Synthesizing Locally Symmetric Parameterized Protocols from Temporal Specifications

Ruoxi Zhang
University of Waterloo
Waterloo, Canada
r378zhan@uwaterloo.ca

Richard Trefler
University of Waterloo
Waterloo, Canada
trefler@uwaterloo.ca

Kedar S. Namjoshi
Nokia Bell Labs
Murray Hill, USA
kedar.namjoshi@nokia-bell-labs.com

Abstract—Scalable protocols and web services are typically parameterized: that is, each instance of the system is formed by linking together isomorphic copies of a representative process. Verification of such systems is difficult due to state explosion for large instances and the undecidability of verifying properties over all instances at once. This work turns instead to the derivation of a parameterized protocol from its specification. We exploit a reduction theorem showing that it suffices to construct a representative process \( P \) that meets a local specification under interference by neighboring copies of \( P \). Every instance of the parameterized protocol is built by deploying replicated instances of \( P \). While the reduction from the original to a local specification is done by hand, the construction of \( P \) is fully automated. This is a new and challenging synthesis question, as one must synthesize an unknown process \( P \) while simultaneously considering interference by copies of this unknown process. We present two algorithms: an eager reduction to the synthesis of a transformed specification, and a lazy, iterative, tableau construction which incorporates fresh interference at each step. The tableau method has worst-case complexity that is exponential in the length of the local specification. We have implemented the tableau construction and show that it is capable of synthesizing parameterized protocols for mutual exclusion, leader election, and dining philosophers.

I. INTRODUCTION

Scalable systems, such as network communication protocols, distributed algorithms, and multi-core hardware models, are typically parameterized — that is, they are composed of many isomorphic copies of a representative process. These processes interact with each other according to an underlying communication scheme. Automated verification of such systems quickly runs into state explosion with increasing instance size, as an instance with \( K \) processes can have a reachable state space that is exponential in \( K \). The alternative of “once and for all” verification of all instances at once is undecidable in general [1].

In this work, we turn instead to the construction (synthesis) of a parameterized system from its specification. The key to the presented methodology is a compositional (i.e. assume-guarantee) reduction theorem from [2] which exploits the symmetry inherent in these systems, showing that it suffices to verify that a localized property holds of a representative process \( P \) under interference from neighboring copies of \( P \). The first step of the methodology is to reduce the global specification of the desired parameterized system to a localized property. This reduction varies by application, as the global specification is itself parameterized and quantified (e.g., “all instances satisfy mutual exclusion”) while the local specification is quantifier-free. The second step is to synthesize an appropriate process \( P \) from the local specification, which is carried out automatically.

This synthesis question is of a new and challenging type. The standard formulation of temporal synthesis is to construct a process \( P \) satisfying a given temporal specification \( \varphi \). However, our reduction requires the construction of a process \( P \) whose closure under interference by copies of (the unknown) \( P \) satisfies a temporal specification \( \varphi \). That is, the synthesis procedure must somehow derive a suitable process while simultaneously taking into account the effects of interference by adjacent copies of this unknown process. Every instance of the protocol is built by deploying replicated instances of the synthesized \( P \).

We provide two algorithms for the synthesis question. The first is an ‘eager’ method that transforms a given specification \( \varphi \) to a new specification \( \mathcal{I}(\varphi) \) which incorporates self-interference; one can then apply standard synthesis methods to \( \mathcal{I}(\varphi) \). The second is a ‘lazy’ method which iteratively constructs a sequence of tableaux starting with a tableau for \( \varphi \); at each iteration, the current tableau is extended with interference transitions. The limit tableau is then pruned to obtain the solution. Although the eager method is direct, the transformation from \( \varphi \) to \( \mathcal{I}(\varphi) \) always incurs an exponential blowup in the number of proposition symbols in \( \varphi \). For this reason, we implement the lazy method and show that it can synthesize solutions for mutual exclusion, leader election, and dining philosophers specifications.

This approach does not provide a complete solution to the parameterized synthesis question, for several reasons. The first is that the reduction from a quantified global specification to an unquantified local specification is carried out by hand. The second is that the process \( P \) to be derived can only have a fixed-size neighborhood, as otherwise one would require an unbounded quantification over the neighbors of \( P \). Hence, the method can derive solutions for rings, tori, wrap-around mesh, and other networks where the degree of a node is independent of the number of nodes in an instance. (The use of localized abstractions, e.g., [3], may help bypass this limitation; we plan to investigate this in future work.) Finally, both algorithms produce a process \( P \) where any two

https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_30

This article is licensed under a Creative Commons Attribution 4.0 International License
states that satisfy the same propositions have identical future behavior. This rules out the synthesis of auxiliary state beyond that defined by propositional valuations. Nonetheless, despite these limitations, one can synthesize correct-by-construction parameterized protocols for the specifications listed above.

In the sequel, for ease of exposition, we limit attention to parameterized protocols on ring networks. Local symmetry ensures that a single representative process suffices. This process has two neighbors, one to the left and one to the right. Specifications are expressed in CTL, augmented with unconditional fairness on process schedules. The tableau method builds on the classical tableau constructions for CTL and Fair CTL and their associated synthesis procedures based on pruning states and transitions from the tableau.

It is worth noting that while the construction of a single instance with a fixed number of processes is a closed synthesis question, the derivation of a representative for all instances is an open synthesis question.

II. PRELIMINARIES

A. Rings: Structure, Semantics, and Interference

A ring of size $K$ is a directed graph with node set $N = \{0..K\}$, and edge set $\{E_i\}$ for $i \in \{0..K\}$. Node $i$ is connected to edges $E_i$ (on its left) and $E_{i+1}$ (on its right). (Arithmetic is implicitly modulo $K$.) Edge $E_i$ is connected to nodes $(i-1)$ (on its left) and $i$ (on its right). Two nodes are neighbors if they have a common connected edge. The set of neighbors of node $i$ is denoted $nbr(i)$.

The parameterized networks of interest are uniform rings of arbitrary size, in that the process at each node is a copy of a single ‘tile’ process (cf. [2]). Figure 1 shows a tile and the construction of an instance through replication. The external variables of a process are those assigned to adjacent edges. (In the figure, an incoming arrow represents read access; an outgoing arrow represents write access.) A process may also have internal state variables assigned to the node.

For readability, we denote the representative process by $P_n$, so we can speak of its neighbors as $P_{n-1}$ and $P_{n+1}$ (and either of them as $P_m$). It is important that $P_n$ is not viewed as the $n$’th process in a particular instance but rather as the representative process for all instances.

The external and internal variables of $P_n$ together form the state space of $P_n$, which is the collection of valuations to these variables. The state machine for $P_n$ is a tuple $(S_n, S^0_n, T_n, \lambda_n)$, where $S_n$ is the state space; $S^0_n$ is a non-empty set of initial states; $T_n \subseteq S_n \times S_n$ is a transition relation; and $\lambda_n : S_n \rightarrow 2^{S^0_n}$ is a function that labels each state with a subset of atomic propositions from the set $\Sigma_n$.

As defined, the state machine of $P_n$ is a labeled state transition system that describes the behavior of the representative process alone in its neighborhood. A neighbor can interfere with $P_n$ by changing the values of commonly shared (necessarily external) variables. A joint state is a pair of states $(s, t)$, with $s$ from $P_n$ and $t$ from a neighbor $P_m$ that agree on the valuation to their shared variables. A joint transition from joint state $(s, t)$ to joint state $(s', t')$ by process $P_m$ is defined if $(t, t')$ is in $T_m$ and the values of variables of $P_n$ that are not shared with $P_m$ are equal in $s$ and $s'$. We say that $(s, s')$ is an interference transition caused by $P_m$. For example, ‘$P_m$ passes a token to $P_n’$ is an interference transition.

We denote the $i$’th copy of the representative process $P_n$ in an instance by $P_i$. The $K$ process instance formed by copies $P_0\ldots P_{K-1}$ has the global state transition relation $G = (S, S^0, T, \lambda)$. Here each state $s \in S$ is a valuation to the internal variables of each process, together with a valuation to the external edge variables; $S^0$ is a non-empty set of initial states, where each state in $S^0$ projects to an initial state of $P_i$ for all $i$. The transition relation $T$ defines non-deterministic interleaving: $(s, i, s')$ is in $T$ if $(s[i], s'[i])$ is in $T_i$ and the value of any variables not in process $P_i$ is the same in $s$ and $s'$. Here, the notation $s[i]$ represents the projection of $s$ on the variables of $P_i$. The labeling $\lambda$ of a state $s$ is the indexed union of all local labelings $\lambda_i(s[i])$.

From $G$ one can define a machine $G_i$ by projecting out the labels of transitions other than those of the $i$’th process. I.e., consider a transition $(s, k, s')$ of $G$. If $k = i$, retain the transition as is; otherwise, replace the label with $\tau$.

The effect of interference on $P_n$ is given by a transition system $H^\theta_n$ defined in [2]; we repeat the definition here. A compositional inductive invariant $\theta$ of an instance is a set of local assertions $\{\theta_n\}$ with the following properties: for every $n$, (1) $\theta_n$ includes the initial states of $P_n$; (2) transitions by $P_n$ preserve $\theta_n$; and (3) interference transitions by $P_m$ from joint states satisfying $\theta_n$ and $\theta_m$ preserve $\theta_n$. These properties can be converted to simultaneous pre-fixpoint form over $\{\theta_n\}$. By the Knaster-Tarski theorem, the least fixpoint is the strongest compositional invariant, denoted by $\theta^*$.

States of $H^\theta_n$ are the local states $S_n$ that satisfy $\theta_n$; transitions of $H^\theta_n$ are of two types: (1) a transition by $P_n$, denoted $(s, n, s')$, where $\theta_n(s)$ holds and $(s, s')$ is in $T_n$, and (2) an interference transition denoted $(n, m, s')$ representing a transition by $P_m$ from a joint state $(s, t)$ where $\theta_n(s)$ and $\theta_m(t)$ hold, to a joint state $(s', t')$.

This transition system is linked to the global transition system with respect to local properties.

**Theorem II.1.** ([2]) $H^\theta_i$ stuttering-simulates $G_i$ for every $i$. Moreover, if $H^\theta_i$ satisfies an ‘outward-facing’ restriction, then $H^\theta_i$ and $G_i$ are stuttering-bisimular.

The systems are equivalent only up to stuttering as $H^\theta_i$ does not take into account transitions by processes ‘far away’.

![Figure 1. The tile of the dining philosophers protocol.](image-url)
from position \( i \), while \( G \) of course contains all transitions. The outward-facing restriction says (informally) that the interference by a neighboring process \( m \) depends only on the valuation of the variables shared by \( P_n \) and \( P_m \).

The transition system \( H_n^m \) induced by the strongest compositional invariant \( \theta^* \) is of special interest; we abbreviate it as \( H_n^* \). It is constructed by an inductive, least fixpoint process. (1) The initial structure \( H_n \) consists of the initial states of \( P_n \). Apply steps (2) or (3) in any fairly interleaved order until no new transitions can be added; the result is \( H_n^* \). Step (2) applies an enabled transition of \( P_n \) to a reachable state of \( H_n^* \), labeling it by \( n \). Step (3) views the currently reachable \( n \)-transitions of \( H_n \) as transitions from its (isomorphic) neighboring copy \( H_m \) and adds an enabled interference transition to a reachable state of \( H_n^* \), labeling it by \( m \).

B. Local Fair CTL

Let the scheduling of the process network be unconditionally fair. We use fair computation tree logic (Fair CTL) [4] to represent a local correctness property \( \varphi_n \), e.g., ‘\( P_n \) accesses the shared resource if \( P_n \) owns the token’. The induced parametric global correctness property is the conjunction \( \bigwedge_i \varphi_i \).

Syntax. The language of Fair CTL contains \( \Sigma_n \), Boolean operators \( \neg, \land, \lor, \Rightarrow, \Leftrightarrow \), linear time temporal operators \( X_n \), (process indexed strong next-time), \( Y_n \), (process indexed weak next-time), \( G \), (always), \( F \), (sometime), \( U \), (until), \( W \), (dual of until), and path quantifiers \( A \), \( E \) (for all paths), \( \exists \) (there exists a path).

We have the following syntax for Fair CTL. If \( p \in \Sigma_n \), then \( p \) is a formula. If \( f, g \) are formulae, then so are \( \neg f, f \land g, f \lor g, f \Rightarrow g, f \Leftrightarrow g, AX_n f, EX_n f, AF_n f, EG f, EF f, AF U g, EF U g, AF W g, W g \).

As given in [5], we use indexed next-time operators \( X_n \) and \( Y_n \) in place of the unindexed ones, where \( X_n f \) means that the immediate successor state \( s' \) (along any maximal path designated by a path quantifier) is reached by executing one step of \( P_n \), and \( f \) is true in \( s' \); and \( Y_n f \) means that if the immediate successor state \( s' \) (along any maximal path designated by a path quantifier) is reached by executing one step of \( P_n \), then \( f \) is true in \( s' \).

Globally, unconditionally fair scheduling asserts that all processes are selected for execution infinitely often by the scheduler. Locally, the fairness assumption is expressed as ‘\( P_n \) and its neighbors are executed infinitely often’. The path quantifiers \( A \) and \( E \) in Fair CTL are subtyped by the fixed local fairness assumption, \( \Phi \), indicating that quantifications are performed only on fair paths.

In Fair CTL, a path quantifier is followed by a linear-time temporal operator. The pairs are the basic modalities. A formula whose basic modality is \( A \), \( E \) is \( A \), \( E \), \( A \), \( E \), or \( E \), \( G \) is an eventuality formula corresponding to a liveness property. Formulae \( A \) and \( G \) are invariants corresponding to safety properties. In addition, we assume all formulae are converted into positive normal form, which means the negations are driven inwards to atomic propositions.

Semantics. A local Fair CTL formula \( \varphi_n \) is interpreted on the local state transition system \( H_n^* \) and the global state transition system \( G_n \). Let \( M = (S, S^0, T, \lambda) \) be a structure. A path, \( \pi = (s_0, s_1, \ldots) \), is a sequence of states such that \( (s_i, s_{i+1}) \in T \) for all \( i \), and \( \pi^\ast = (\tau_0, \tau_1, \ldots) \) is the suffix of \( \pi \) starting at state \( \tau \). A full path is an infinite path, and self-loops are allowed. A full path is fair if it satisfies \( \Phi \).

We use \( M, s \models f \) to mean that the formula \( f \) is true in \( M \) at state \( s \) under the fairness assumption \( \Phi \). We define \( \models \) inductively as follows:

- \( M, s \models p \) if \( p \in \lambda(s) \) for atomic proposition \( p \).
- \( M, s \models \neg f \) if not \( (M, s \models f) \).
- \( M, s \models f \land g \) if \( M, s \models f \) and \( M, s \models g \).
- \( M, s \models \exists_n f \) if there exists \( \pi = (s_0, s_1, \ldots) \) such that \( (s_0, s_1) \in T_n \), \( M, \pi \models \Phi \), and \( M, s_1 \models f \).
- \( M, s \models \forall_n f \) if for all \( \pi = (s_0, s_1, \ldots) \), if \( (s_0, s_1) \in T_n \) and \( M, \pi \models \Phi \), then \( M, s_1 \models f \).
- \( M, s \models \exists_g f \) if there exists \( \pi = (s_0, s_1, \ldots) \), such that \( M, \pi \models \Phi \), and there exists \( i \geq 0 \), such that \( M, s_i \models g \), and for all \( 0 \leq j < i \), \( M, s_j \not\models f \).
- \( M, s \models \forall_g f \) if for all \( \pi = (s_0, s_1, \ldots) \), if \( M, \pi \models \Phi \), then there exists \( i \geq 0 \), such that \( M, s_i \models g \), and for all \( 0 \leq j < i \), \( M, s_j \not\models f \).

By abbreviations, \( f \lor g \equiv \neg(\neg f \land \neg g), A(f W g) \equiv \neg E(\neg f U g), E(f W g) \equiv \neg A(\neg f U g), AG \equiv \neg EF \neg f, AE \equiv \neg AF \neg f \) (hence, \( A f = A \text{(trueWf)} \)), \( EF \equiv E(\text{trueWf}) \), \( AGf \equiv A(\text{falseWf}) \), and \( EGf \equiv E(\text{falseWf}) \).

A formula \( f \) is satisfiable if there exists a model \( M \) such that \( M, s \models f \) for some state \( s \) of \( M \).

C. Fairness and Outward-Facing

The local fairness assumption \( \Phi = F^\infty \bigwedge \bigwedge_m \bigwedge n \bigwedge_n F^\infty \bigwedge_n \).

The path formula \( F^\infty \bigwedge_n \) asserts that \( P_n \) is selected for execution infinitely often by the scheduler. The infinitary linear time operator \( F^\infty \) abbreviates \( GF \) and is interpreted as \( M, \pi \models F^\infty g \) iff for every \( i \geq 0 \), there exists \( j \geq i \), such that \( M, \pi^j \models g \).

Formally, outward-facing is defined relative to \( \Phi \), extending the definition in [2]. Let \( s \) and \( t \) be two states on \( H_n^* \): \( s \) and \( t \) are related by a relation \( B_{n,m} \) if \( s[e] = t[e] \) for every common connected edge \( e \) between \( n \) and \( m \). The notation \( s[e] \) denotes the value of the external variable assigned to \( e \) at \( s \). Process \( P_n \) is outward-facing in its interactions with \( P_m \) if \( B_{n,m} \) is a stuttering bisimulation on \( H_n^* \).

D. Parameterized Synthesis

We can now explain precisely how the reduction theorem supports parameterized synthesis.

Theorem II.2. Let \( \varphi_n \) be a local FairCTL specification. Let \( P_n \) be a process such that its derived \( H_n^* \) satisfies \( \varphi_n \). Every instance of the parameterized system constructed from isomorphic copies of \( P_n \) satisfies the global property \( \bigwedge_i \varphi_i \).

Proof. Consider \( P_n \) and its induced \( H_n^* \) which satisfies the local correctness property \( \varphi_n \). By symmetry, each copy \( P_i \) of
the representative $P_n$ has an isomorphic $H_i^n$ which satisfies the corresponding $\varphi_i$.

Consider an instance of the parameterized system constructed from isomorphic copies of $P_n$. Let $G$ be the global state space of the instance. Let $i$ be a node of the instance. By Theorem II.1, $G_i$ satisfies $\varphi_i$; hence, by the locality of $\varphi_i$, it follows that $G$ satisfies $\varphi_i$. As this holds for every node, $G$ satisfies the global property $(\forall_i \varphi_i)$. The first part of Theorem II.1, this ‘inflationary’ consequence holds for any universal Fair CTL property. It holds for all Fair CTL properties if $H_i^n$ is outward-facing.

The synthesis procedures of the following sections will, in effect, simultaneously construct both the strongest invariant $\theta^*$ and the resulting $H_i^n$.

III. EAGER SYNTHESIS

We describe the eager method of synthesizing a representative process $P_n$ whose interference closure $H_i^n$ satisfies the Fair CTL formula $\varphi_n$. The atomic propositions in $\varphi_n$ are divided into two disjoint groups: $X$, representing properties of the external state, and $L$, representing properties of the internal state. We use $a, b, a', b'$ to refer to valuations of variables in $X$, and $k, l, k', l'$ to refer to valuations of variables in $L$. The notation $X = a$ means that each variable in $X$ has the value given to it in $a$.

Given a local property $\varphi_n$, the eager method produces a Fair CTL formula $I(\varphi_n)$ that is a conjunction of $\varphi_n$ with several constraints. The constraints are expressed in CTL extended with the modal operators $\langle c \rangle$ and its negation dual $\langle \neg c \rangle$, where $\langle c \rangle$ is the set of states from which there is a transition labeled with propositions from $X$ satisfying $c$ to a state satisfying $f$. It is straightforward to adjust the Fair CTL synthesis procedure for this variant of the EX operator.

The candidate models are labeled transition systems where transitions are labeled either by $n$ (the representative) or by $m$ (a neighbor). States are labeled with propositions from $X$ and $L$. The constraints added to $\varphi_n$, intuitively, make the models ‘look’ similar to $H_i^n$.

A pair $(a, a')$ of valuations in $X$ is an interference pair if $\text{EF}((X = a) \land \langle n \rangle (X = a'))$ holds at the initial state of a candidate model; i.e., if there is a reachable state labeled $a$ with an $n$-successor labeled $a'$. By symmetry, the $n$-transition producing this pair may be viewed as an $m$-transition of a neighbor. A pair $(b, b')$ of valuations in $X$ is considered the result of interference by $(a, a')$ viewed as a neighboring $m$-transition if (1) the $X$-variables shared between $m$ and $n$ have the same valuations in $b$ and $a$, and in $b'$ and $a'$, and (2) the $X$-variables not shared between $m$ and $n$ have the same valuation in $b$ and $b'$. The set of such pairs is denoted $\iota_m(a, a')$.

The Fair CTL formula $I(\varphi_n)$ is the conjunction of $\varphi_n$ with the constraints (1)-(4) given below. The added constraints are expressible in CTL as $X$ and $L$ have finitely many valuations.

1) Every interference pair induces an interference transition at all matching states. I.e., for every interference pair $(a, a')$ and every $(b, b')$ in $\iota_m(a, a')$, the property $\text{AG}((X = b) \Rightarrow \langle m \rangle (X = b'))$ holds.

2) $m$-transitions do not modify local state. I.e., $\text{AG}((L = l) \Rightarrow [m](L = l))$ for every valuation $l$ of the local propositions.

3) Every $m$-transition is induced by an interference pair. I.e., for every $b, b'$ such that $\text{EF}((X = b) \land \langle m \rangle (X = b'))$, there is an interference pair $(a, a')$ such that $(b, b') \in \iota_m(a, a')$.

4) States with the same propositional label have similar successors. I.e., for $c$ ranging over $m$ and $n$: if $\text{EF}((X = a \land L = l) \land \langle c \rangle (X = a' \land L = l'))$ holds, then $\text{AG}((X = a \land L = l) \Rightarrow \langle c \rangle (X = a' \land L = l'))$.

A specification is realizable if it has a satisfying model.

Theorem III.1. $I(\varphi_n)$ is realizable if and only if there is a process $P_n$ with state space $2^X \times 2^L$ whose interference-closure $H_i^n$ satisfies $\varphi_n$.

Proof. We show that any solution to the right-hand condition induces a solution to $I(\varphi_n)$, and vice-versa.

From right-to-left, consider a process $P_n$ meeting the right-hand condition. We claim that $H_i^n$ satisfies conditions (1)-(4) by its inductive construction. If an interference pair $(a, a')$ becomes reachable at some stage of the construction, it is used to construct interference transitions at all subsequent stages; thus, condition (1) holds. Interference transitions do not modify local state, meeting condition (2). Moreover, all interference transitions stem from an interference pair introduced at an earlier stage, meeting condition (3). Finally, as the closure is defined over the same state space as $P_n$, there is a unique state for each propositional labeling, satisfying condition (4).

The proof for the left-to-right direction is more involved, as we cannot a priori restrict the models of $I(\varphi_n)$ to the state space $2^X \times 2^L$. Thus, consider any model $M_0$ of $I(\varphi_n)$. We may assume that every transition of $M_0$ is reachable. (If not, limiting $M_0$ to its reachable state space still satisfies $I(\varphi_n)$.)

Let $\sim$ be the relation defined by $s \sim t$ if states $s$ and $t$ satisfy the same propositions. Condition (4) implies that $\sim$ is a strong bisimulation on $M_0$. (Proof: Consider states $s, t$ such that $s \sim t$ and a $c$-successor $s'$ of $s$. Let $a, l$ be the propositions over $X$ and $L$ (respectively) that are satisfied by $s$, and let $a', l'$ be the corresponding propositions satisfied by $s'$. The transition from $s$ to $s'$ is a witness to the assumption of (4); hence $t$ must have a $c$-successor $t'$ satisfying $a', l'$. By definition, $s' \sim t'$ holds.)

Let $M_1$ be the quotient of $M_0$ under $\sim$. As $\sim$ is a strong bisimulation, $M_0$ and $M_1$ are strongly bisimilar; hence, both satisfy the same Fair CTL formulas; in particular, $M_1$ also satisfies $I(\varphi_n)$. Let process $P$ be the subgraph formed by the $n$-transitions of $M_1$. We show that $M_1$ is the interference closure of $P$.

Note that by the definition of $\sim$ and the quotient construction, every propositional valuation is associated with at most one state of $M_1$, so we can consider $M_1$ to be isomorphic to a process with state space $2^X \times 2^L$.

We first show that the interference closure of $P$ is a subgraph of $M_1$, by induction on the stages of the closure construction. Initially, that is true as $P$ is a subgraph of

238
Consider the transition added at the next step. If this is a
transition of \( P_n \), it is already present in \( M_1 \). If the transition
is an interference transition applied at a state \( s \), it must be
derived from an \( n \)-transition present at the current stage. By
the induction hypothesis, the inducing \( n \)-transition and the
state \( s \) both belong to \( M_1 \). By conditions (1) and (2), the
derived interference transition from \( s \) also belongs to \( M_1 \). It
follows that the closure process constructed as the limit of
these steps is a subgraph of \( M_1 \).

We also need to rule out the existence of transitions in \( M_1 \)
that are not in the closure process. Let \( t \) be a transition of
\( M_1 \), from a state \((b, k)\) to \((b', k')\). If this is a \( n \)-transition, it
belongs to \( P \) and hence to the closure. Consider the case where
it is an \( m \)-transition. By (2), \( k' \) must equal \( k \). From (3), there
is an interference pair \((a, a')\) in \( M_1 \) induced by an \( n \)-transition
\( t' \) such that \((b, a') \in t_n (a, a')\). The \( n \)-transition \( t' \) is in \( P \)
definition and hence in the closure. Therefore the interference
transition \( t \) induced by \( t' \) is also in the closure.

The eager method is technically interesting as it transforms
the new, self-referential synthesis question into a standard
form, simply by adding constraints that encode interference.
However, the transformation results in an exponential blowup
as the added constraints range over all propositional valuations.
Hence, this method is likely to be impractical. The following
section formulates a lazy procedure that gradually introduces
interference into a tableau of the original formula.

IV. THE TABLEAU APPROACH

A tableau of \( n \) is a tuple \( T_n = (V_n, R, L) \), where \( V_n \) is a
set of nodes; \( R \) is a transition relation over \( V_n \), and \( L : V_n \rightarrow 2\text{Prop} \) is a labeling function. A tableau has two types
of nodes, \( V_n = V_n^C \cup V_n^D \) such that \( V_n^C \cap V_n^D = \emptyset \), where
\( V_n^C \) is a set of AND-nodes that are potential states of \( P_n \),
and \( V_n^D \) is a set of OR-nodes. The transition relation \( R =
R_n^{DC} \cup R_n^{CD} \), where \( R_n^{DC} \subseteq V_n^D \times V_n^C \), \( R_n^{CD} \subseteq V_n^C \times V_n^D \), and transitions in \( R_n^{CD} \) are labeled with \( n \) or \( m \in \text{nbr}(n) \).
Each node \( v_n \in V_n \) is labeled with a subset of \( Prop \), where
\( Prop \) is the extended Fischer-Ladner closure of \( \varphi_n \) [6], [7].
The closure \( Prop \) describes the negation, subset, and fixpoint
closure of the temporal operators.

We adopt the two-pass tableau approach of [8], [4], i.e.,
first construct a tableau from the specification, then prune
and unravel the tableau into a model. The local property \( \varphi_n \) of
interest is in the format of \( \text{init-spec} \land \text{other-spec} \). Hence,
\( \text{init-spec} \) specifies a single initial state. For multiple initial
states, a set of local properties \( \{ \varphi_n^0, \varphi_n^1, \ldots \} \) is generated, each
with the same \( \text{other-spec} \) but a different \( \text{init-spec} \).

We modify the classical tableau approach to synthesize \( H_n^* \)
from \( \varphi_n \), such that \( H_n^* \) is outward-facing and closed under interference.
Subsection IV-A shows how to derive the initial tableau \( T_n^0 \) closely following the original procedure [8]. Our
main innovation is that we assume the neighbors are isomor-
phic copies of \( T_n^0 \) and subsection IV-B shows how to construct
\( T_n^{n+1} \) by adding interference transitions to \( T_n^0 \). The iterative
procedure continues until a fixpoint tableau \( T_n^* \) is reached such that
\( T_n^* \) is closed under interference by isomorphic copies of
\( T_n^* \). We then apply deletion rules (in Subsection IV-C), extract
a model \( H_n^* \) from the pruned fixpoint tableau, and obtain \( P_n \)
from \( H_n^* \) by removing interference transitions (in Subsection IV-D).
These steps follow the original tableau procedure with slight variations.

A. The Initial Tableau

Similar to the classical tableau approach [8], [4], the root of
the tableau, \( d_{root} \), is an OR-node labeled with \( \{ \varphi_n \} \). Starting
with \( d_{root} \), the initial tableau \( T_n^0 \) is constructed by repeatedly creating successors and appending them to the leaf nodes. In
the case of duplicate labels and types, the newly created node
is merged with the existing node, i.e. the new node is deleted
and its incoming and outgoing edges are added to the existing
node. The construction of \( T_n^0 \) terminates when there are no
more leaf nodes. If there are multiple initial states, we repeat
the steps of constructing the initial tableau with different init-
specs while merging duplicates.

For each OR-node \( d \), \text{blocks}(d) \) is a set of successors of \( d \)
such that each AND-node \( c_i \in \text{blocks}(d) \) represents a way of
satisfying the formulae in \( L(d) \). The generation of \text{blocks}(d)
follows the classical tableau approach, with a slightly different
\( \alpha \)-\( \beta \) expansion: as listed in Table I (c.f. [6]), most expansions
are binary, except for \( AY \) and \( EX \), which expand to a list
of operators indexed by \( n \) and the neighbors in \text{nbr}(n). The
unindexed next-time operators are not part of \( \varphi_n \) but can be
added to node labels during formula expansion. Formulae in
\( L(d) \) are satisfiable iff there exists a node in \text{blocks}(d) whose
label is satisfiable.

For each AND-node \( c \), \text{tiles}(c) \) is the minimal set of \( n \)-
successors of \( c \), i.e. the next-time states reachable through
transitions labeled with \( n \). Let \( CA_n = \{ \{ A \varphi_n f \mid f \in L(c) \} \}
and \( CE_n = \{ g \mid E \varphi_n g \in L(c) \} \). For each \( g \in CE_n \), an
OR-node labeled \( CA_n \cup \{ g \} \) is created as a successor node
of \( c \). Edges from \( c \) to nodes in \text{tiles}(c) \) are labeled with \( n \). Here,
we only consider a single edge case. I.e., if both \( CA_n \)
and \( CE_n \) are empty sets, then we add a ‘dummy’ successor
\( d_n \) to \( c \) and set \text{blocks}(d_n) \) = \{ \( c \) \}. If \( L(c) \) is satisfiable, then
the labels of all nodes in \text{tiles}(c) \) are satisfiable.

In the classical tableau approach, for each neighbor \( m \), the
set of \( m \)-successors of \( c \) are created in a similar way to \( n-
successors. However, since the local property \( \varphi_n \) only specifies the behavior of \( P_n \), interference transitions by neighboring processes \( P_m \) are not specified in \( \varphi_n \). Instead, we infer the transitions labeled with \( m \) based on transitions labeled with \( n \). The next subsection shows the detailed steps of adding interference transitions and \( m \)-successors.

**B. The Fixpoint Tableau**

Starting from the initial tableau \( T_n^0 \) containing only transitions labeled with \( n \), we construct \( T_n^{i+1} \) from \( T_n^i \) through the following steps.

First, we summarize the interferences contained in the tableau so far. We search \( T_n^i \) for \( n \)-transitions that change the values of shared variables and convert these \( n \)-transitions to a set of \( m \)-transitions for each neighbor \( m \) by bijection. That is, for each pair of AND-nodes \( c \) and \( c' \) such that \( c' \in \text{blocks}(d) \) for \( d \in t_i \), let \( Y \) and \( Y' \) be the values of the shared variables in \( L(c) \) and \( L(c') \), respectively. If \( Y' \) is different from \( Y \), \( c' \) is added as an interference transition that changes the values of shared variables between \( n \) and \( m \) from \( Y_m \) to \( Y'_m \).

Next, we add interference transitions to the current tableau. For each unique tuple \((m, Y_m, Y'_m)\), we add the interference transition to each applicable AND-node and label the transition with \( m \). An AND-node \( c \) in \( T_n^i \) is applicable to an interference transition \((m, Y_m, Y'_m)\) if the values of shared variables in \( Y_m \) match those in \( L(c) \), and the interference transition is not already added to \( c \).

- For each AND-node \( c \), \( \text{bricks}_m(c) \) is a possibly empty set of \( m \)-successors of \( c \), and \( \text{bricks}(c) = \bigcup_{m \in \text{blocks}(a)} \text{bricks}_m(c) \). An empty \( \text{bricks}_m(c) \) indicates an implicit self-loop if \( m \) in \( c \), i.e., transitions labeled with \( m \) do not interfere with \( n \) in \( c \).
- The set \( \text{bricks}_m(c) \) is generated as follows. Let \( CA_m = \{ f \mid A_pY_m f \in L(c) \} \) and \( CE_m = \{ g \mid E_pX_m g \in L(c) \} \). These \( m \)-indexed properties are not subformulæ of \( \varphi_n \) but are added to node labels as a result of \( \alpha \beta \) expansion. For example, \( A_pG_p \) expands to \( p, A_pY_pA_pG_p \) and \( A_pY_pA_pG_p \) for each \( m \). For each unique interference \((m, Y_m, Y'_m)\) and applicable AND-node \( c \), we create an OR-node successor \( d_m \). The label of \( d_m \) contains formulæ in \( Y'_m \), \( CA_m \), and values of variables in \( L(c) \) that are not shared with \( m \).
- In addition to that, we also create an OR-node successor of \( c \) for each \( E_pX_m g \in L(c) \). These successors capture the changes to shared variables as well as the satisfaction of existential next-time properties. For a given \( Y_m \), consider the set of \( Y'_m \) such that \((m, Y_m, Y'_m)\) is a tuple. Those \( Y'_m \) form the possible interference to shared variables. The changes to \( Y_m \) are translated into a disjunctive formula. Each change is represented as a conjunct of values of variables in \( Y'_m \). For each \( g \in CE_m \) and applicable AND-node \( c \), we create an OR-node \( d_m \), and \( L(d_m) \) contains \( g \), the disjunctive formulæ in \( CA_m \), and values of variables in \( L(c) \) that are not shared with \( m \). For each newly created node \( d_m \), we connect \( c \) to \( d_m \) by an edge labeled \( m \) and merge \( d_m \) if duplicated.

Figure 2 is an example of adding \( \text{bricks}_m \) to a given AND-node (\( n \)-successor nodes are omitted from the figure).

In this example, \( a \) and \( b \) are two external variables shared between \( n \) and \( m \), and \( c \) is an internal variable of \( n \). Suppose \( m \) interferes with \( n \) only by changing \((a, b) \) to \( (¬a, b) \) or \( (a, ¬b) \). The disjunctive formula representing changes of \((a, b) \) is \((¬a \land b) \lor (a \land ¬b) \). Property \( E_pG_a \) is propagated to exactly one \( m \)-successor, and \( A_pG_c \) is propagated to all \( m \)-successors.

The propagation is done through blue formulæ in the figure.

Finally, for each newly added OR-node \( d \), we create descendants of \( d \) that are reachable via \( n \)-transitions. The construction terminates when there are no more leaf nodes. The size of the resulting tableau \( T_n^{i+1} \) is greater than or equal to the size of \( T_n^i \). We repeat these steps until no more transitions or nodes can be added, i.e., when \( T_n^{i+1} = T_n^i \). The resulting tableau captures all the changes to values of shared variables by neighboring processes as interference transitions.

Based on the fairness constraint, interference transitions will eventually be executed. At each AND-node \( c \) where the value of shared variables between \( n \) and \( m \) is represented by \( Y_m \), we need to distinguish between two cases: (1) \( m \) changes \( Y_m \) to \( Y'_m \) through a (stuttering) transition such that \( Y_m \neq Y'_m \), and (2) \( m \) keeps \( Y_m \) unchanged in a fair cycle. The first case was captured as interference transitions and the second as implicit self-loops. However, if both cases happen at the same \( Y_m \), the corresponding node \( c \) should have the interference transition indicating the change as well as an explicit \( m \)-labeled self-loop indicating the choice of ‘remaining unchanged forever’. We add these self-loops to applicable AND-nodes in \( T_n^i \) by using dummy nodes.

When no more transitions can be added, the tableau has reached its fixpoint, \( T_n^* \), and the construction terminates.

**C. Tableau Pruning**

The goal is to construct a model \( H_n^* \) such that \( P_n \) is outward-facing in \( H_n^* \). Since \( H_n^* \) is extracted from the pruned \( T_n^* \), we added a restricted outward-facing assumption to only focus on tableaux where all the encoded models are outward-facing. For each neighbor \( m \) and each set of values of shared variables \( Y_m \), the restricted outward-facing assumption requires the representative \( n \) to make the same set of changes to the shared state \( Y_m \) no matter which AND-node child is selected to be in the model. This guarantees a strictly stronger form of the outward-facing property.
TABLE II
THE DELETION RULES FOR TABLEAU PRUNING.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteP</td>
<td>Delete any node whose label is propositionally inconsistent.</td>
</tr>
<tr>
<td>deleteOR</td>
<td>Delete any OR-node all of whose successors are deleted.</td>
</tr>
<tr>
<td>deleteAND</td>
<td>Delete any AND-node one of whose successors is deleted.</td>
</tr>
<tr>
<td>deleteEU</td>
<td>Delete any node ( v ) if ( E_\Phi(fUg) \in L(v) ), and there does not exist an AND node ( c' ) reachable from ( v ) through a finite path ( \pi ).</td>
</tr>
<tr>
<td>deleteAU</td>
<td>Delete any node ( v ) if ( A_\Phi(fUg) \in L(v) ), and there does not exist a subdag ( U ) rooted at ( v ).</td>
</tr>
<tr>
<td>deleteEG</td>
<td>Delete any node ( v ) if ( E_\Phi Gg \in L(v) ), and there does not exist a fair full path ( \pi ) starting at ( v ) such that ( g \in L(c') ) for all leaf nodes ( c' ) in ( U ) and ( f \in L(c) ) for all internal AND-nodes ( c ) of ( U ).</td>
</tr>
<tr>
<td>deleteJoint</td>
<td>Delete any AND-node ( c_n ) if every AND-node ( c_m ) of a neighbor ( m ) that forms a joint state ((c_n, c_m)) is deleted.</td>
</tr>
</tbody>
</table>

Before pruning, we verify the assumption on \( T_n^* \) and terminate the synthesis procedure if the assumption is violated. (Relaxing the assumption and finding an outward-facing model from any tableau is a future research direction.)

Similar to the classical approach, pruning tableau \( T_n^* \) is done by deleting inconsistent nodes. As shown in Table II (c.f. [4]), an additional rule \textit{deleteJoint} is added. \textit{DeleteJoint} deletes any AND-node in \( T_n^* \) that fails to form joint states with neighboring isomorphic tableau, \( T_m^* \). For each neighbor \( m \) and each value of shared variables \( Y_m \), let \( C_m^Y \) be a set of AND-nodes of \( n \) such that \( Y_m \subseteq L(c_n) \) for each node \( c_n \) in the set. Let \( C_m^Y \) be a set of AND-nodes of \( m \) such that each \( c_m \in C_m^Y \) forms joint states with the nodes in \( C_m^Y \). If all the isomorphic AND-nodes \( b_n \) of each \( c_n \) in \( C_n^Y \) are deleted, we delete all the AND-nodes \( c_n \) in \( C_m^Y \) because the joint states of \( Y_m \) no longer hold, and vice versa.

The pruning process eventually terminates because the number of nodes in \( T_n^* \) is finite. Upon termination, if the root of the tableau is deleted, then \( \varphi_n \) is not satisfiable by our procedure. Otherwise, we extract a model \( H_n^* \) from the pruned tableau.

D. Extraction of a Model

We reuse the existing procedure in [8], [6] to ‘unravel’ the pruned tableau \( T_n^* \) into a model.

For each AND-node \( c \) in \( T_n^* \), we construct a fragment of \( c \) following the standard tableau approach. The structure of a fragment is taken from \( T_n^* \). All nodes in a fragment are AND-nodes. Nodes \( s \) and \( t \) are connected with a directed edge in a fragment if there exists transitions \((c, d), (d, c') \in R \) in \( T_n^* \), such that \( s \) and \( t \) are copies of \( c \) and \( c' \), respectively. The fragment of \( c \) certifies the fulfillment of all eventualities in \( L(c) \). When it comes to universal eventualities like \( A_\Phi(fUg) \), if there are multiple subdags in the tableau, we choose the one with the least number of unfair cycles.

A model \( H_n^* \) is formed by connecting fragments together following the standard tableau approach. The process \( P_n \) is obtained from \( H_n^* \) by removing the interference transitions.

E. Soundness and Complexity

Theorem IV.1. Soundness. If a labeled transition system \( H_n^* \) is constructed from \( \varphi_n \), then \( H_n^* \) satisfies \( \varphi_n \). \( H_n^* \) is closed under neighboring interference, and process \( P_n \) is outward-facing in \( H_n^* \).

Proof. During tableau construction, \( \text{blocks}(d) \) computes successors of an OR-node \( d \), \( \text{tiles}(c) \) computes \( m \)-successors of an AND-node \( c \), and \( \text{bricks}(c) \) computes \( m \)-successors of \( c \) for neighbors \( m \). Based on the constructions of the tableau, all formulae in node labels are propagated correctly in the tableau of \( n \), including \( T_0^* \), any intermediate \( T_n^* \), and \( T_n^* \) (similar to the proofs in [6]). For example, \( A_\Phi(fUg) \) in the label of a node propagates to successor nodes as either \( g \) or \( f \), \( A_\Phi(Y_nA_\Phi(fUg)) \), and \( A_\Phi Y_nA_\Phi(fUg) \). The propagation continues forever along each path until \( g \) is reached.

Since all the nodes in the pruned \( T_n^* \) are consistent, all the eventualities in the label of any AND-node in the pruned \( T_n^* \) are fulfilled in a fragment rooted at the node. Since \( H_n^* \) is constructed by concatenating fragments, starting with a root that automatically satisfies \( \varphi_n \), \( H_n^* \) is a model of \( \varphi_n \).

The size of the tableau increases monotonically until it reaches a fixpoint, \( T_n^* \). Since the size of \( T_n^* \) is bounded, the tableau construction eventually terminates at the fixpoint. By construction, each intermediate tableau \( T_n^* \) fully reflects the interference of neighboring isomorphic copies of \( T_n^{i-1} \). The construction continues until no more nodes can be added to the tableau. Therefore, \( T_n^* \) is closed under self-interference.

Then, we show that the model is also closed under self-interference. Based on deleteJoint, in the pruned tableau \( T_n^* \), for each \( m \)-labeled transition \( Y_m \rightarrow Y_m' \) and each AND-node \( c \) whose label contains \( Y_m \), \( c \) forms joint states with neighbors \( m \), and there exists transitions isomorphic to \( Y_m \rightarrow Y_m' \) in the pruned \( T_n^* \). On the other hand, in the pruned \( T_n^* \), the set of interference transitions reflects exactly the set of transitions labeled with \( n \) that change the values of shared variables. Based on model extraction, \( H_n^* \) is closed.

Since \( T_n^* \) satisfies the restricted outward-facing tableau assumption, for all the encoded models \( H_n^* \), process \( P_n \) is outward-facing in \( H_n^* \).

Lemma IV.2. Let \( \varphi_n \) be a local property of \( n \), and \( \Sigma_n^{\text{share}} \) the set of shared variables in \( \Sigma_n \). The size of tableau \( T_n \) is bounded by \( \exp(|\varphi_n| + \exp(|\Sigma_n^{\text{share}}|)) \).

Proof. For each \( n \)-successor \( v_n \) in \( T_n \), \( L(v_n) \subseteq \text{Prop} \), so the number of formulae in \( L(v_n) \) is less than or equal to \(|\text{Prop}| \). Since duplicate nodes are merged, the number of \( n \)-successors in \( T_n \) is bounded by \( \exp(|\text{Prop}|) \).

As in Section IV-B, an extra disjunctive formula is added to the labels of some OR-nodes to represent the interference transitions of neighbors \( m \). Considering binary variables, the number of different values of shared variables is \( \exp(|\Sigma_n^{\text{share}}|) \). Hence, there are at most \( \exp\left(\exp\left(|\Sigma_n^{\text{share}}|\right)\right) \) different ways related to the presence of a disjunctive formula in node labels. Therefore, the number of nodes in \( T_n \) is bounded by \( \exp(|\text{Prop}|) + \exp\left(\exp\left(|\Sigma_n^{\text{share}}|\right)\right) \). Since \(|\text{Prop}| \)
is linear in terms of $|\varphi_n|$, the number of nodes in $T_n$ is in $O(\exp(|\varphi_n| + \exp(|\Sigma_n^{\text{share}}|)))$. In applications, $\exp(|\text{Prop}|)$ is more likely to dominate $\exp(\exp(|\Sigma_n^{\text{share}}|))$. In most cases, the size of $T_n$ is exponential in the length of the input local Fair CTL property $\varphi_n$.

**Lemma IV.3.** The cost of constructing $P_n$ is in time polynomial in the size of the tableau.

**Proof.** For each node $v$ in tableau $T_n$, the sum of the lengths of the formulae in $L(v)$ is in $O(|\varphi_n|^2)$. The cost of computing successor for $v$ is polynomial in $|\varphi_n|$. Fixpoint construction, tableau pruning, and unraveling all require time polynomial in the size of the tableau. Therefore, the total cost of constructing $P_n$ is in time $O(\exp(|\varphi_n| + \exp(|\Sigma_n^{\text{share}}|)))$.

The tableau approach constructs $P_n$ as a template for the locally symmetric processes. To deploy the template throughout the process network, the subscript indices on all state and transition labels are changed accordingly.

**V. APPLICATIONS**

We illustrate our approach with three ring-based protocols, namely, mutual exclusion, leader election, and dining philosophers. Our approach is implemented in Python with the CTL module provided in the pyModelChecking API. We tested the synthesis procedure on a 2.5 GHz CPU and 16 GB of memory, and each ran for 5.3, 297, and 261 seconds, respectively. In each case, the procedure converged within three tableau iterations.

**A. Mutual Exclusion**

Mutual exclusion is a mechanism that prevents processes from accessing a shared resource simultaneously. Globally, the mutual exclusion property asserts that no two processes can be in the critical section at the same time. Locally, the property is achieved through token passing.

For any $K \geq 2$ and a generic $n$, the external variables $tok_n$ and $tok_{n+1}$ are shared with $n-1$ and $n+1$, respectively. The internal variable $N_n$ stands for non-critical, $T_n$ for trying, and $C_n$ for critical. We specify $\varphi_n$ as follows.

- Three initial conditions: $N_n \land tok_n \land \neg tok_{n+1}$ ($n$ has the token), $N_n \land \neg tok_n \land tok_{n+1}$ (the right neighbor has the token), and $N_n \land \neg tok_n \land \neg tok_{n+1}$ (no token locally).
- Local mutual exclusion: $A \varphi(G(\neg tok_n \lor \neg tok_{n+1}))$.
- Moves of $n$ from non-critical to trying (while keeping the token) or remains in non-critical (while passing the token): $A \varphi(G((N_n \land \neg tok_n) \Rightarrow (E \varphi X_n (N_n \land \neg tok_{n+1} \land E \varphi X_{n+1} (T_{n+1} \land \neg tok_{n+1}))))$.
- Moves of $n$ from trying to critical with the token: $A \varphi(G((T_n \land tok_n) \Rightarrow A \varphi Y_n (C_n \land tok_{n+1})))$.
- Moves of $n$ from critical to non-critical while passing the token, $A \varphi(G(C_n \Rightarrow A \varphi Y_n (N_n \land \neg tok_n \land tok_{n+1})))$.
- The liveness property: $A \varphi(G(T_n \Rightarrow A \varphi FC_n))$

Properties that ensure variables remain unchanged are omitted from the list for the sake of clarity. By induction on the size $K$, assuming the initial condition that exactly one process owns a single token, if $\varphi_n$ is true for all processes in a ring, then it guarantees that each process eventually gets and passes the token, and there is exactly one token (i.e., tokens are not generated or lost). Hence, no two processes access the critical resource simultaneously.

Fig. 3 is a model of $\varphi_n$. Rectangles represent local states, where yellow corresponds to initial states. Solid arrows are transitions by $P_n$, and dashed arrows are interference transitions. Rectangles with red borders are inconsistent states because $\varphi_n$ has no information about the initial conditions of non-neighboring processes. I.e., in the perspective of $n$, there is at most one token locally, but globally, $n$ does not know. Instead of deleting the parents of these inconsistent states according to the deletion rules, we manually refine the set of interference transitions by taking into account the initialization of all processes in the ring. I.e., there is only one token.

**B. Chang and Roberts Leader Election**

Suppose each process has a finite and unique competing value (abbr. cv). The goal of the protocol is to select the process with the largest $cv$ to be the leader. Globally, the correctness of the protocol is specified as a safety property, i.e.,
there is never more than one leader, and a liveness property, i.e., eventually there will be a leader. The specification can be written locally from the perspective of a generic \( n \) [9]. The \( cv \) of \( n \) may or may not be the greatest on the network.

Initially, some but not all processes detect the absence of the leader, i.e., \( P_n \) may become a participant in the election and send out an election message containing its \( cv \) to the right. When \( P_n \) receives an election message from its left, \( P_n \) compares the competing value in the message, denoted by \( cv' \), with its own \( cv \). In general, the comparison yields three different outcomes, i.e., \( cv' > cv \), \( cv' < cv \), and \( cv' = cv \). If \( cv' > cv \), \( P_n \) forwards the message to the right. If \( cv' < cv \), \( P_n \) sends a message of its own \( cv \). If \( cv' = cv \), \( P_n \) becomes the leader. A non-participant becomes a participant after forwarding or sending an election message, and a participant no longer sends election messages of its own \( cv \). A new leader sends a message to the right to terminate the election. Upon receiving the termination message, a process becomes non-participant and forwards the message.

A constructed model \( H_n^* \) is shown in Fig. 4. External variables \( b_n \) and \( b_{n+1} \) are shared with left and right neighbors, representing shared message buffers of size one. Internal variables \( par_n \) denotes that \( P_n \) is a participant, \( l_n \) denotes that \( P_n \) is the leader. Comparisons are abstracted into boolean variables. When \( \text{comp}_n \) is true indicating a comparison in progress, one of the following is true, \( f_n \) (greater/forward), \( s_n \) (smaller/send), \( d_n \) (smaller/discard), \( e_n \) (equal), and \( t_n \) (election termination). For comparison results other than \( d_n \), \( b_{n+1} \) becomes true, i.e., a message is sent to the right.

The global reasoning for this protocol is as follows. Globally, there exists one process whose competing value is the greatest. Based on the global initialization and local specification, and supposing the message comparison always yields correct results, the process with the greatest \( cv \) sends and receives a message with its own competing value. For all the other processes, messages with their competing values will not go through the full round of message passing, and these messages will eventually be discarded by processes with a greater competing value. Therefore, there will eventually be a leader and never more than one leader.

C. Dining Philosophers

In a standard dining philosopher protocol [10], the internal state of \( P_n \) is one of \( T_n \) (thinking), \( H_n \) (hungry), or \( E_n \) (eating). Fig. 1 indicates the external variables of \( n \), where \( r_n \) means that \( P_n \) picks up its left fork, and \( l_n \) means that the left neighbor picks up the fork. Similarly, \( l_{n+1} \) means that \( P_n \) picks up its right fork, and \( r_{n+1} \) means that the right neighbor picks up the fork. The variables \( r_n \) and \( l_n \) cannot be true at the same time, nor can \( r_{n+1} \) and \( l_{n+1} \). Both variables \( r \) and \( l \) are false means the corresponding fork is available. Process \( P_n \) can read and write \( r_n \) and \( l_{n+1} \), but \( P_n \) has read-only access to \( l_n \) and \( r_{n+1} \).

Process \( P_n \) can stay in thinking or move to hungry at any time, and \( P_n \) in its hungry state picks up available forks. While holding both the left and the right forks (i.e., \( r_n \land l_{n+1} \)), \( P_n \) should enter into the eating state. After eating, \( P_n \) goes back to thinking and returns the forks (i.e., \( \neg r_n \land \neg l_{n+1} \)). The specification guarantees that no two neighboring processes are eating simultaneously.

Fig. 5 shows a model of \( \varphi_n \). Adding a liveness property, \( A_{\Phi} G(H_n \Rightarrow A_{\Phi} F E_n) \) would make \( \varphi_n \) unsatisfiable. Livelock and starvation are possible and are observed locally in the model. The unsatisfiability of appending the liveness property to \( \varphi_n \) does not mean there is no local solution to the dining philosopher problem. On the contrary, the problem can be solved using acyclic precedence graphs as in [10] (i.e., by modifying \( \varphi_n \) and introducing more variables).

VI. RELATED WORK AND CONCLUSION

In this paper, we reduce the synthesis problem for a parameterized protocol to the problem of synthesizing a representative process that meets a local specification under interference from neighboring copies of itself. The algorithm runs in time exponential in the length of the local property, which is expressed in Fair CTL and may include safety as well as liveness aspects, using both universal and existential path quantification. The approach is incomplete and not fully automated, but it succeeds on several interesting cases.

The novelty is in our solution to the new ‘self-referential’ synthesis question. Our tableau construction builds on the classical one of [8] for CTL and that of [4] for Fair CTL. These constructions work in closed synthesis settings where the environment is assumed to be cooperative. A fully open synthesis procedure was devised for LTL in [11]. In our case, the environment is formed of copies of the unknown to-be-synthesized process, which is an open synthesis problem of a special type.

The work relies on the compositional inductive invariant under local symmetry given in [12], [13], and [2]. We capture the behaviors of a representative in its neighborhood as a fixpoint tableau. Other work related to inductive invariants (c.f. [14]) uses similar fixpoint characterizations to compute thread-modal reify-guarantee assertions under abstractions.
Synthesis of a distributed system is undecidable, even with a fixed number of components [15]. Decidable architectures are known [16] as are decision procedures (c.f. [17], [18]), but the complexity is exponential or even nonelementary in the number of processes. In contrast, our procedure produces a representable process that is replicated to form arbitrary-size instances, so its complexity is independent of the instance size.

Reduction or generalization theorems are also central to prior work on parametrized synthesis. In [5] representative processes are constructed from synthesis of pair-systems. The paper [19] decides ‘almost always satisfiability’ for indexed but restricted CTL properties. Cutoff results for parametrized verification are applied in [20] to synthesize ring protocols; however, the dining philosophers and leader election examples fall outside the class for which cutoffs are known. The paper [21] takes an automata-theoretic approach to rotation-symmetric architectures. Synthesis of symmetric processes in self-stabilizing parameterized unidirectional rings is explored by [22]. The paper [23] focuses on round-bounded parameterized systems.

The different approaches that exploit symmetry in the system structures make use of a kind of global symmetry c.f. [24], [25], and [26]. In contrast, the work presented in this paper relies on notions of local symmetry as introduced in [12], [13], and [2]. The differences are important because local symmetry properly generalizes ‘global symmetry,’ often allowing for exponentially more reduction, for instance in the case of ring architectures. Our work here is the first to show how the notation of local symmetry can be used to form the basis of a synthesis procedure whose output is a single representative that can be deployed across all network instances in the parametric family of networks.

The reduction theorem on which the work in this paper is based is of an assume-guarantee type. Existing formulations of assume-guarantee synthesis (c.f. [27], [28]) however do not allow for the self-referential form of synthesis that is required by the reduction theorem.

We are currently working on applications to other protocols, including those with several representative processes, to fault tolerant protocols [6], and towards relaxing the outward-facing assumption.

Acknowledgments. Kedar Namjoshi was supported in part by DARPA under contract HR001120C0159. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Richard Trefler and Ruoxi Zhang were supported, in part, by an Individual Discovery Grant from the Department of Defense or the U.S. Government. Richard Trefler and Ruoxi Zhang interpreted as representing the official views or policies of the Department.

The views expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES
