particular case of token rings we use a modified version, explained below.

- We use symmetry constraints to encode that all processes should be isomorphic. Particularly, we use the same function symbols for state labels of all system processes, and special constraints for the local transition functions, also explained below.
- We use the semantic variant where environment inputs are not stored in system states, but are directly used in the transition term that computes the following state. This results in an implementation which is a factor of $|O_{env}|$ smaller.[3]

Encoding Token Rings. For the particular case of token rings, we use the following modifications to the SMT encoding:

- We want to obtain an asynchronous system in which the environment is always scheduled, along with exactly one system process. Thus, we do not need $|P|$ scheduling variables, but can encode the index of the scheduled process into a binary representation with $log_2(|P^-|)$ inputs.
- We encode the special features of token rings: i) exactly one process should have the token at any time, ii) only a process which has the token can send it, iii) if process i is scheduled, currently has the token and wants to send it, then in the next state process $i+1$ has the token and process i does not, and iv) if process i has the token and does not send it (or is not scheduled), it also has the token in the next state. Properties ii) – iv) are encoded in the following constraints, where $\mathsf{tok}_i((d_i(t))$ is true in state t iff process i has the token, $\mathsf{send}(d_i(t))$ is true iff i is ready to send the token, and $\mathsf{sched}_i(I)$ is true iff the scheduling variables in I are such that process i is scheduled:

$$\begin{array}{ll}
\forall i.\ \forall t.\ \forall I. & \mathsf{tok}(d_i(t)) \rightarrow (\mathsf{send}(d_i(t)) \wedge \mathsf{sched}_i(I)) \vee \mathsf{tok}(d_i(\tau(t,I))) \\
\forall i.\ \forall t. & \neg\mathsf{tok}(d_i(t)) \rightarrow \neg\mathsf{send}(d_i(t)) \\
\forall i.\ \forall t.\ \forall I. & \mathsf{send}(d_i(t)) \wedge \mathsf{sched}_i(I) \rightarrow \neg\mathsf{tok}(d_i(\tau(t,I))) \\
\forall i.\ \forall t.\ \forall I. & \mathsf{send}(d_{i-1}(t)) \wedge \mathsf{sched}_i(I) \rightarrow \mathsf{tok}(d_i(\tau(t,I)))
\end{array}$$

  We do not encode property i) directly, because it is implied by the remaining constraints whenever we start in a state where only one process has the token.
- Token passing is an exception to the rule that only the scheduled process changes its state: if process i is scheduled in state t, and both $\mathsf{tok}(d_i(t))$ and $\mathsf{send}(d_i(t))$ hold, then in the following transition both processes i and $i+1$ will change their state. The constraint which ensures that only scheduled processes may change their state is modified into

$$\begin{array}{ll}
\forall i.\ \forall t.\ \forall I. & \neg\mathsf{sched}_i(I) \wedge \neg(\mathsf{sched}_{i-1}(I) \wedge \mathsf{tok}(d_{i-1}(t)) \wedge \mathsf{send}(d_{i-1}(t))) \\
& \rightarrow d_i(\tau(t,I)) = d_i(t)
\end{array}$$

- Finally, we need to restrict local transitions in order to obtain isomorphic processes. The general rule is that local transitions of process i should only

[3] The different semantics (compared to the input-preserving LTSs used in [14, 15]) is already reflected in our definition of LTSs and satisfaction of LTL formulas.

depend on the local state and inputs in I_i. With our definition, token passing is an exception to this rule. The resulting constraints for local transitions are:

$$\begin{aligned}
\forall i > 1.\ \forall t,t'.\ \forall I,I'.\ \ & d_1(t) = d_i(t') \wedge \mathsf{sched}_1(I) \wedge \mathsf{sched}_i(I') \\
& \rightarrow\ d_1(\tau(t,I)) = d_i(\tau(t',I')) \\
\forall i > 1.\ \forall t,t'.\ \forall I,I'.\ \ & d_1(t) = d_i(t') \wedge \mathsf{send}(d_n(t)) \wedge \mathsf{send}(d_{i-1}(t')) \\
& \wedge\ \mathsf{sched}_n(I) \wedge \mathsf{sched}_{i-1}(I') \wedge I \cap I_1 = I' \cap I_i \\
& \rightarrow\ d_1(\tau(t,I)) = d_i(\tau(t',I'))
\end{aligned}$$

Fairness of Scheduling and Token Passing. A precondition of Thm. 1 is that the implementation needs to ensure fair token-passing. Thus, we always add

$$\forall i.\ \mathsf{fair\_scheduling}\ \rightarrow\ (\mathsf{G}(\mathsf{tok}_i\ \rightarrow\ \mathsf{F}\,\mathsf{send}_i))$$

to φ, where $\mathsf{fair\_scheduling}$ stands for $\forall j.\ \mathsf{G}\,\mathsf{F}\,\mathsf{sched}_j$. Note that with this condition, the formula does not fall into any of the cases from Thm. 1. However, in the model of Emerson and Namjoshi, fairness of scheduling is an implicit assumption, since otherwise fairness of token passing will also be violated. Thus, by adding this formula, we are making explicit two of the assumptions of Emerson and Namjoshi, and this formula does not need to be taken into account when choosing which case of the theorem needs to be applied.

Similarly, the $\mathsf{fair\_scheduling}$ assumption needs to be added to any liveness conditions of the specification, as without fair scheduling in general liveness conditions cannot be guaranteed. As before, this does not need to be taken into account considering Thm. 1.

Correctness and Completeness of Bounded Synthesis for Token Rings. Based on completeness of the original bounded synthesis approach (and correct modeling of the features of token rings), we obtain

Corollary 2. *If a given specification φ is satisfiable in a token ring of a given size n, then the bounded synthesis algorithm, adapted to token rings, will eventually find this implementation.*

Finally, based on the correctness of our adaption of bounded synthesis, and Corollary 1, we obtain

Theorem 4. *If a given specification φ falls into class a (b,c,d) of Thm. 1 and the adapted bounded synthesis algorithm finds an implementation that satisfies φ in a token ring of size 2 (3,4,5), then this implementation satisfies φ in token rings of arbitrary size.*

5 Synthesizing a Parameterized Arbiter

In this section, we show how parameterized synthesis can be used to obtain process implementations for token ring architectures. Our example is a parameterized arbiter in a token ring as depicted in Fig. 1, with the following specification:

$$\begin{array}{ll} \forall i \neq j. & \mathsf{G}\,\neg(g_i \wedge g_j) \\ \forall i. & (\mathsf{G}(r_i \rightarrow \mathsf{F}\,g_i)) \end{array}$$

Every process i has an input r_i for requests from the environment, which it can grant by activating an output g_i. We want grants of all processes to be mutually exclusive, and every request to be eventually followed by a grant. The specification satisfies case c) in Theorem 1, i.e., a ring of size 4 is sufficient to synthesize implementations that satisfy the specification for rings of any size.

According to the adapted bounded synthesis approach from Sect. 4.2, we need to add the token fairness requirement, and add the fair scheduling assumption to all liveness constraints. This results in the extended specification

$$\begin{array}{ll} \forall i \neq j. & \mathsf{G}\,\neg(g_i \wedge g_j) \\ \forall i. & \mathsf{fair\_scheduling} \rightarrow (\mathsf{G}(r_i \rightarrow \mathsf{F}\,g_i)) \\ \forall i. & \mathsf{fair\_scheduling} \rightarrow (\mathsf{G}(\mathsf{tok}_i \rightarrow \mathsf{F}\,\mathsf{send}_i)). \end{array}$$

We translate the specification into a universal co-Büchi automaton, shown for 2 processes in Fig. 2. This automaton translates to a set of first-order constraints for the annotations of an LTS implementing φ, a part of which is shown in Fig. 3 (only constraints for states $0, 1, 3, 5$ of the automaton are shown). These constraints, together with general constraints for asynchronous systems, isomorphic processes, token rings, and size bounds, are handed to Z3 [16]. For correctly chosen bounds ($|T_A| \leq 4$ and $|T_p| \leq 2$), we obtain a model of the process implementation in ∼10 seconds (on an Intel Core i7 CPU @ 2.67 GHz). The solution is very simple: every process needs only 2 states, with send_i and g_i signals high iff the process has the token. In the parallel composition of 4 such processes, only 4 global states are reachable. Theorem 4 guarantees that with this process implementation, φ will be satisfied for any instance of the architecture.

Note that synthesis is "easy" in this case because we can restrict it to a small ring of 4 processes, and have a rather simple specification. For 5 processes (and $|T_A| \leq 5$), Z3 already needs ∼100 seconds to solve the resulting constraints. We expect similar increases in needed time for specifications with more system variables.

The translation of specifications into SMT constraints is currently not fully automated. We leave the development of an automatic tool and its application to more complex case studies for future work.

6 A Framework for Parameterized Synthesis

Our approach for reduction of parameterized synthesis to distributed/isomorphic synthesis is not limited to token rings. In the following, we sketch a framework which allows us to lift decision procedures for the verification of parameterized systems to semi-decision procedures for their synthesis.

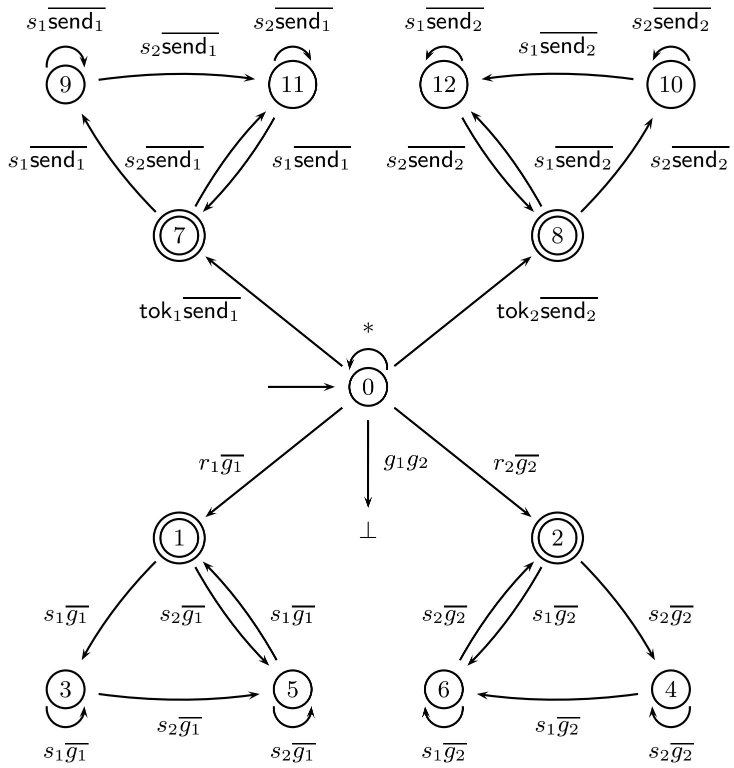
$s_1\overline{\mathsf{send}_1}$
$s_2\overline{\mathsf{send}_1}$
$s_2\overline{\mathsf{send}_1}$
$s_1\overline{\mathsf{send}_2}$
$s_1\overline{\mathsf{send}_2}$
$s_2\overline{\mathsf{send}_2}$
9
11
12
10
$s_1\overline{\mathsf{send}_1}$
$s_2\overline{\mathsf{send}_1}$
$s_1\overline{\mathsf{send}_1}$
$s_2\overline{\mathsf{send}_2}$
$s_1\overline{\mathsf{send}_2}$
$s_2\overline{\mathsf{send}_2}$
7
8
$\mathsf{tok}_1\overline{\mathsf{send}_1}$
$\mathsf{tok}_2\overline{\mathsf{send}_2}$
$*$
0
$r_1\overline{g_1}$
g_1g_2
$r_2\overline{g_2}$
$\perp$
1
2
$s_1\overline{g_1}$
$s_2\overline{g_1}$
$s_1\overline{g_1}$
$s_2\overline{g_2}$
$s_1\overline{g_2}$
$s_2\overline{g_2}$
3
5
6
4
$s_2\overline{g_1}$
$s_1\overline{g_2}$
$s_1\overline{g_1}$
$s_2\overline{g_1}$
$s_1\overline{g_2}$
$s_2\overline{g_2}$

Fig. 2. Universal co-Büchi automaton for specification φ

6.1 General Token-passing Systems

Clarke, Talupur, Touilli, and Veith [10] have extended the results of Emerson and Namjoshi to arbitrary token-passing networks. They reduce the parameterized verification problem to a finite set of model checking problems, where the number of problems and the size of systems to be checked depends on the architecture of the parameterized system and on the property to be proved.

To lift these results to the synthesis of parameterized token-passing systems in general, we need to adapt the bounded synthesis algorithm further, such that it searches for a process implementation which satisfies the required properties for all verification problems in the set determined by architecture and specification. This requirement can easily be encoded into corresponding constraints for the SMT solver, but may of course increase complexity of synthesis significantly.

Encoding of token-passing into SMT constraints must be adapted to the possibility that processes may be able to choose which other process will receive the token. Furthermore, Clarke et al. [10] have the assumption that the system satisfies fair token passing. For synthesis, we must strengthen the given specification of the system such that it will satisfy this property. For general token-passing networks, the assumption that every process that holds the token

$$
\begin{array}{ll}
 & \lambda^{\mathbb{B}}_0(0) \\
 & \mathsf{tok}(d_1(0)) \wedge \forall i \neq 1.\ \neg\mathsf{tok}(d_i(0)) \\
\forall t.\ \forall I. & \lambda^{\mathbb{B}}_0(t) \rightarrow \lambda^{\mathbb{B}}_0(\tau(t,I)) \wedge \lambda^{\#}_0(\tau(t,I)) \geq \lambda^{\#}_0(t) \\
\forall i \neq j.\ \forall t. & \lambda^{\mathbb{B}}_0(t) \rightarrow \neg(g(d_i(t)) \wedge g(d_j(t))) \\
\forall i.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_0(t) \wedge \mathsf{sched}_1(I) \wedge r_i \in I \rightarrow \lambda^{\mathbb{B}}_1(t) \wedge \lambda^{\#}_1(\tau(t,I)) > \lambda^{\#}_0(t) \\
\forall i \neq j.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_1(t) \wedge \neg\mathsf{sched}_2(I) \wedge \neg g(d_1(t)) \rightarrow \lambda^{\mathbb{B}}_3(t) \wedge \lambda^{\#}_3(\tau(t,I)) \geq \lambda^{\#}_1(t) \\
\forall i \neq j.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_1(t) \wedge \mathsf{sched}_2(I) \wedge \neg g(d_1(t)) \rightarrow \lambda^{\mathbb{B}}_5(t) \wedge \lambda^{\#}_5(\tau(t,I)) \geq \lambda^{\#}_1(t) \\
\forall i \neq j.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_3(t) \wedge \neg\mathsf{sched}_2(I) \wedge \neg g(d_1(t)) \rightarrow \lambda^{\mathbb{B}}_3(t) \wedge \lambda^{\#}_3(\tau(t,I)) \geq \lambda^{\#}_3(t) \\
\forall i \neq j.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_5(t) \wedge \neg\mathsf{sched}_1(I) \wedge \neg g(d_1(t)) \rightarrow \lambda^{\mathbb{B}}_5(t) \wedge \lambda^{\#}_5(\tau(t,I)) \geq \lambda^{\#}_5(t) \\
\forall i \neq j.\ \forall t.\ \forall I. & \lambda^{\mathbb{B}}_5(t) \wedge \mathsf{sched}_1(I) \wedge \neg g(d_1(t)) \rightarrow \lambda^{\mathbb{B}}_1(t) \wedge \lambda^{\#}_1(\tau(t,I)) > \lambda^{\#}_5(t) \\
\ldots & \ldots
\end{array}
$$

Fig. 3. Constraints that are equivalent to realizability of φ

will always eventually send it may not be enough to ensure fair token passing. One possibility to ensure fair token-passing in general networks is to require

$$\forall i.\forall j.\, \mathsf{G}(\mathsf{tok}_i \rightarrow \mathsf{F}\,\mathsf{send}_{(i,j)}),$$

where i quantifies over all processes as usually, j over all processes which can receive the token from process i, and $\mathsf{send}_{(i,j)}$ means that i sends the token to j.

6.2 Other Results with Cutoffs

In the literature, there is a vast body of work on the verification of parameterized systems, much of it going beyond token-passing systems (e.g., [18, 19]). In particular, many of these results prove a cutoff for the given class of systems and specifications [20–22], making the verification problem decidable.

In principle, any verification result that provides a cutoff, i.e., reduces the verification of LTL properties for parameterized systems to the verification of a finite set of fixed-size systems, can be used in a similar way to obtain a semi-decision procedure for the parameterized synthesis problem. Our limitation is the ability to encode the special features of the class of systems in decidable first-order constraints, and the specifications under consideration are omega regular.

Approaches that detect a cutoff for a given system implementation dynamically [23, 24] (i.e., not determined by architecture and specification) are less suited for our framework: they could in principle be integrated with our approach, but cutoff detection would have to be interleaved with generation of candidate implementations, making it hard or impossible to devise a complete synthesis approach.

7 Conclusions

We have stated the problem of parameterized realizability and parameterized synthesis: whether and how a parameterized specification can be turned into a simple recipe for constructing a parameterized system. The realizability problem asks whether a parameterized specification can be implemented for any number of processes, i.e., whether the specification is correct. The answer to the synthesis question gives a recipe that can quickly be turned into a parameterized system, thus avoiding the steeply rising need for resources associated with synthesis for increasing n using classical, non-parameterized methods.

We have considered the problem in detail for token rings, and to some extent for general token-passing topologies. Using results from parameterized verification, we showed that the parameterized synthesis problem reduces to distributed synthesis of a small network of isomorphic processes with fairness constraints on token passing. Unfortunately, the synthesis problem remains undecidable.

Regardless of this negative result, we managed to synthesize an actual—albeit very small—example of a parameterized arbiter. To this end, we used Schewe and Finkbeiner's results on bounded synthesis. In theory, this approach will eventually find an implementation if it exists. In practice, this currently only works for small implementations. One line of future work will be on making synthesis feasible for larger systems, possibly as an extension of the lazy synthesis approach [25].

For unrealizable specifications, our approach will run forever. It is an interesting question whether it could be combined with incomplete methods to check unrealizability.

We note that the topologies we considered do limit communication between processes and therefore also the possible solutions. For our running example, processes give grants only when they hold the token. Obviously, this means that response time increases linearly with the number of processes, something that can be avoided in other topologies. The use of more general results on parameterized verification may widen the class of topologies that we can synthesize.

Acknowledgments. Many thanks to Leonardo de Moura for his help with little known features of Z3. We thank the members of ARiSE, particularly Helmut Veith, for stimulating discussions on parameterized synthesis, and Bernd Finkbeiner for discussions on distributed and bounded synthesis. Finally, thanks to Hossein Hojjat for useful comments on a draft of this paper.

References

1. Church, A.: Logic, arithmetic and automata. In: Proceedings International Mathematical Congress. (1962)
2. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. Symposium on Principles of Programming Languages (POPL '89). (1989) 179–190
3. Piterman, N., Pnueli, A., Sa´ar, Y.: Synthesis of reactive(1) designs. In: VMCAI. (2006) 364–380
4. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Specify, compile, run: Hardware form PSL. In: 6th International Workshop on Compiler Optimization Meets Compiler Verification. (2007) Electronic Notes in Theoretical Computer Science.

5. Bloem, R., Galler, S., Jobstmann, B., Piterman, N., Pnueli, A., Weiglhofer, M.: Automatic hardware synthesis from specifications: A case study. In: In Proceedings of the Design, Automation and Test in Europe. (2007) 1188–1193
6. Katz, G., Peled, D.: Synthesizing solutions to the leader election problem using model checking and genetic programming. In: HVC. (2009) 117–132
7. Apt, K., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Information Processing Letters **22** (1986) 307–309
8. Suzuki, I.: Proving properties of a ring of finite state machines. Information processing Letters **28** (1988) 213–214
9. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. International Journal of Foundations of Computer Science **14** (2003) 527–549
10. Clarke, E.M., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: Proc. Concurrency Theory. (2004) 276–291
11. Pnueli, A., Rosner, R.: Distributed systems are hard to synthesize. In: Proc. Foundations of Computer Science (FOCS). (1990) 746–757
12. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Logic in Computer Science (LICS), IEEE Computer Society Press (2005) 321–330
13. Schewe, S., Finkbeiner, B.: Synthesis of asynchronous systems. In: LOPSTR. Volume 4407 of LNCS. (2006) 127–142
14. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Automated Technology for Verification and Analysis (ATVA'07). (2007) 474–488
15. Finkbeiner, B., Schewe, S.: Bounded synthesis. Software Tools for Technology Transfer (to appear).
16. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. (2008) 337–340
17. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: FOCS. (2005) 531–542
18. Clarke, E.M., Grumberg, O., Jha, S.: Verifying parameterized networks. ACM Trans. Program. Lang. Syst. **19**(5) (1997) 726–750
19. Zuck, L.D., Pnueli, A.: Model checking and abstraction to the aid of parameterized systems (a survey). Computer Languages, Systems & Structures **30**(3-4) (2004) 139–169
20. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3) (1992) 675–735
21. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE. (2000) 236–254
22. Kahlon, V., Ivancic, F., Gupta, A.: Reasoning about threads communicating via locks. In: CAV. (2005) 505–518
23. Hanna, Y., Basu, S., Rajan, H.: Behavioral automata composition for automatic topology independent verification of parameterized systems. In: ESEC/SIGSOFT FSE. (2009) 325–334
24. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: CAV. (2010) 645–659
25. Finkbeiner, B., Jacobs, S.: Lazy synthesis. In: VMCAI. (2012) (to appear).