# Synthesis of Parametric Locally Symmetric Protocols from Abstract Temporal Specifications*

Ruoxi Zhang[1(✉)], Richard Trefler[2], and Kedar S. Namjoshi[3]

[1] University of Waterloo, Waterloo, ON, Canada
r378zhan@uwaterloo.ca
[2] University of Waterloo, Waterloo, ON, Canada
trefler@uwaterloo.ca
[3] Nokia Bell Labs, Murray Hill, NJ, USA
kedar.namjoshi@nokia-bell-labs.com

**Abstract.** Scalable distributed systems are typically parametric in design. The key parameter is the number of isomorphic components, $K$. A second important parameter is the number of neighbors, $k$, of each component process. In this work, we describe a methodology that uses an automated synthesis procedure to construct parametric system instances where both $K$ and $k$ can vary arbitrarily, extending prior work on synthesis for a fixed $k$. The methodology relies crucially on locality, symmetry, and abstraction. The first step is to eliminate $K$ by refining a general, system-wide specification to a local temporal specification for a generic process in its parameterized neighborhood. Next, the local process specification is abstracted to remove its dependence on $k$. These steps are done by hand. The given synthesis procedure then automatically constructs an abstract process from the abstract local specification with a worst-case cost exponential in the length of the abstract local specification. We show that, for any $k$, the concretized abstract process meets the local specification. We then show that instantiating the abstract process with different $k$ and $K$ forms system instances that satisfy the system-level specification. The worst-case cost of instantiation is linear in $K$. We use this method to synthesize an atomic snapshots protocol on fully connected networks and a dining philosophers protocol on hypercubes.

## 1 Introduction

Scalable distributed systems, such as network protocols, web services, and multicore processors, are typically parameterized. In this work, we focus on parametric systems composed of $K$ isomorphic processes communicating with neighboring

* Richard Trefler and Ruoxi Zhang were supported, in part, by an Individual Discovery Grant from the Natural Sciences and Engineering Research Council of Canada. 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.

processes according to an underlying network topology. We further assume that each component process in a parametric system has $k$ neighbors. We suppose that $K$ and $k$ can both take on arbitrarily large values.

Fully automated verification and synthesis of parametric systems are undecidable [3, 26]. We present a semi-automated method for parametric synthesis:

1. Reduce a global specification to a local one that describes the requirements of a single, generic process $P$. See [2, 9, 21] for examples of such reductions.
2. Abstract the unbounded neighborhoods of $P$ and rewrite the local specification under this abstraction.
3. Automatically synthesize from the abstract local specification a model for $P$ that satisfies this specification under interference transitions from isomorphic neighbors. Automatically extract an abstract $P$ from the model.

Our proposed method guarantees that the synthesized $P$ meets the global specification for arbitrarily large $k$ and $K$. The abstract $P$ forms a template that can be replicated on nodes of arbitrary networks.

Prior work [29] shows how to synthesize parametric systems with fixed neighborhoods (e.g., rings, where each node has two neighbors). This work extends [29] to parametric systems with neighborhoods of unbounded, varying sizes, i.e., it adds another layer of parameterization. A key difficulty is that, in general, abstraction does not preserve realizability. An abstract specification may have a model while the concrete specification does not. Hence, the abstraction process must carry side-information that makes concretization possible. We detail the method in this paper.

An example abstraction is that of counting the number of neighbors for which certain variables are true. We assume that the $k$ neighbors of $P$ can be divided into a fixed number of non-empty partitions according to how $P$ interacts with its neighbors. We define a bounded number of Boolean predicates to approximate the *counts* for each partition and manually rewrite the parametric local specification using these predicates to remove the dependence on $k$.

For example, consider a parametric property of the generic process $P$ that describes 'for each neighbor, if unread, then all next steps of $P$ read the port shared with the neighbor.' In addition, assume that a port once read, stays read. We rewrite the parametric property as an abstract property stating that 'if not all neighbors are read, then in all next steps of $P$, all neighbors of $P$ are read.'

In this example, the abstract property is equivalent to the original. We also handle the case where it is impossible to rewrite a parametric property into an equivalent abstract property using a given set of abstract variables. Hence, we allow the abstract property to over-approximate the parametric property. The precondition in the abstract property describes a set of concrete states. We define *context* as auxiliary information to describe a subset of the over-approximated state set so that the update in the parametric property only applies to this subset. Although the process indices of neighbors are omitted in the abstract property, the context connects the current and next states that $P$ shares with each neighbor during concretization.

We define a restricted parametric specification format, e.g., parameterized eventualities are not allowed. We then present a synthesizer that takes as input an abstract local specification and produces a model that can be concretized for any given $k$ and $K$ (c.f. [13, 14, 29]). Our synthesizer has time complexity exponential in the size of the abstract local specification. However, once $P$ is synthesized, instances can be constructed in time linear in $K$ by replicating $P$.

The synthesizer has been implemented and used to automatically synthesize parametric protocols. We detail two protocols of interest: an atomic snapshots protocol [1] on fully connected networks and a dining philosophers protocol [8] on hypercubes. Automatically constructing abstract representative processes for both protocols takes only a few minutes.

## 2 Preliminaries

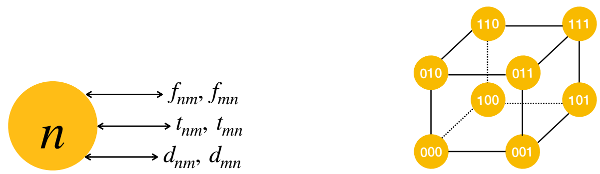
***Networks and Processes.*** A *network* $(N, E)$ is a directed graph, where $N$ is a set of nodes, and $E$ is a set of edges. The size of the network, denoted $K$, is the number of nodes in $N$. Each *node* $n \in N$ is connected to a set of *edges* $E_n \subseteq E$. The connection is towards $n$, away from $n$, or both. Two nodes with a common connected edge are *neighbors*. The set of neighbors of $n$ is denoted by $nbr(n)$, and $k \coloneqq |nbr(n)|$ denotes the size of the neighborhood.

Processes are deployed on nodes, e.g., $P_n$ denotes the process deployed on $n$. Edges are assigned *external* variables, where incoming and outgoing connections of a node represent the *read* and *write* permissions on the corresponding variables, respectively. An external variable assigned to a common connected edge is called a *shared* variable between the pair of neighboring processes.

***Parametric Tiles.*** This work focuses on *uniform* networks, where all nodes are associated with the same parametric tile process $P_n \coloneqq \{P_n(k)\}$. Note that $P_n$ is a family of concrete tile processes, and in any system instance, a node with $k$ neighbors is assigned a replica of $P_n(k)$. The *parametric tile* describes the variables shared between $n$ and each $m \in nbr(n)$, where $n$ refers to the representative of all nodes in all networks induced by the parametric tile.

For example, Fig. 1a shows the parametric tile for the dining philosophers protocol that works for scalable $k$ and $K$. Variable $f_{nm}$ (resp. $f_{mn}$) represents that the fork between $n$ and $m$ is owned by $P_n$ (resp. $P_m$) . Note that $f_{nm}$, from the perspective of $n$, and $f_{mn}$, from the perspective of $m$, are isomorphic, and $f_{nm} \equiv \neg f_{mn}$ because the fork is either owned by $P_n$ or $P_m$. Similarly, possession of the request token for the fork shared by $n$ and $m$ is denoted by $t_{nm}$ and $t_{mn}$. Variables $d_{nm}$ and $d_{mn}$, with equal values, represent whether the shared fork is dirty or clean. As shown by the bidirectional connections between $n$ and the edges, $P_n$ has read/write access to all these variables.

Protocol instances can be constructed by deploying the tile process on underlying networks. For example, a concrete network with $K = 8$, $k = 3$ is formed by replicating the tile in Fig. 1a based on the 3-dimensional cube in Fig. 1b.




(a) A parametric tile. (b) A hypercube.

**Fig. 1.** Obtaining instances by copying a tile process according to underlying networks.

***Concrete Processes and Instances.*** Semantically, an *instance* $P_0||...||P_{K-1}$ is a concurrent program consisting of $K$ sequential processes running in a non-deterministic interleaving manner. The behavior of an instance is defined as a *global state transition system*, denoted $G$. Each component process $P_{i\in[0,K)}$ in an instance interacts with $k$ neighbors through shared memory.

If the neighborhood pattern defined by the tile is fixed, then a single representative process $P_n^c$ suffices for all instances where $K$ varies. We use the superscript $c$ (for 'concrete') to indicate the case where the neighborhood is fixed.

The state machine for $P_n^c$ is a tuple $(S_n, S_n^0, T_n, \lambda_n)$, where $S_n$ is the set of local states, $S_n^0$ is the set of initial states, $T_n \subseteq S_n \times S_n$ is the transition relation, and $\lambda_n : S_n \rightarrow 2^{\Sigma_n^c}$ is the labeling function that labels each state $s \in S_n$ with variables true in $s$. All transitions in $T_n$ are labeled with $n$ and are therefore called $n$*-transitions*. Here, $\Sigma_n^c$ denotes the set of atomic propositions (i.e., internal and external variables) of $n$. The *internal states* (resp. *shared states*) are the projection of state labels to internal (resp. shared) valuations.

***Interference Transitions.*** For each $m \in nbr(n)$, a *joint state* $(s,t)$ is a pair of local states $s \in S_n$ and $t \in S_m$ such that $s$ and $t$ have the same value on each variable shared between $n$ and $m$. A *joint transition* is a transition from $(s,t)$ to the joint state $(s',t')$, labeled with $n$ if $(s,s') \in T_n$ or $m$ if $(t,t') \in T_m$.

A joint transition caused by a neighbor interferes with the local states of a process by changing the common shared state. The transition from local state $s$ to $s'$, caused by $P_m^c$ through a joint transition labeled $m$, is called an *interference transition* in the state space of the interfered process, $P_n^c$. The internal state of the interfered process remains unchanged by the interference transition.

The *local state transition system* of $n$, denoted $H_n^\theta$, describes the local state space of $P_n^c$ with interference transitions. Here, $\theta = \forall i \in [0,K) : \theta_i^c$ denotes a compositional invariant of any instance, where each local invariance assertion $\theta_i^c$ defines a set of local states for $P_i^c$. A transition $(s,s')$ in $H_n^\theta$ is either an $n$-transition by $P_n^c$ where $\theta_n^c(s)$ holds or an interference transition caused by $P_m^c$ via a joint transition from $(s,t)$ to $(s',t')$ where $\theta_n^c(s)$ and $\theta_m^c(t)$ hold.

***Global-Local Reduction.*** Theorem 1 shows the relationship between $H_n^\theta$ and any instance $G$ built from copies of $P_n^c$ extracted from $H_n^\theta$. See [24] for proof. Here, $G_n$ is obtained by replacing the transition labels in $G$ except $n$ with $\tau$.

**Theorem 1.** *([24]) Let the scheduling of processes be unconditionally fair. With outward-facing interaction, $H_n^\theta$ and $G_n$ are stuttering-bisimular.*

Informally, $P_n^c$ is outward-facing if its interference with each neighbor $P_m^c$ depends solely on the shared state between $n$ and $m$. Formally, as shown in [24], two local states of $P_n^c$ are related by a relation $B_{mn}$ on $H_n^\theta$ if every common shared variable has the same value. If for each $m \in nbr(n)$, $B_{mn}$ is a stuttering bisimulation, then $P_n^c$ is *outward-facing* in the interactions with its neighbors.

We follow the standard definition of stuttering bisimulation [6]. A *stuttering simulation* from $M_1 := (S_1, S_1^0, T_1, \lambda_1)$ to $M_2 := (S_2, S_2^0, T_2, \lambda_2)$ is a binary relation $B$ from $S_1$ to $S_2$ satisfying the following conditions. ($B$ is a *stuttering bisimulation* if $B^{-1}$ is also a stuttering simulation.)

1. $(s_1^0, s_2^0) \in B$ for each pair of initial states $s_1^0 \in S_1^0$ and $s_2^0 \in S_2^0$;
2. If $(s_1, s_2) \in B$ for states $s_1 \in S_1$ and $s_2 \in S_2$, then the common variables in labels $\lambda_1(s_1)$ and $\lambda_2(s_2)$ have the same value;
3. If $(s_1, s_2) \in B$, then for each state $t_1$ reachable from $s_1$ through a finite path $\pi_1$ with labels $\tau^*; a$, there exists $t_2$ reachable from $s_2$ through $\pi_2$ with labels $\tau^*; a$ such that $(t_1, t_2) \in B$ and $(u_1, v_2) \in B$ for every pair of intermediate states $u_1$ on $\pi_1$ and $v_2$ on $\pi_2$. Here, $a$ is a transition label, and $\tau \neq a$.

The work in [29] introduced a decision procedure for concrete local specifications in Fair CTL. The input specification, denoted $\varphi_n^c$, describes the behavior of $P_n^c$ over a fixed neighborhood. The synthesizer constructs a model for $\varphi_n^c$, denoted $H_n^c$, as shown in Theorem 2. The representative $P_n^c$ is derived from $H_n^c$ by eliminating interference transitions in $H_n^c$.

**Theorem 2.** *([29]) A labeled system $H_n^c$ synthesized from a concrete local specification $\varphi_n^c$ using the decision procedure in [29] satisfies $\varphi_n^c$, is closed under neighboring interference, and unravels into an outward-facing $P_n^c$.*

***Partition the Neighbors.*** The neighbors of $P_n$ are partitioned according to how $P_n$ interacts with them. Let $P_m$ and $P_r$ be two neighbors of $P_n$. We define the partitioning of neighbors using a bijection on the local state space of $P_n$ for any $k$. The bijection, denoted $\delta$, swaps the shared state between $P_n$ and $P_m$ with the shared state between $P_n$ and $P_r$. We say $(P_m, \delta, P_r)$ holds if, for each joint transition $((s_n, s_m), (t_n, t_m)) \in T_n$, there exists another joint transition in $T_n$ from $\delta(s_n, s_m)$ to $\delta(t_n, t_m)$. A *partition*, denoted $nbr(n)_i$, is defined as the maximal set of neighbors such that $(P_m, \delta, P_r)$ and $(P_r, \delta^{-1}, P_m)$ for each pair of neighbors $P_m, P_r \in nbr(n)_i$.

Unless otherwise stated, we assume that all neighbors are in a single, monolithic partition, which is the case for both example protocols in this paper. We can also handle the case of multiple neighbor partitions, i.e., $\cup_i nbr(n)_i = nbr(n)$, and these partitions are mutually disjoint and non-empty. The token-ring protocol is an example of having two partitions, where tokens only move from left to right. We extend the definition of parametric tiles accordingly. The parametric tiles we are interested in have a fixed number of neighbor partitions.

***Fair CTL.*** We write specifications as *Fair CTL* [14] formulas. Fair CTL requires that any path quantifier, $\mathsf{A}$ (for all paths) or $\mathsf{E}$ (there exists a path), be immediately followed by a linear-time temporal operator, $\mathsf{X}_n$ (indexed strong next-time), $\mathsf{Y}_n$ (indexed weak next-time), $\mathsf{G}$ (always), $\mathsf{F}$ (sometime), $\mathsf{U}$ (until), or $\mathsf{W}$ (weak until) to form a Fair CTL *modality*.

The grammar of Fair CTL is as follows. If $p \in \Sigma_n$, then $p$ is a Fair CTL formula. If $f$ and $g$ are Fair CTL formulas, then so are $\neg f$, $f \vee g$, $\mathsf{AY}_n f$, $\mathsf{EX}_n f$, $\mathsf{A} f \mathsf{U} g$, and $\mathsf{E} f \mathsf{U} g$. The syntax is extended with abbreviations $f \wedge g \equiv \neg(\neg f \vee \neg g)$, $f \Rightarrow g \equiv \neg f \vee g$, $\mathsf{A}(f \mathsf{W} g) \equiv \neg \mathsf{E}(\neg f \mathsf{U} \neg g)$, $\mathsf{E}(f \mathsf{W} g) \equiv \neg \mathsf{A}(\neg f \mathsf{U} \neg g)$, $\mathsf{AF} f \equiv \mathsf{A}(\top \mathsf{U} f)$, $\mathsf{EF} f \equiv \mathsf{E}(\top \mathsf{U} f)$, $\mathsf{AG} f \equiv \mathsf{A}(\bot \mathsf{W} f)$, and $\mathsf{EG} f \equiv \mathsf{E}(\bot \mathsf{W} f)$. A formula is called an *eventuality* if its modality is one of $\mathsf{AU}$, $\mathsf{EU}$, $\mathsf{AF}$, $\mathsf{EF}$, or $\mathsf{EG}$.

The local specification $\varphi_n$ is interpreted over a system $M \coloneqq (S, S^0, T, \lambda)$. Let $M, s \models f$ denote that $f$ holds at state $s$ in $M$ under a fairness condition. A formula $f$ is *satisfiable* iff there exists an $M$ such that $M, s \models f$ for some state $s$ in $M$. The semantics of Fair CTL are defined in the usual way (see Appendix B).

We assume that the network scheduling of processes is *unconditionally fair*, i.e., each process is selected to run infinitely often. Locally, fairness is denoted by $\Phi \coloneqq \wedge_{m \in nbr(n)} \mathsf{GF} ex_m \wedge \mathsf{GF} ex_n$, where $M, \pi \models \mathsf{GF} g$ iff for every $i \geq 0$, there exists $j \geq i$, such that $M, \pi^j \models g$. A *path* $\pi$ is a sequence of states $(s_0, s_1, ...)$ such that $(s_i, s_{i+1}) \in T$ for all $i$, and its *suffix* $\pi^j = (\pi_j, \pi_{j+1}, ...)$. A fair *full path* is an infinite path that satisfies $\Phi$.

## 3 Our Approach

We summarize the semi-automated approach into four steps and illustrate these steps using a property taken from the dining philosophers protocol.

***Step 1 (cf. [29]).*** Given a *global correctness specification*, $\varphi = \wedge_{i \in [0,K)} \varphi_i$, describing the behavior of any instance $||_{i \in [0,K)} P_i$, the purpose of step 1 is to obtain a specification independent of $K$. To achieve this, we manually reduce $\varphi$ to a *local correctness specification* $\varphi_n$ describing the behavior of a single representative process $P_n$ in its parametric neighborhood. Note that $P_n$ can be viewed as a function that outputs a concrete representative process $P_n^c$ for each $k$ in the range described by the protocol. An example of a local property is:

$$eg_1 \coloneqq \wedge_{m \in nbr(n)} \mathsf{AG}((\underbrace{f_{nm} \wedge d_{nm} \wedge t_{nm}}_{\alpha_{nm}}) \Rightarrow \mathsf{AY}_n((\underbrace{\neg f_{nm} \wedge \neg d_{nm} \wedge t_{nm}}_{\beta_{nm}})).$$

Property $eg_1$ represents that 'for all neighbors $P_m$ of $P_n$, if $P_n$ holds the dirty fork and the request token shared with $P_m$, then in any next step of $n$, $P_n$ cleans and sends the fork to $P_m$.' In this example, $\alpha_{nm}$ and $\beta_{nm}$ denote the parametric precondition and postcondition of $eg_1$, respectively. We also assume that for any $m$, if $\alpha_{nm}$ is false, each variable not explicitly updated remains unchanged in all the next steps of $P_n$ described by $eg_1$.

We use $eg_1$ as a running example. Given any abstract precondition $\alpha_n^a$ describing that $\alpha_{nm}$ holds for some $m$, we rewrite $eg_1$ as an abstract property $\mathsf{AG}(\alpha_n^a \Rightarrow (\mathsf{AY}_n\beta_n^a \wedge \mathsf{EX}_n\gamma_n^a))$, where $\beta_n^a$ and $\gamma_n^a$ are abstract postconditions.

***Step 2, in Section 4.*** We consider $\varphi_n$ as a Fair CTL formula of the form $\wedge_j f_j$, which can be split into a finite list of subformulas using $\wedge$ as the separator. A subformula $f_{nm}$ is *parameterized by* $k$ if $\wedge_{m\in nbr(n)} f_{nm}$ is in $\varphi_n$, i.e., there are $k$ replicas of $f_{nm}$ in $\varphi_n$, where each replica corresponds to a different neighbor. We call $\varphi_n$ the *parametric local specification* because $\varphi_n$ contains subformulas parameterized by $k$. We manually rewrite $\varphi_n$ into an *abstract local specification* independent of $k$, denoted $\varphi_n^a$.

In this work, we accomplish the rewriting through a *counting abstraction*. For each set of related variables $\cup_{m\in nbr(n)}\{p_{nm}\} \subseteq \Sigma_n$ are replaced by abstract variables that approximate the number of neighbors such that $p_{nm}$ holds. Example abstract variables include $p_n^{\mathtt{A}}$ for '$\forall m : p_{nm}$,' and $p_n^{\mathtt{E}}$ for '$\exists m : p_{nm}$.'

The abstract specification $\varphi_n^a$ is created using variables in the abstract alphabet $\Sigma_n^a$ while preserving the behavior of $P_n$ described in $\varphi_n$. The specification transformation only applies to $\varphi_n$ subject to certain format restrictions, i.e., (roughly) the formulas in $\varphi_n$ describe the current or next-time behavior of $P_n$ as invariants, and we only allow restricted formats of eventualities. For example, eventualities parameterized by $k$, such as $\wedge_{m\in nbr(n)}\mathsf{AF}p_m$, are not allowed.

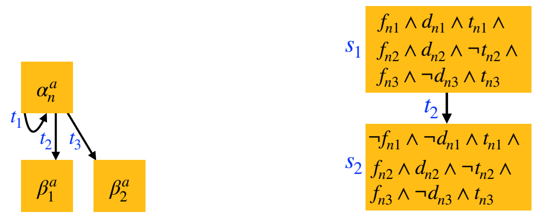
In $eg_1$, $\beta_{nm}$ indicates a reduction by one in the number of forks owned by $P_n$ and the number of dirty forks adjacent to $P_n$. This count update applies to every $m \in nbr(n)$ that meets $\alpha_{nm}$. Hence, for any abstract precondition composed of at least one neighbor in $\alpha_{nm}$, the abstract postconditions must reflect the results of applying the corresponding updates.

Given, for example, $\Sigma_n^a = \{f_n^{\mathtt{E}}, f_n^{\mathtt{A}}, d_n^{\mathtt{E}}, d_n^{\mathtt{A}}, t_n^{\mathtt{E}}, t_n^{\mathtt{A}}\}$, the count update in $eg_1$ affects all abstract preconditions containing $f_n^{\mathtt{E}} \wedge t_n^{\mathtt{E}} \wedge d_n^{\mathtt{E}}$. Let $\alpha_n^a = f_n^{\mathtt{E}} \wedge f_n^{\mathtt{A}} \wedge d_n^{\mathtt{E}} \wedge \neg d_n^{\mathtt{A}} \wedge t_n^{\mathtt{E}} \wedge \neg t_n^{\mathtt{A}}$, representing '$P_n$ owns all forks and some but not all request tokens, and some but not all forks adjacent to $P_n$ are dirty,' be one of the affected abstract preconditions. The abstract property for $\alpha_n^a$ is formulated as:

$$eg_1^a \coloneqq \mathsf{AG}(\alpha_n^a \Rightarrow (\mathsf{AY}_n(\alpha_n^a \vee \beta_1^a \vee \beta_2^a) \wedge \mathsf{EX}_n\alpha_n^a \wedge \mathsf{EX}_n\beta_1^a \wedge \mathsf{EX}_n\beta_2^a))$$

where $\beta_1^a = f_n^{\mathtt{E}} \wedge \neg f_n^{\mathtt{A}} \wedge d_n^{\mathtt{E}} \wedge \neg d_n^{\mathtt{A}} \wedge t_n^{\mathtt{E}} \wedge \neg t_n^{\mathtt{A}}$ and $\beta_2^a = f_n^{\mathtt{E}} \wedge \neg f_n^{\mathtt{A}} \wedge \neg d_n^{\mathtt{E}} \wedge \neg d_n^{\mathtt{A}} \wedge t_n^{\mathtt{E}} \wedge \neg t_n^{\mathtt{A}}$. For simplicity, the subscript $n$ is omitted from $\beta_1^a$ and $\beta_2^a$.

Formula $eg_1^a$ covers all transitions from concrete states satisfying $\alpha_n^a$. When $\alpha_{nm}$ is false for all $m \in nbr(n)$, transitions of $n$ leave $\alpha_n^a$ unchanged. Otherwise, when $\alpha_{nm}$ is triggered for some $m$, $\alpha_n^a$ changes to $\beta_1^a$ or $\beta_2^a$ through an $n$-transition depending on how $\alpha_n^a$ is satisfied. Each set of related shared valuations satisfying $\alpha_n^a$ is expressed as a combination of parametric preconditions. For combinations containing $\alpha_{nm}$ and $(f_{nm} \wedge d_{nm} \wedge \neg t_{nm})$, $\alpha_n^a$ becomes $\beta_1^a$ because 'some but not all dirty forks owned by $P_n$ are requested by the neighbors' in $\alpha_n^a$. For combinations containing $\alpha_{nm}$ but not $(f_{nm} \wedge d_{nm} \wedge \neg t_{nm})$, $\alpha_n^a$ becomes $\beta_2^a$ because 'all dirty forks owned by $P_n$ are requested by the neighbors' in $\alpha_n^a$.




(a) (b)

**Fig. 2.** An example of abstract structure and concrete structure.

For any abstract precondition $\alpha_n^a$, we save, as *context*, the combinations that satisfy $\alpha_n^a$ and the count updates for each parametric precondition in the combinations. The context is used to track changes in shared variables between $n$ and individual neighbors. The context is not included in $\varphi_n^a$ but will be used as auxiliary information to concretize the abstract model constructed from $\varphi_n^a$.

***Step 3, in Section 5.*** Section 5 describes the steps and complexity for automatically constructing an abstract model $H_n^a$ from a satisfiable $\varphi_n^a$.

Fig. 2a shows an abstract structure constructed from $eg_1^a$ rooted in a state labeled $\alpha_n^a$. This state has three outgoing $n$-transition: $t_1$, $t_2$, and $t_3$. Interference transitions are inferred from these existing $n$-transitions. In this example, a neighbor $m$, performing an isomorphic transition, affects the states of $P_n$ whose label contains $\neg f_n^{\mathtt{A}} \wedge d_n^{\mathtt{E}} \wedge \neg t_n^{\mathtt{A}}$ by increasing the number of forks owned by $P_n$ and the number of clean forks adjacent to $P_n$.

***Step 4, in Section 6.*** We explain the concretization of the synthesized model and give the overall correctness theorem in Section 6. We show that the abstract model, $H_n^a$, can be concretized into a concrete model, $H_n^c$ of $\varphi_n^c$, for any $k$. Based on the global-local reduction in Theorem 1, the abstract process $P_n^a$ extracted from $H_n^a$ can be instantiated as programs on nodes of underlying networks to form instances that satisfy the global specification.

An example of concretization is as follows. When $k = 3$, Fig. 2b shows the transition of a given concrete state $s_1$ derived from the abstract model in Fig. 2a. State $s_1$ is in the concrete interpretation of $\alpha_n^a$, and by examining the context of $\alpha_n^a$, $t_2$ in Fig. 2a is the only transition whose starting combination matches that of $s_1$. Hence, $s_2$ is derived from $\beta_1^a$ by applying the count update to $m_1$.

## 4 Local Specification Transformation

In this section, we form the abstract alphabet $\Sigma_n^a$, formulate restrictions on the parametric local specification $\varphi_n$, explain the connection between $\varphi_n$ and the abstract local specification $\varphi_n^a$ resulting from its rewriting, give an example of rewriting $\varphi_n$ into $\varphi_n^a$, and describe the manual rewriting steps in detail.

### 4.1 Defining Abstract Variables

***Parametric Variables, Terms, and Properties.*** In this paper, we stick to the following naming convention for variables. In the parametric alphabet $\Sigma_n$, the internal variables of $n$ are indexed $n$, and the parametric external variables shared between $n$ and $m \in nbr(n)$ are indexed $nm$, $mn$, or simply $m$.

A *parametric term* is of the form $\vee_w t_w$ or $\wedge_w t_w$, where $w$ ranges over $nbr(n)$, and $t_w$ comprises (optional) logical connectives and shared variables between $n$ and $w$. Examples of parametric terms are $\vee_w p_w$ and $\wedge_w (p_w \vee q_w)$.

We define *parametric local properties* as $(\wedge_{m \in nbr(n)} f_{nm}) \mid g_n$, where $f_{nm}$ is a Fair CTL formula containing parametric terms, internal variables, and shared variables between $n$ and $m$. The definitions of $g_n$ and $f_{nm}$ are almost identical, but $g_n$ does not contain any shared variables other than those that contained in the parametric terms in $g_n$.

***Abstract Variables, Terms, and Properties.*** An *abstract shared variable* represents the quantification over a set of parametric shared variables. An *abstract local property* is a Fair CTL formula whose alphabet $\Sigma_n^a$ is derived by replacing all parametric variables in $\Sigma_n$ with abstract shared variables. An *abstract term* comprises logical connectives and variables in $\Sigma_n^a$.

We describe how to form $\Sigma_n^a$. To begin with, for each set of related variables $\cup_{m \in nbr(n)} \{p_m\}$ in $\Sigma_n$, we define two abstract variables, $p_n^{\mathtt{E}}$ and $p_n^{\mathtt{A}}$, to approximate the number of neighbors for which $p_m$ is true. The abstract variables are subscripted $n$ because these variables count the number of neighbors centered on $n$.

We relax the definition of parametric terms by allowing their indices to range over subsets of $nbr(n)$. For example, in the atomic snapshots protocol, we specify that 'exactly one unread and unselected port is selected to be read next' as:

$$\wedge_{m \in nbr(n)} \mathsf{AG}((\neg r_m \wedge \neg s_m) \Rightarrow \mathsf{EX}_n(\neg r_m \wedge s_m \wedge (\wedge_{w \in nbr(n) \setminus \{m\}} \neg s_w)))$$

where $r_m$ (resp. $s_m$) indicates that the port shared by $n$ and $m$ is read (resp. selected) by $n$, and $\wedge_{w \neq m} \neg s_w$ ranges over all neighbors of $n$ except $m$.

Accordingly, more abstract variables can be added to $\Sigma_n^a$. For example, in addition to $s_n^{\mathtt{E}}$ and $s_n^{\mathtt{A}}$, we can express the number of selected neighbors more precisely by adding an abstract variable $s_n^{\mathtt{One}}$ that means '$s_m$ is true for exactly one neighbor $m$.' However, we are only interested in abstract alphabets whose size is independent of $k$.

When the neighbors of $n$ are divided into multiple partitions, and the behavior of $P_n$ interacting with neighbors in each partition $nbr(n)_i$ is specified separately, we create abstract shared variables separately for each $nbr(n)_i$. For example, suppose half of the neighbors are in $nbr(n)_1$, and the other half are in $nbr(n)_2$. The variables in $\cup_{m \in nbr(n)} \{s_m\}$ are divided into $\cup_{m_1 \in nbr(n)_1} \{s_{m_1}\}$ and $\cup_{m_2 \in nbr(n)_2} \{s_{m_2}\}$. Assume that in $nbr(n)_1$, one unread port is selected at a time, but in $nbr(n)_2$, all unread ports are selected simultaneously. Instead of defining $s_n^{\mathtt{E}}$ and $s_n^{\mathtt{A}}$ for $nbr(n)$, we create separate abstract variables for each partition, e.g., $s_n^{\mathtt{E}_i}$, which means 'there exists a neighbor $m_i \in nbr(n)_i$ such that $s_{m_i}$ is true,' and $s_n^{\mathtt{A}_i}$, which means '$s_{m_i}$ is true for all neighbors in $nbr(n)_i$.'

***Concrete vs. Abstract Interpretation.*** For any $k$, let $M^c := (S^c, S^{c,0}, T^c, \lambda^c)$ be a concrete system defined over the language of $\varphi_n^c$. Let $t$ be a parametric term that defines a set of concrete states $S_t^c$ in $M^c$, called the *concrete interpretation* of $t$. Given any abstract system $M^a := (S^a, S^{a,0}, T^a, \lambda^a)$, let $h$ be an abstract term whose *abstract interpretation* is a set of abstract states $S_h^a$ in $M^a$. Here, $h$ also defines a set of concrete states $S_h^c$. If for all $k$, $S_t^c \subseteq S_h^c$, we say $h$ *includes* $t$.

For example, for all multi-process models, $p_n^{\mathtt{E}}$ includes $\vee_{w \in nbr(n)} p_w$, and $p_n^{\mathtt{A}} \vee q_n^{\mathtt{A}} \vee (p_n^{\mathtt{E}} \wedge q_n^{\mathtt{E}})$ includes $\wedge_{w \in nbr(n)}(p_w \vee q_w)$. If for every state $s \in S^c$, $M^c, s \models t$ iff $M^c, s \models h$, then we say $t \equiv h$, i.e., $t$ and $h$ are *equivalent*.

### 4.2 Format Restrictions

In this subsection, we describe the format restrictions on $\varphi_n$. We assume that a more complex specification is refined (manually, in most cases) into this restricted format, e.g., [9] contains several examples of such refinements.

The parametric local specifications of interest are in the form of *init-spec* $\wedge$ *other-spec*, where *init-spec* specifies a single initial state. If there are multiple initial states, we create a copy of $\varphi_n$ for each initial state such that each copy has a different *init-spec* but the same *other-spec*.

We further require that every parametric property of the form $\wedge_{m \in nbr(n)} f_{nm}$ in *other-spec* be either $\wedge_m \mathsf{AG}(\alpha_{nm} \Rightarrow \mathsf{AY}_n \beta_{nm})$ or $\wedge_m \mathsf{AG}(\alpha_{nm} \Rightarrow \mathsf{EX}_n \gamma_{nm})$. Among the remaining properties in *other-spec*, for any property $g_n$ containing parametric terms, either $g_n$ is restricted to $\mathsf{AG}(\alpha_n \Rightarrow \mathsf{AY}_n \beta_n)$ or $\mathsf{AG}(\alpha_n \Rightarrow \mathsf{EX}_n \gamma_n)$, or we require that for every parametric term $t$ in $g_n$, there is an equivalent abstract term $h$ based on $\Sigma_n^a$. Here, the preconditions and postconditions are terms consisting of logical connectives and variables.

If every parametric term $t$ in $g_n$ is exactly interpreted as an abstract term $h$, we rewrite $g_n$ into an equivalent property by replacing each $t$ with $h$. For example, $\mathsf{AG}((\vee_{w \in nbr(n)} p_w) \Rightarrow \mathsf{AF}(\wedge_{w \in nbr(n)} p_w))$ is rewritten as $\mathsf{AG}(p_n^{\mathtt{E}} \Rightarrow \mathsf{AF} p_n^{\mathtt{A}})$. An eventuality is allowed only if it is in a formula $g_n$ that can be exactly rewritten.

Finding an exact interpretation for every parametric term in $g_n$ may not be possible. This is the case for $\mathsf{AG}((\wedge_w (p_w \vee q_w)) \Rightarrow \mathsf{AF}((\wedge_w p_w) \wedge (\wedge_w q_w)))$, where $w \in nbr(n)$, because $\wedge_w (p_w \vee q_w)$ cannot be exactly described by any abstract term given $\Sigma_n^a = \{p_n^{\mathtt{E}}, p_n^{\mathtt{A}}, q_n^{\mathtt{E}}, q_n^{\mathtt{A}}\}$. We can add more variables to $\Sigma_n^a$ to ensure that the requirement for exact interpretation is met, such as $z_n^{\mathtt{A}} := \wedge_w z_w$, which represents the count of a new parametric variable $z_w := p_w \vee q_w$. The eventuality is rewritten into $\mathsf{AG}(z_n^{\mathtt{A}} \Rightarrow \mathsf{AF}(p_n^{\mathtt{A}} \wedge q_n^{\mathtt{A}}))$. Except for those properties that can be directly rewritten as equivalent abstract properties, the remaining properties require a more complex rewriting approach, as described in Section 4.5.

### 4.3 Parametric vs. Abstract Local Specifications

In this subsection, we elaborate on the goals of the specification rewriting transformation. Subsequently, we describe in Section 4.5 how to perform concrete-to-abstract specification rewriting to achieve the goals. We explain in Section 6 how to concretize an abstract model into concrete models.

For each precondition $\alpha_{nm}$ in parametric properties of the form $\wedge_{m\in nbr(n)} f_{nm}$ in $\varphi_n$, there is a universal postcondition $\beta_{nm}$ and a set of existential postconditions $\{\gamma_{nm}\}$. For any $k$, $S^c_{\alpha_{nm}}$ denotes the concrete interpretation of $\alpha_{nm}$, i.e., $S^c_{\alpha_{nm}}$ consists of the concrete states that satisfy $\alpha_{nm}$ for at least one $m$. Given an abstract alphabet $\Sigma^a_n$, let $\alpha^a_n$ be an abstract term whose interpretation, denoted $S^c_{\alpha^a_n}$, is a superset of $S^c_{\alpha_{nm}}$. The rewriting guarantees that the abstract postconditions of $\alpha^a_n$ preserve the following property:

**Lemma 1.** *For any $k$, for each state $s^c \in S^c_{\alpha^a_n}$ and neighbor $m \in nbr(n)$, if $M^c, s^c \models \alpha_{nm}$, then $M^c, s^c \models \mathsf{AY}_n\beta_{nm}$, and for each $\gamma_{nm}$, $M^c, s^c \models \mathsf{EX}_n\gamma_{nm}$.*

*Proof.* When $S^c_{\alpha^a_n} \subseteq S^c_{\alpha_{nm}}$, the rewriting ensures Lemma 1 by establishing the following on an abstract system $M^a$. For each abstract state $s^a \in S^a_{\alpha^a_n}$, $M^a, s^a \models \mathsf{AY}_n\beta^a_n$, and $M^a, s^a \models \mathsf{EX}_n\gamma^a_n$ for each $\gamma^a_n$, where $S^a_{\alpha^a_n}$ is the abstract interpretation of $\alpha^a_n$. Here, $\beta^a_n$ is obtained by applying the update from $\alpha_{nm}$ to $\beta_{nm}$ to each applicable $m$, and $\{\gamma^a_n\}$ is obtained by applying each update from $\alpha_{nm}$ to $\gamma_{nm}$ to one of the applicable neighbors.

When $S^c_{\alpha^a_n} \setminus S^c_{\alpha_{nm}} \neq \emptyset$, there are multiple ways to satisfy $\alpha^a_n$. Each way of satisfying $\alpha^a_n$ is represented as a combination of parametric preconditions, denoted $\hat{A}_i$, where $S^c_{\alpha^a_n} \cap S^c_{\alpha_{nm}}$ corresponds to the combinations containing $\alpha_{nm}$. The rewriting establishes that, for each abstract state $s^a \in S^a_{\alpha^a_n}$, $M^a, s^a \models \mathsf{AY}_n(\vee_i \beta^a_{n,i})$, and $M^a, s^a \models \mathsf{EX}_n(\beta^a_{n,i} \wedge \gamma^a_{n,i})$ for each $\gamma^a_{n,i}$, where $i$ ranges over the combinations in $\alpha^a_n$.

The concrete successor states of each $s^c \in S^c_{\alpha^a_n}$ are obtained by concretizing a subset of the abstract successor states of $\alpha^a_n$. These abstract successor states meet the parametric postconditions, as do the concrete successor states. □

Note that $\alpha^a_n$, in its disjunctive normal form, can be split into multiple abstract preconditions $\cup_j\{\alpha^a_j\}$, where each $\alpha^a_j$ is a conjunctive clause in $\alpha^a_n$. The postconditions of each $\alpha^a_j$ must guarantee the aforementioned conditions for each concrete state $s^c \in S^c_{\alpha^a_j}$, where $S^c_{\alpha^a_j}$ is the concrete interpretation of $\alpha^a_j$.

Similarly, for each precondition $\alpha_n$ in parametric properties $g_n$ of the form $\mathsf{AG}(\alpha_n \Rightarrow \mathsf{AY}_n\beta_n)$ or $\mathsf{AG}(\alpha_n \Rightarrow \mathsf{EX}_n\gamma_n)$, we define an abstract term $\alpha^a_n$ for $\alpha_n$ such that $S^c_{\alpha_n} \subseteq S^c_{\alpha^a_n}$, where $S^c_{\alpha_n}$ is the concrete interpretation of $\alpha_n$. The rewriting guarantees the following property: (See Appendix C.1 for proof.)

**Lemma 2.** *For any $k$, for each state $s^c \in S^c_{\alpha^a_n}$, if $M^c, s^c \models \alpha_n$, then $M^c, s^c \models \mathsf{AY}_n\beta_n$, and for each $\gamma_n$, $M^c, s^c \models \mathsf{EX}_n\gamma_n$.*

### 4.4 Specification Rewriting: Example

We take part of the atomic snapshots protocol as an example to illustrate the rewriting procedure: (We specify that any variable that is not explicitly updated remains unchanged in the next state.) The informal, full description of the protocol is given in Section 7.

c1 [Initial Condition] An odd-numbered pass begins, and all shared ports are unread and unselected by $P_n$: $o_n \wedge (\wedge_m \neg r_m) \wedge (\wedge_m \neg s_m)$

c2 $P_n$ stays on the current pass until all ports are read:
$\mathsf{AG}((o_n \wedge (\vee_m \neg r_m)) \Rightarrow \mathsf{AY}_n o_n)$
c3 When all ports are read, $P_n$ starts the next pass (an even-numbered pass):
$\mathsf{AG}((o_n \wedge (\wedge_m r_m)) \Rightarrow \mathsf{AY}_n(\neg o_n \wedge (\wedge_m \neg r_m) \wedge (\wedge_m \neg s_m)))$
c4 $P_n$ selects one unread port at a time:
$\wedge_m \mathsf{AG}((\neg r_m \wedge \neg s_m) \Rightarrow \mathsf{EX}_n(\neg r_m \wedge s_m \wedge (\wedge_{w \neq m} \neg s_w)))$
c5 $P_n$ reads and deselects the selected port:
$\wedge_m \mathsf{AG}((\neg r_m \wedge s_m \wedge (\wedge_{w \neq m} \neg s_w)) \Rightarrow \mathsf{AY}_n(r_m \wedge \neg s_m))$

In this example, indices $m \in nbr(n)$ and $w \in nbr(n) \setminus \{m\}$. Variables $r_m$ (i.e., read port $m$) and $s_m$ (i.e., select port $m$) are shared variables between $n$ and $m$. Variable $o_n$ (i.e., the current pass number is odd) is internal to $n$.

Properties c1-c3 are directly rewritten into equivalent abstract properties a1-a3, respectively. Properties $\mathsf{AG}(\neg s_n^{\mathtt{E}} \Rightarrow \neg s_n^{\mathtt{A}})$ and $\mathsf{AG}(s_n^{\mathtt{A}} \Rightarrow s_n^{\mathtt{E}})$ for $s$, are part of the abstract specification, as are similar properties for $r$. Next, we rewrite c4-c5.

a1 [Initial Condition] $o_n \wedge \neg r_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{E}}$
a2 $\mathsf{AG}((o_n \wedge \neg r_n^{\mathtt{A}}) \Rightarrow \mathsf{AY}_n o_n)$
a3 $\mathsf{AG}((o_n \wedge r_n^{\mathtt{A}}) \Rightarrow \mathsf{AY}_n(\neg o_n \wedge \neg r_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{E}}))$

Let $\alpha_1 := \neg r_m \wedge \neg s_m$ and $\alpha_2 := \neg r_m \wedge s_m \wedge (\wedge_{w \neq m} \neg s_w)$ be preconditions in c4 and c5, respectively. For each neighbor, either $\alpha_1$ is true, $\alpha_2$ is true, or neither (further specified in the full protocol). The consistent combinations of $\alpha_1$ and $\alpha_2$ are $\{\alpha_1\}$, $\{\alpha_2\}$, and $\{\alpha_1, \alpha_2\}$, where $\{\alpha_1\}$ stands for '$\alpha_1$ holds for all $m$,' $\{\alpha_2\}$ stands for '$\alpha_2$ holds for one $m$, requiring $k = 1$,' and $\{\alpha_1, \alpha_2\}$ stands for '$\alpha_2$ holds for one $m$, and $\alpha_1$ holds for every $w \neq m$, requiring $k \geq 2$.'

These combinations are rewritten into abstract terms: $\{\alpha_1\}$ is rewritten into $\neg r_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{E}}$, $\{\alpha_2\}$ is rewritten into $\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}}$, and $\{\alpha_1, \alpha_2\}$ is rewritten into $\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}}$. For each abstract term, we create an abstract property that is premised on that term:

a4 $\mathsf{AG}((\neg r_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{E}}) \Rightarrow (\mathsf{EX}_n(\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}}) \wedge \mathsf{EX}_n(\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}})))$
a5 $\mathsf{AG}((\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}}) \Rightarrow \mathsf{AY}_n(r_n^{\mathtt{A}} \wedge \neg s_n^{\mathtt{E}}))$
a6 $\mathsf{AG}((\neg r_n^{\mathtt{E}} \wedge s_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{A}} \wedge s_n^{\mathtt{One}}) \Rightarrow (\mathsf{AY}_n(r_n^{\mathtt{E}} \wedge \neg s_n^{\mathtt{A}}) \wedge \mathsf{EX}_n(\neg r_n^{\mathtt{A}} \wedge s_n^{\mathtt{E}} \wedge s_n^{\mathtt{One}})))$

For each abstract precondition $\alpha^a$, the postconditions are derived as follows. First, we record the change to $\alpha_1$ in c4 and the change to $\alpha_2$ in c5 as count updates. The existential update in c4, denoted $s_+^{\mathtt{e}}$, represents the change from $\neg s_m$ to $s_m$ applied to a single $m$ that meets $\alpha_1$. We also append to the update any abstract terms in the postcondition, e.g. $\wedge_{w \neq m} \neg s_w$. Whereas the update in c5, denoted $(r_+, s_-)^{\mathtt{u}}$, is universal and applies to all neighbors that meet $\alpha_2$.

Next, for each combination in $\alpha^a$, we apply the count updates to the parametric preconditions in the combination and rewrite the resulting combinations as abstract postconditions of $\alpha^a$. We take a6 as an example. The abstract precondition in a6 has a single combination, $\{\alpha_1, \alpha_2\}$. The result of applying $s_+^{\mathtt{e}}$ is that $\neg r_m \wedge s_m \wedge (\wedge_{w \neq m} \neg s_w)$ holds for exactly one $m$. The result is included in the existential abstract postcondition in a6. The result of applying $(r_+, s_-)^{\mathtt{u}}$ is that $r_m \wedge \neg s_m$ holds for the $m$ whose precondition is $\alpha_2$. The result is included in the universal abstract postcondition in a6.

### 4.5 Specification Rewriting: Full Procedure

The input to the manual specification rewriting procedure, i.e., the parametric local specification $\varphi_n$ can be viewed as a list of properties, denoted $\cup_{i\in[0,l)}\{\psi_i\}$, where $\psi_i$ is either a non-parametric property or a parametric property of the form $\wedge_{m\in nbr(n)} f_{nm}$ or $g_n$. We focus on cases where $l$ and $k$ are independent. In this subsection, we describe the rewriting procedure in three stages.

***Step 1. Grouping Properties.*** We group the properties in $\varphi_n$ according to preconditions. Let $\alpha_{\psi_i}$ be the precondition in property $\psi_i$ and $\mathcal{P}(\cup_{i\in[0,l)}\{\alpha_{\psi_i}\})$ be the power set of the preconditions for all properties in $\varphi_n$. We construct the set of parametric preconditions, denoted $A$, as follows. Starting with $A = \emptyset$, for each $\alpha \in \mathcal{P}(\cup_{i\in[0,l)}\{\alpha_{\psi_i}\})$, $\alpha$ is appended to $A$ if $\alpha$ is propositionally consistent, and there is no superset of $\alpha$ in $A$. Expressions are inconsistent if they contradict each other, such as $p_m$ and $\wedge_{w\in nbr(n)}\neg p_w$. Parametric variables are inconsistent if they cannot both hold for the same neighbor, such as $p_m$ and $\neg p_m$.

***Step 2. Obtaining Abstract Preconditions.*** We perform case splitting to obtain the abstract preconditions. Starting with $\mathcal{G} = \mathcal{P}(A)$, we remove from $\mathcal{G}$ any combination of parametric preconditions $\hat{A} \in \mathcal{G}$ if the concrete interpretation of $\hat{A}$ contains concrete states that violate the following conditions: (1) for each $m \in nbr(n)$, one of the parametric preconditions in $\hat{A}$ holds, (2) each parametric precondition $\alpha \in \hat{A}$ holds for at least one $m \in nbr(n)$, and (3) internal values and parametric terms hold for $n$. The resulting $\mathcal{G}$ is the set of combinations.

Each $\hat{A} \in \mathcal{G}$ is then transformed into an abstract term $h$ that includes $\hat{A}$. If $h$ is in disjunctive normal form, $h$ is further split into multiple conjunctive terms. Each term is an abstract precondition. Depending on the abstract alphabet $\Sigma_n^a$, an abstract precondition may correspond to multiple combinations, where each combination represents a different way of satisfying the abstract precondition.

***Step 3. Deriving Abstract Postconditions.*** We conduct a case analysis to derive the abstract postconditions for each abstract precondition.

First, we extract the updates to a parametric precondition $\alpha$ from $\varphi_n$. If $\mathsf{AY}_n\beta$ (resp. $\mathsf{EX}_n\gamma$) is a next-time property of $\alpha$, then the update from $\alpha$ to $\beta$ (resp. $\gamma$) is *universal* (resp. *existential*). There is no update for variables that remain unchanged. Updates to the variables in $\cup_{m\in nbr(n)}\{p_m\}$ are considered changes in the number of neighbors for which $p_m$ is true. Therefore, updates are either incremental, denoted $p_+$, decremental, denoted $p_-$, or propositionally inconsistent (e.g., the postcondition contains $p_m$ and $\neg p_m$ for the same neighbor). The updates of $\alpha$ is the union of the updates of each $\alpha_{\psi_{i\in[0,L)}} \in \alpha$. The updates of $\alpha$ are divided into a universal subset, $B$, and an existential subset, $\Gamma$.

Following that, let $\alpha^a$ be an abstract precondition. For each combination $\hat{A}$ included in $\alpha^a$, we do the following. For each $\alpha \in \hat{A}$, we apply the universal updates in $B$ (to all neighbors satisfying $\alpha_{nm}$) and express the result as an abstract term $\beta^a$ that includes the updated valuation. The term $\beta^a$ is the universal postcondition of $\hat{A}$. Similarly, for each $\alpha \in \hat{A}$ and each existential update $\gamma \in \Gamma$, we express the result of applying $\gamma$ (to one neighbor satisfying $\alpha_{nm}$) as abstract terms $\{\gamma^a\}$, where each term $\gamma^a$ is an existential postcondition of $\hat{A}$.

If $\alpha^a$ includes a single combination (e.g., a4-a6 in Section 4.4) or multiple combinations but with the same universal postcondition $\beta^a$, then we create an $\mathsf{AY}_n\beta^a$ subformula for $\beta^a$ and an $\mathsf{EX}_n\gamma^a_j$ subformula for each existential postcondition $\gamma^a_j$ of $\alpha^a$. The resulting subformulas are filled in to create an abstract property $\mathsf{AG}(\alpha^a \Rightarrow (\mathsf{AY}_n\beta^a \wedge (\wedge_j \mathsf{EX}_n\gamma^a_j)))$ for $\alpha^a$.

However, if the universal postconditions of the combinations in $\alpha^a$ are different from each other (as in the example in Section 3), we create an abstract property $\mathsf{AG}(\alpha^a \Rightarrow (\mathsf{AY}_n(\vee_i\beta^a_i) \wedge (\wedge_{ij}\mathsf{EX}_n(\beta^a_i \wedge \gamma^a_{ij}))))$ for $\alpha^a$. Here, $i$ ranges over the combinations in $\alpha^a$, and $j$ ranges over the existential postconditions of each combination $\hat{A}_i$.

We can take a shortcut to generate $\varphi^a_n$ by directly rewriting the properties that satisfy the exact interpretation requirements (e.g., c2-c3 in Section 4.4) into equivalent abstract properties and then rewriting the remaining properties (e.g., c4-c5 in Section 4.4) using the detailed procedure.

***Generate Contexts.*** We introduce context as auxiliary information to maintain the connection between parametric and abstract properties. The *context* of an abstract precondition $\alpha^a$ is a list of tuples, where each tuple consists of a combination $\hat{A}$ in $\alpha^a$, and the count updates for each parametric precondition $\alpha \in \hat{A}$. For each update, we indicate whether it is universal or existential. For example, the context of property a4 in Section 4.4 is a tuple $(\{\alpha_1\}, \alpha_1 : (s_+, \wedge_{w \neq m} \neg s_w)^{\mathsf{e}})$, where $\{\alpha_1\}$ is the only combination in the abstract precondition of a4, and $(s_+, \wedge_{w \neq m} \neg s_w)^{\mathsf{e}}$ is the only update of $\alpha_1$.

## 5 Synthesis

The work in [29] describes a tableau-based, iterative decision procedure for concrete local specifications $\varphi^c_n$ written in Fair CTL. We modify the algorithm to take an abstract local specification $\varphi^a_n$ as input and automatically build an abstract model $H^a_n$ of $\varphi^a_n$. We describe the modified algorithm and its complexity.

A *tableau* of $n$ is denoted by $\mathcal{T}_n \coloneqq (V, R, L)$, where $V$ is a set of nodes, $R$ is a left-total transition relation, and $L : V \to 2^{cl(\varphi^a_n)}$ is a labeling function using $cl(\varphi^a_n)$ to represent the Fischer-Ladner closure of $\varphi^a_n$ [4, 17], i.e., the negation, subset, and fixpoint closure of Fair CTL modalities in $\varphi^a_n$. The formulas are in *negation normal form*, i.e., negation is driven to the proposition-level.

In $\mathcal{T}_n$, $V \coloneqq C \cup D$ and $R \subseteq (C \times D) \cup (D \times C)$, where $C$ is a set of AND-nodes (i.e., the potential states of $P^a_n$), and $D$ is a set of OR-nodes. AND-OR transitions are labeled with $n$ or the letter $m$, whereas OR-AND transitions are unlabeled. Since the process indices of the neighbors are hidden, when applying the abstraction, $m$ denotes a generic neighbor of $n$ without an explicit index.

Starting from an initial tableau $\mathcal{T}^0_n$, the algorithm iteratively constructs $\mathcal{T}^{i+1}_n$ from $\mathcal{T}^i_n$ until a fixpoint tableau is reached. For simplicity, we assume that the input specification specifies a single initial condition. The root of $\mathcal{T}^0_n$ is an OR-node labeled $\{\varphi^a_n\}$, and $\mathcal{T}^0_n$ is constructed by adding successors to leaf nodes.

**Table 1.** The expansion rules (cf. [13, 14]).

| | | |
|---|---|---|
| $a = f \wedge g$ | $a_1 = f$ | $a_2 = g$ |
| $a = \mathsf{A}(f\mathsf{W}g)$ | $a_1 = g$ | $a_2 = f \vee \mathsf{AYA}(f\mathsf{W}g)$ |
| $a = \mathsf{E}(f\mathsf{W}g)$ | $a_1 = g$ | $a_2 = f \vee \mathsf{EXE}(f\mathsf{W}g)$ |
| $a = \mathsf{AG}g$ | $a_1 = g$ | $a_2 = \mathsf{AYAG}g$ |
| $a = \mathsf{EG}g$ | $a_1 = g$ | $a_2 = \mathsf{EXEG}g$ |
| $a = \mathsf{AY}g$ | $a_1 = \mathsf{AY}_n g$ | $a_2 = \mathsf{AY}_m g$ |
| $b = f \vee g$ | $b_1 = f$ | $b_2 = g$ |
| $b = \mathsf{A}(f\mathsf{U}g)$ | $b_1 = g$ | $b_2 = f \wedge \mathsf{AYA}(f\mathsf{U}g)$ |
| $b = \mathsf{E}(f\mathsf{U}g)$ | $b_1 = g$ | $b_2 = f \wedge \mathsf{EXE}(f\mathsf{U}g)$ |
| $b = \mathsf{AF}g$ | $b_1 = g$ | $b_2 = \mathsf{AYAF}g$ |
| $b = \mathsf{EF}g$ | $b_1 = g$ | $b_2 = \mathsf{EXEF}g$ |
| $b = \mathsf{EX}g$ | $b_1 = \mathsf{EX}_n g$ | $b_2 = \mathsf{EX}_m g$ |

***Constructing the Initial Tableau (cf. [13, 14]).*** The successors of an OR-node $d$ describe various ways of satisfying the formulas in $L(d)$. The algorithm computes the successors of $d$ by expanding $L(d)$ according to the rules in Table 1. In Table 1, the conjunctive rules expand formula $a$ into $a_1 \wedge a_2$, and the disjunctive rules expand formula $b$ into $b_1 \vee b_2$. Each successor of $d$ is an AND-node $c$, corresponding to an expansion of $L(d)$.

We show how to derive the expansions of $L(d)$, i.e., the label of each successor $c$ of $d$. Starting from $L(c) = L(d)$, if $a \in L(c)$, both $a_1$ and $a_2$ are added to $L(c)$. If $b \in L(c)$, one of $b_1$ and $b_2$ is added to $L(c)$. Each formula is expanded at most once. The expansion continues until all unexpanded formulas in $L(c)$ are *elementary* formulas, i.e., each unexpanded formula is either a variable $p$, its negation $\neg p$, or a formula whose modality is indexed $\mathsf{AY}$ or $\mathsf{EX}$.

The successors of an AND-node $c$ are created to satisfy the next-time formulas indexed by $n$ in $L(c)$. For each $\mathsf{EX}_n g \in L(c)$, an OR-node $d$ is created as a successor of $c$ such that $g \in L(d)$. For each $\mathsf{AY}_n f \in L(c)$, $f \in L(d)$ for each $d$. If there are no $\mathsf{EX}_n g$ in $L(c)$, a single successor $d$ is created for $c$ such that for every $\mathsf{AY}_n f$ in $L(c)$, $f$ is in $L(d)$. If there is no next-step formula in $L(c)$, the successor $d$ copies the label of $c$ and points back to $c$ to form a self-loop of $c$.

The transition from $c$ to each $d$ is labeled $n$. The modified decision procedure extends the labels of AND-OR transitions to include the corresponding context. Let $\alpha_n^a$ be an abstract precondition in $L(c)$, and $\beta_n^a$ (resp. $\gamma_n^a$) be an abstract postcondition satisfied in $L(d)$. The algorithm appends the combinations and updates corresponding to $\beta_n^a$ (resp. $\gamma_n^a$) in the context of $\alpha_n^a$ to the label of $(c, d)$.

Duplicate nodes of the same type and label are merged. When there are no more leaf nodes, an initial tableau consisting of nodes reachable from the root via $n$-transitions is constructed.

***Constructing the Fixpoint Tableau (cf. [29]).*** The next tableau, $\mathcal{T}_n^{i+1}$, is constructed based on $\mathcal{T}_n^i$. The algorithm captures the updates in the labels of the $n$-transitions in $\mathcal{T}_n^i$ and creates isomorphic updates (caused by $m$) as interference transitions for the AND-nodes affected by these updates.

**Table 2.** The deletion rules for tableau pruning (cf. [29]).

| | |
|---|---|
| *deleteP* | Delete any node whose label is propositionally inconsistent. |
| *deleteOR* | Delete any OR-node all of whose successors have been deleted. |
| *deleteAND* | Delete any AND-node one of whose successors has been deleted. |
| *deleteEU* | Delete any node $v$ if $\mathsf{E}(f\mathsf{U}g) \in L(v)$, and there is no AND-node $c'$ reachable from $v$ via a finite path $\pi$ such that $g \in L(c')$ and $f \in L(c)$ for each AND-node $c$ on $\pi$ except $c'$. |
| *deleteAU* | Delete any node $v$ if $\mathsf{A}(f\mathsf{U}g) \in L(v)$, and there is no subdag $\mathcal{U}$ rooted at $v$ such that $g \in L(c')$ for each leaf AND-node $c'$ in $\mathcal{U}$, and $f \in L(c)$ for each internal AND-node $c$ of $\mathcal{U}$. |
| *deleteEG* | Delete any node $v$ if $\mathsf{EG}g \in L(v)$, and there is no fair full path $\pi$ starting from $v$ such that $g \in L(c)$ for each AND-node $c$ on $\pi$. |
| *deleteJoint* | Delete any AND-node $c_n$ if every AND-node $e_n$ forming the joint state $(e_n, c_m)$ has been deleted, where $c_m$ is isomorphic to $c_n$. |
| *deleteUpdate* | Delete any AND-node $c_n$ if one of its updates is inconsistent. |

Let $(c_n, c'_n)$ be an $n$-transition from AND-node $c_n$ to $c'_n$ in $\mathcal{T}_n^i$, where $c_n$ and $c'_n$ have different shared states by updating the parametric precondition $a_{nm}$ that holds in $c_n$. When a neighbor of $n$ executes an isomorphic transition $(c_m, c'_m)$, an AND-node $e_n$ in $\mathcal{T}_n^i$ is affected by the $m$-transition if a parametric precondition isomorphic to $\alpha_{nm}$ is in one of the combinations in $e_n$. Note that an interference update is existential for $n$. The update only changes the values of variables shared with the neighbor that initiated the transition.

Let $y$ be a shared state, and $\cup_i\{y'_i\}$ be a set of $m$-successor shared states of $y$ under the interference of a generic neighbor $m$. The algorithm computes the successors of each affected AND-node $e_n$ as follows. For each $y'_i$, an OR-node whose label contains $y'_i$ is created as an $m$-successor of $e_n$. For each $\mathsf{EX}_m g \in L(e_n)$, an OR-node whose label contains $g$ and $\vee_i y'_i$ is also added to $e_n$ as an $m$-successor. For each $\mathsf{AY}_m f \in L(e_n)$, $f \in L(d)$ for every $m$-successor $d$ of $c$.

After adding interference transitions to $\mathcal{T}_n^i$, the decision procedure adds successors to each leaf node via $n$-transitions. The resulting tableau is $\mathcal{T}_n^{i+1}$. When $\mathcal{T}_n^{i+1} = \mathcal{T}_n^i$, the fixpoint tableau, denoted $\mathcal{T}_n^*$, is reached. As with the algorithm in [29], an error message is raised if it is detected that $\mathcal{T}_n^*$ may produce a non-outward-facing model. If no error message is issued, $\mathcal{T}_n^*$ is pruned by repeatedly applying the deletion rules in Table 2. If the root of $\mathcal{T}_n^*$ is deleted, $\varphi_n^a$ is unsatisfiable. Otherwise, the pruned $\mathcal{T}_n^*$ is unraveled into a model $H_n^a$, from which the representative process $P_n^a$ is extracted by removing interference transitions.

***Checking Fulfillment of Eventualities.*** We elaborate on the differences between the pruning steps for abstract and concrete fixpoint tableaux.

Note that an abstract precondition $\alpha_n^a$ can include multiple combinations of parametric preconditions. We compute successor combinations for $\alpha_n^a$ by applying updates to each combination in $\alpha_n^a$. These successor combinations, written as abstract terms, are considered preconditions for subsequent transitions. Thus, we say that a path $(\alpha_0^a, \alpha_1^a, \alpha_2^a)$ of abstract shared states *includes concrete paths* if the combinations produced by transition $(\alpha_0^a, \alpha_1^a)$ intersect with the combinations required by transition $(\alpha_1^a, \alpha_2^a)$.

We require that the fulfilling path $\pi$ in deletion rules *deleteEU* and *deleteEG* (in Table 2) be a path that includes concrete paths. Since the fulfillment of abstract eventualities must be independent of $k$, we further require that $\pi$ be a valid path for all $k$. For example, suppose $\pi$ contains a transition whose precondition shows that a shared variable $p_{nm}$ holds for multiple neighbors. Since $\pi$ does not apply to the case of only one neighbor, either $\pi$ is not a path to verify the fulfillment of the eventuality or $k$ cannot take the value one.

The construction of subdags that verify the fulfillment of AU is also modified accordingly. In *deleteAU*, *subdag* $\mathcal{U}$ is a directed graph extracted from tableau $\mathcal{T}$. The paths in $\mathcal{U}$ are abstract paths that include concrete paths. Beyond that, $\mathcal{U}$ contains no fair full path, i.e., all cycles in $\mathcal{U}$ must be finite. For example, because $k$ is unbounded but not infinite, a cycle is finite if it contains $p_+$ as the only update for $\cup_{m \in nbr(n)}\{p_m\}$ but does not contain any node labeled $p_n^{\mathtt{A}}$.

Rule *deleteUpdate* removes nodes with inconsistent updates. The rule ensures that no propositional inconsistencies are in the values of concrete variables shared with any neighbor, even if the values of abstract variables are consistent.

***Correctness and Complexity.*** We show that the modified decision procedure satisfies Lemma 3. See Appendix C.2 for proof.

**Lemma 3.** *The constructed system $H_n^a$ satisfies the abstract local specification $\varphi_n^a$, is closed under neighboring interference, and unravels into an outward-facing abstract representative process $P_n^a$.*

The modified decision procedure constructs a fixpoint tableau $\mathcal{T}_n^*$ whose size is bounded by $2^{|\varphi_n^a|}$. The cost of constructing $H_n^a$ from $\varphi_n^a$ is in time polynomial in the size of $\mathcal{T}_n^*$. Therefore, the synthesis of $P_n^a$ can be done in time $O(2^{|\varphi_n^a|})$, independent of $k$ and $K$. The synthesized $P_n^a$ is instantiated on the network nodes. The run time of instantiating $P_n^a$ to form a protocol instance is $O(K)$.

## 6 Concretization and Soundness

The synthesizer constructs an abstract model $H_n^a$ from an abstract local specification $\varphi_n^a$. We show that, for any $k$, a concrete system $H_n^c$ is derived from $H_n^a$ by *concretization*, and $H_n^c$ is a model of the concrete local specification $\varphi_n^c$.

We show how concretization works. First, we create the root of $H_n^c$ and label it with the concrete *init-spec* in $\varphi_n^c$. The label of the concrete root is included in the label of the abstract root in $H_n^a$. Next, we add successor states to each concrete leaf state. Upon termination, the resulting system is $H_n^c$.

Let $s^c$ be a concrete leaf state and $s^a$ in $H_n^a$ be the abstract state corresponding to $s^c$. The steps to attach successor states $\cup_i\{t_i^c\}$ to $s^c$ are as follows. If $s^c$ is in the concrete interpretation of a combination $\hat{A}$ in the context of $s^a$, then for each successor combination $\hat{A}'$ of $\hat{A}$, there is an abstract state $t^a$ such that transition $(s^a, t^a) \in T_n^a$ and $\hat{A}'$ is included in $t^a$. Based on transition $(\hat{A}, \hat{A}')$, a concrete state $t_i^c$ is created and attached to $s^c$ as a successor. The label of $t_i^c$ is obtained by replacing the abstract shared state in $\lambda(t^a)$ with the concrete shared state derived by applying the updates of $(\hat{A}, \hat{A}')$ to $\lambda(s^c)$.

Note that, by symmetry, there are multiple concrete ways to apply an existential update. For example, when $\neg r_m \wedge \neg s_m$ is the parametric precondition for existential update $s_+^{\mathsf{e}}$, different choices of selecting one from all candidate neighbors that meet the precondition yield different concrete successor states.

**Theorem 3.** *If an abstract model $H_n^a$ is synthesized from $\varphi_n^a$, then for any $k$, a concrete system $H_n^c$ derived from $H_n^a$ by concretization satisfies $\varphi_n^c$.*

By Theorem 3 (see Appendix C.3 for proof), the abstract process $P_n^a$, extracted from $H_n^a$, is instantiated at each node of a given network graph to form a protocol instance. By Theorem 1, all instances satisfy the global specification.

## 7 Experiment

We implemented the modified decision procedure in Python. We constructed models for two example parametric protocols (informally described as follows) using the synthesizer on a machine with a 2.5 GHz CPU and 16 GB of memory. The model of the atomic snapshots protocol has 19 states and takes about 50 seconds to construct. The dining philosophers model has a total of 44 states, and its construction takes about 290 seconds. See appendix D for details.

***Atomic Snapshots.*** We describe an example atomic snapshots protocol [1]. An atomic snapshot object is a group of $K$ processes with an operation called 'snapshot' that returns the simultaneous state of all processes. In a fully connected network, an atomic process shares a port with each neighbor, so that the process writes to the port (values indicating the current state of the process and sequence numbers that distinguish the order of write operations), and the neighbor reads from the port. That is, a process reads $k$ ports, however, a process can read only one port at a time. The objective of a snapshots algorithm is to implement an atomic snapshot object using these read/write variables.

To implement the objective, a process reads all ports once and then reads them all again. If the two read passes yield the same outcome, then the process takes as a snapshot the list of value-sequence number pairs read from the ports. That is, for each neighbor, the process picks a point between the end of the first pass and the start of the second pass and records the data in the port as the state of the neighbor. Otherwise, the process restarts the snapshot operation.

When a process is not running the snapshot operation, it can simultaneously write to all shared ports. The algorithm is wait-free, but the snapshot operation may never complete because ports read by the process are prone to changes made by its neighbors, thus interfering with the creation of a valid snapshot.

***Dining Philosophers.*** We apply the classic dining philosophers protocol to hypercubes. For each increase of one in the number of neighbors, $k$ goes to $k+1$. The hypercube size grows from $K = 2^k$ to $K = 2^{k+1}$.

A total of $K$ philosophers are seated at the nodes of a $k$-dimensional cube, with a fork shared between each pair of neighbors. In local states, philosophers are either 'thinking,' 'hungry,' or 'eating.' A thinking philosopher becomes hungry in finite time. A hungry philosopher enters into eating by holding all adjacent forks. A pair of philosophers cannot hold a particular fork at the same time, which prevents any neighbors from simultaneously accessing the eating state.

To pursue fair access to the eating state so that no philosopher is permanently hungry, we use the hygienic solution introduced in [8], where the conflict between hungry neighbors over who eats is resolved using a precedence graph. The evolution of the precedence graph is implemented using forks, dirty/clean status of forks, and request tokens for forks. Each fork (resp. token) is held by one of its two adjacent philosophers. Initially, all philosophers are thinking, all forks are dirty, and tokens and forks are possessed in such a way that the precedence graph is acyclic (see Appendix A). A hungry philosopher sends its tokens to request unowned forks. Upon receiving a token, if the requested fork is dirty, the philosopher cleans the fork and gives it to the neighbor. A hungry philosopher turns into eating when it owns all adjacent forks, and each fork is either clean or not requested. An eating philosopher dirties all its forks at once. The described local specification enforces the absence of starvation.

## 8 Related Work and Conclusion

To solve the parameterized synthesis problem, Emerson and Srinivasan [12] use counting arguments for fully symmetric systems. Emerson and Attie [5] construct a pair-system, which is the product of two connected processes obtained from each other by swapping the indices. Jacobs and Bloem [18] apply cutoff results and attack the distributed synthesis problem on token-ring networks with a semi-decision procedure for bounded synthesis. Ehlers and Finkbeiner [11] take an automata-theoretic approach on rotation-symmetric architectures, where the symmetry property is partially encoded in the specification and partially ensured by post-processing an obtained computation tree. Klinkhamer and Ebnenasir [19] synthesize the code for symmetric processes in self-stabilizing parameterized unidirectional rings, where a set of legitimate states of the template process are provided as inputs. Bollig et al. [7] focus on round-bounded parameterized systems and turn the synthesis problem into multi-pushdown games, where a winning strategy is computed in non-elementary time.

This work extends an approach that reduces the parameterized synthesis problem to the problem of synthesizing representatives of equivalence classes of balanced, locally symmetric processes [29]. Our approach applies to network families constructed from abstract tiling patterns, including but not limited to rings, tori, rectangular meshes, fully connected networks, and hypercubes. Assuming a uniform system and unconditionally fair scheduling, the approach finds an outward-facing representative $P_n$ in worst-case time exponential in the length of the abstract local specification. The approach is incomplete because the formulation of local specifications for $P_n$ is done manually and has formatting restrictions. Parametric synthesis of reactive, distributed systems is generally undecidable [26]. Furthermore, the approach is not fully automated as global information and proofs are required. The choice of abstract alphabets is up to the designer, taking into account parametric local specifications and formatting guidelines.

Emerson and Clarke [13], Manna and Wolper [22] addressed the *satisfaction* problem and proposed a decision procedure that determines if there is a set of processes such that their joint behavior satisfies a temporal logic formula. The implementation is reactive but not *open* because the environment is assumed to be cooperative instead of adversarial. Pnueli and Rosner [25, 27] introduced the *realization* problem that finds out whether there exists a system process implementation against all possible inputs obtained from the environment. As shown in [26], the realization problem for propositional temporal specifications in general architectures is undecidable (recursively enumerable complete) even for finite-state machines. Finkbeiner and Schewe [15] characterized undecidable architectures as cases when processes have incomparable information from the environment. Even for decidable cases, the size of the global state space, i.e., the automata-theoretic product of $K$ sequential processes, is exponential in $K$.

The distributed synthesis problem has been addressed in many different ways, including the automata-theoretic approach [20], game-theoretic approach [23], bounded synthesis [28, 16], and symbolic synthesis [10]. Our method is based on the tableau construction in [13]. The main difference is that we repeatedly add neighboring interference until we get a fixpoint tableau. Our approach is not open, but it is also not closed because the way that neighbors interfere with the local states of $P_n$ is not specified as a property of $P_n$.

***Future Work.*** Parametric systems can contain several connectivity templates and process types. Our approach can be modified to apply to systems with multiple equivalence classes, such as red/black protocols, where each red (resp. black) process has only black (resp. red) neighbors (cf. [24]), and we are applying the work to other protocols (cf. [9]). We are investigating automation in deriving abstract local specifications, applying the work using general abstract domains, and extending the work to fault-tolerant systems (cf. [4]).

## References

1. Afek, Y., Dolev, D., Attiya, H., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. In: Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing. p. 1–13. PODC '90, Association for Computing Machinery, New York, NY, USA (1990). https://doi.org/10.1145/93385.93394
2. Alford, M.W., Ansart, J.P., Hommel, G., Lamport, L., Liskov, B., Mullery, G.P., Schneider, F.B.: Distributed systems: methods and tools for specification. An advanced course. Springer-Verlag, Berlin, Heidelberg (1985)

3. Apt, K.R., Kozen, D.: Limits for automatic verification of finite-state concurrent systems. Inf. Process. Lett. **22**(6), 307–309 (1986)
4. Attie, P.C., Arora, A., Emerson, E.A.: Synthesis of fault-tolerant concurrent programs. ACM Trans. Program. Lang. Syst. **26**(1), 125–185 (Jan 2004). https://doi.org/10.1145/963778.963782, https://doi.org/10.1145/963778.963782
5. Attie, P.C., Emerson, E.A.: Synthesis of concurrent systems with many similar processes. ACM Trans. Program. Lang. Syst. **20**(1), 51–115 (Jan 1998). https://doi.org/10.1145/271510.271519
6. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
7. Bollig, B., Lehaut, M., Sznajder, N.: Round-Bounded Control of Parameterized Systems. In: ATVA. LNCS, vol. 11138, pp. 370–386 (2018)
8. Chandy, K.M., Misra, J.: The drinking philosophers problem. ACM Trans. Program. Lang. Syst. **6**(4), 632–646 (oct 1984). https://doi.org/10.1145/1780.1804
9. Chandy, K., Misra, J.: Parallel Program Design: A Foundation. Computer Science Series, Addison-Wesley Publishing Company (1988)
10. Ehlers, R.: Symbolic bounded synthesis. Formal Methods in System Design **40**(2), 232–262 (2012). https://doi.org/10.1007/s10703-011-0137-x
11. Ehlers, R., Finkbeiner, B.: Symmetric synthesis. In: FSTTCS. LIPIcs, vol. 93, pp. 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
12. Emerson, A., Srinivasan, J.: A decidable temporal logic to reason about many processes. In: Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing. p. 233–246. PODC '90, Association for Computing Machinery, New York, NY, USA (1990). https://doi.org/10.1145/93385.93425
13. Emerson, E.A., Clarke, E.M.: Using branching time temporal logic to synthesize synchronization skeletons. Science of Computer Programming **2**(3), 241–266 (1982). https://doi.org/10.1016/0167-6423(83)90017-5
14. Emerson, E.A., Lei, C.L.: Temporal reasoning under generalized fairness constraints. In: Monien, B., Vidal-Naquet, G. (eds.) STACS 86. pp. 21–36. Springer Berlin Heidelberg, Berlin, Heidelberg (1986)
15. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05). pp. 321–330 (2005). https://doi.org/10.1109/LICS.2005.53
16. Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer **15**(5-6), 519–539 (2013). https://doi.org/10.1007/s10009-012-0228-z
17. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**(2), 194–211 (1979). https://doi.org/10.1016/0022-0000(79)90046-1
18. Jacobs, S., Bloem, R.: Parameterized synthesis. Log. Methods Comput. Sci. **10**(1) (2014). https://doi.org/10.2168/LMCS-10(1:12)2014
19. Klinkhamer, A.P., Ebnenasir, A.: Synthesizing parameterized self-stabilizing rings with constant-space processes. In: Dastani, M., Sirjani, M. (eds.) Fundamentals of Software Engineering. pp. 100–115. Springer International Publishing, Cham (2017)
20. Kupferman, O., Vardi, M.: Synthesizing distributed systems. In: Proceedings 16th Annual IEEE Symposium on Logic in Computer Science. pp. 389–398 (2001). https://doi.org/10.1109/LICS.2001.932514
21. Lamport, L.: Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers. Addison-Wesley Longman Publishing Co., Inc., USA (2002)

22. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. ACM Trans. Program. Lang. Syst. **6**(1), 68–93 (jan 1984). https://doi.org/10.1145/357233.357237
23. Mohalik, S., Walukiewicz, I.: Distributed games. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS. pp. 338–351. Springer, Berlin, Heidelberg (2003)
24. Namjoshi, K.S., Trefler, R.J.: Symmetry reduction for the local mu-calculus. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 379–395. Springer International Publishing, Cham (2018)
25. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 179–190. POPL '89, Association for Computing Machinery, New York, NY, USA (1989). https://doi.org/10.1145/75277.75293
26. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings 31st Annual Symposium on Foundations of Computer Science. pp. 746–757 vol.2 (1990). https://doi.org/10.1109/FSCS.1990.89597
27. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Proceedings of the 16th International Colloquium on Automata, Languages and Programming. p. 652–671. ICALP '89, Springer-Verlag, Berlin, Heidelberg (1989)
28. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) Automated Technology for Verification and Analysis. pp. 474–488. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
29. Zhang, R., Trefler, R.J., Namjoshi, K.S.: Synthesizing locally symmetric parameterized protocols from temporal specifications. In: Griggio, A., Rungta, N. (eds.) 22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022. pp. 235–244. IEEE (2022). https://doi.org/10.34727/2022/ISBN.978-3-85448-053-2_30

## A Dining Philosophers Initial Conditions for Hypercubes

In the hygienic solution to the dining philosophers problem in [8], conflict resolution is achieved using a precedence graph. In a precedence graph, each pair of conflicting philosophers is connected by a directed edge from the one with higher priority to the one with lower priority. A philosopher in conflict but with a higher priority can eat. The hygienic solution is an implementation of a precedence graph that is fair and remains acyclic such that each hungry philosopher eventually gets the higher priority. Therefore, we only need to show that the protocol is initially set up in such a way that the precedence graph is acyclic. We formulate the problem as, given any $k$-dimensional cube, finding a directed acyclic graph (DAG) by assigning a direction to each edge in the $k$-cube.

Nodes in a $k$-cube are labeled with binary numbers from 0 to $2^k - 1$, where the labels of any pair of neighbors differ by one bit. Figure 1b shows one way to label a 3-cube. Given a $k$-cube, a $(k+1)$-cube is created as follows. Let $A$ and $B$ be two copies of the $k$-cube. We append a leading 0 (resp. 1) to the label of each node in $A$ (resp. $B$) and connect each pair of nodes that have different leading bits but the same remaining bits with an edge. The resulting graph is a $(k+1)$-cube, and we turn it into a DAG by directing each edge from the node with a smaller label to the node with a larger label.

We prove that the DAG is acyclic by induction. The DAG for a 3-cube in Figure 1b is acyclic. We show that if the DAG for a $k$-cube is acyclic, then the DAG for a $(k+1)$-cube is also acyclic. By the hypothesis, the DAGs for $k$-cubes $A$ and $B$ are acyclic. Any edge connecting these two DAGs is directed from a node in $A$ to a node in $B$. Therefore, the DAG for a $(k+1)$-cube is also acyclic.

In the dining philosophers protocol, initially, all forks are dirty, and each fork and its request token are held by different philosophers. For a pair of neighbors, the one with the fork has lower priority. Hence, given an acyclic DAG for a $k$-cube, each fork is initialized to be held by the lower-priority philosopher of the pair. We generate three abstract initial conditions for the representative process to cover all possible initial conditions for processes in the parametric protocol, namely $f_n^{\mathtt{A}} \wedge d_n^{\mathtt{A}} \wedge \neg t_n^{\mathtt{E}}$, $f_n^{\mathtt{E}} \wedge \neg f_n^{\mathtt{A}} \wedge d_n^{\mathtt{A}} \wedge t_n^{\mathtt{E}} \wedge \neg t_n^{\mathtt{A}}$, and $\neg f_n^{\mathtt{E}} \wedge d_n^{\mathtt{A}} \wedge t_n^{\mathtt{A}}$.

## B Fair CTL Semantics

The relation of entailment is inductively defined over system $M = (S, S^0, T, \lambda)$ as follows:

- $M, s \models p$ iff $p \in \lambda(s)$, for atomic proposition $p$.
- $M, s \models \neg f$ iff not $(M, s \models f)$.
- $M, s \models f \wedge g$ iff $M, s \models f$ and $M, s \models g$.
- $M, s_0 \models \mathsf{EX}_n f$ iff there exists $\pi = (s_0, s_1, ...)$ in $M$, such that $(s_0, s_1) \in T_n$, $M, \pi \models \Phi$, and $M, s_1 \models f$.
- $M, s_0 \models \mathsf{AY}_n f$ iff for all $\pi = (s_0, s_1, ...)$ in $M$, if $(s_0, s_1) \in T_n$ and $M, \pi \models \Phi$, then $M, s_1 \models f$.
- $M, s_0 \models \mathsf{E}(f \mathsf{U} g)$ iff there exists $\pi = (s_0, s_1, ...)$ in $M$, such that $M, \pi \models \Phi$, and there exists $i \geq 0$, such that $M, s_i \models g$, and for all $0 \leq j < i$, $M, s_j \models f$.
- $M, s_0 \models \mathsf{A}(f \mathsf{U} g)$ iff for all $\pi = (s_0, s_1, ...)$ in $M$, if $M, \pi \models \Phi$, then there exists $i \geq 0$, such that $M, s_i \models g$, and for all $0 \leq j < i$, $M, s_j \models f$.

## C Proofs of Lemmas and Theorems

### C.1 Proof of Lemma 2

The proof of Lemma 2 is similar to that of Lemma 1, except for the parametric preconditions and postconditions. In a parametric property $g_n$ containing parametric terms, the precondition $\alpha_n$ and postconditions $\beta_n$ and $\gamma_n$ describe the values of the shared variables between $n$ and all its neighbors (rather than a single $m \in nbr(n)$). The proof of Lemma 2 is as follows:

*Proof.* When $S^c_{\alpha_n} = S^c_{\alpha^a_n}$, the rewriting establishes that, for each abstract state $s^a \in S^a_{\alpha^a_n}$, $M^a, s^a \models \mathsf{AY}_n \beta^a_n$, and $M^a, s^a \models \mathsf{EX}_n \gamma^a_n$ for each $\gamma^a_n$, where $M^a$ is an abstract system, and $S^a_{\alpha^a_n}$ is the abstract interpretation of $\alpha^a_n$. Here, $\beta^a_n$ is derived by applying the update from $\alpha_n$ to $\beta_n$, and each $\gamma^a_n \in \{\gamma^a_n\}$ is derived by applying the update from $\alpha_n$ to one $\gamma_n \in \{\gamma_n\}$.

When $S^c_{\alpha_n} \subset S^c_{\alpha^a_n}$, there are multiple combinations (of parametric preconditions), denoted $\cup_i\{\hat{A}_i\}$, in $\alpha^a_n$. Here, $S^c_{\alpha_n}$ is the union of concrete interpretations of the combinations in $\alpha^a_n$ that contain $\alpha_{nm}$. The rewriting establishes that, for each abstract state $s^a \in S^a_{\alpha^a_n}$, $M^a, s^a \models \mathsf{AY}_n(\vee_i \beta^a_{n,i})$, and $M^a, s^a \models \mathsf{EX}_n(\beta^a_{n,i} \wedge \gamma^a_{n,i})$ for each $\gamma^a_{n,i}$, where $i$ ranges over the combinations in $\alpha^a_n$.

The concrete successor states of each $s^c \in S^c_{\alpha^a_n}$ are obtained by concretizing all or part of the abstract successor states of $\alpha^a_n$. These abstract successor states meet the parametric postconditions, and so do the concrete successor states. □

### C.2 Proof of Lemma 3

*Proof.* Lemma 3 shows the correctness of the modified synthesizer for abstract local specifications $\varphi^a_n$. Following the classical decision procedure for CTL in [13], the proposed synthesizer propagates the formulas according to the expansion rules. Let $c$ be an AND-node. If $\mathsf{AY}_i f \in L(c)$, then for every $i$-successor OR-node $d$ of $c$, $f \in L(d)$, and for every successor AND-node $c'$ of $d$, $f \in L(c')$. If $\mathsf{EX}_i g \in L(c)$, then for at least one $i$-successor OR-node $d$ of $c$, $g \in L(d)$, and for every successor AND-node $c'$ of $d$, $g \in L(c')$. Here, $i$ is either $n$ or $\mathbf{m}$. Note that there is no $\mathsf{AY}_{\mathbf{m}} f$ or $\mathsf{EX}_{\mathbf{m}} g$ in $\varphi^a_n$, but other formulas, such as an $\mathsf{AG}$ formula, can be expanded to produce $\mathbf{m}$-indexed next-time properties.

The generation of interference transitions caused by individual neighbors of $n$ additionally involves changes in the shared states. These changes are inferred from the current tableau of $n$. For each change in a shared state, an affected AND-node has at least one corresponding $\mathbf{m}$-successor OR-node reflecting the change. Given $\varphi^a_n$ as the input to the decision procedure, the size of tableaux created from $\varphi^a_n$ is bounded by $2^{cl(\varphi^a_n)}$. The iterative tableau construction eventually terminates when no more nodes and transitions can be added. The fixpoint tableau is closed under neighboring interference.

Eventualities are checked for fulfillment against the deletion rules in [29]. In this work, abstract fulfilling paths are redefined considering the finite parameter $k$ and the relationship between abstract and concrete paths. If the root of the fixpoint tableau is not removed in the pruning step, the tableau unravels into a model of $\varphi^a_n$.

The modified decision procedure adopts the outward-facing filter used in the decision procedure in [29] to ensure that the tableaux that are not filtered out unravel into an outward-facing process. □

### C.3 Proof of Theorem 3

We prove Theorem 3 according to the following outline. By Lemma 1 and 2, $\varphi^a_n$ preserves the next-time properties in $\varphi_n$. By Lemma 3, the tableau-based decision procedure constructs an abstract system $H^a_n$ that satisfies $\varphi^a_n$. By concretization, for any $k$, a concrete system $H^c_n$ is obtained from $H^a_n$, and $H^c_n$ is a model of $\varphi^c_n$, which again is based on the preservation of properties in $\varphi_n$. We show that states, transitions, paths, and subdags concretized from the abstract model preserve the corresponding parametric properties.

*Proof.* **States.** Let $\alpha$ be a parametric precondition. The specification rewriting procedure creates a set of disjoint abstract preconditions $\cup_i\{\alpha_i^a\}$ such that $S_\alpha^c \subseteq \cup_i S^c_{a_i^\alpha}$, where $S_\alpha^c$ and $S^c_{a_i^\alpha}$ are the concrete interpretation of $\alpha$ and $a_i^\alpha$, respectively. Here, $S^c_{a_i^\alpha}$ is derived from the abstract interpretation $S^a_{a_i^\alpha}$ by concretization. That is, each concrete state $s^c \in S_\alpha^c$ is also in $S^c_{a_i^\alpha}$, and $s^c$ is concretized from an abstract state $s^a \in S^a_{a_i^\alpha}$.

***Transitions.*** The next-time behavior of $P_n$ is described as pairs of preconditions and postconditions. For each abstract precondition $\alpha_i^a \in \cup_i\{\alpha_i^a\}$ and each combination $\hat{A}$ included in $\alpha_i^a$, the rewriting procedure produces a set of abstract postconditions. Based on Lemma 1 and 2, these abstract postconditions preserve the next-time properties of each parametric precondition in $\hat{A}$. By Lemma 3, the synthesizer generates a set of abstract successor states $T^a_{\alpha_i^a}$ for the states in $S^a_{a_i^\alpha}$.

Let $s^c$ be a concrete state in $S_\alpha^c$ and $S^c_{a_i^\alpha}$. Let $\hat{A}$ be the combination in $\alpha_i^a$ corresponding to $s^c$. By concretization, the concrete successor states of $s^c$, denoted $\{t^c\}$, are derived from $T^a_{\alpha_i^a}$ using the updates of $\hat{A}$. If $\mathsf{AY}_n\beta$, then for all $t \in \{t\}$, $\beta$ is true in $t$ (for all neighbors satisfying $\alpha$ in $s^c$). If $\mathsf{EX}_n\gamma$, there exists $t^c$ such that $\gamma$ is true in $t^c$ (for at least one neighbor satisfying $\alpha$ in $s^c$).

***Eventualities.*** Respecting the format restrictions in Section 4.2, eventualities in $\varphi_n$ are rewritten by replacing parametric terms with exact abstract terms. The fulfillment of an eventuality in an abstract state $s^a$ implies the fulfillment of the eventuality in any concrete state $s^c$ derived from $s^a$ by concretization. □

# D Applications

Here, we show the abstract models constructed for the example protocols.

***The Atomic Snapshot Model.*** Fig. 3 shows an abstract model constructed for the atomic snapshot protocol, where rectangles are states (the pink rectangle is the initial state), solid arrows are transitions by $P_n$, and dashed arrows are interference transitions caused by a generic neighbor $P_m$. The figure explicitly shows the $n$-transitions that form finite self-loops, while the finite self-loops labeled $m$ are omitted. The representative $P_n$ is extracted from the model by removing all dashed arrows.

We describe the protocol parametrically with the following variables. The abstract alphabet is formed by replacing the parametric shared variables with abstract variables $r_n^{\mathtt{E}}$, $r_n^{\mathtt{A}}$, $s_n^{\mathtt{E}}$, and $u_n^{\mathtt{E}}$. For simplicity, abstract variables $s_m^{\mathtt{One}}$, $s_m^{\mathtt{A}}$, and $u_m^{\mathtt{A}}$ are omitted in Fig. 3.

- $\cup_{m\in nbr(n)}\{r_m\}$, where $r_m$ represents that $P_n$ reads the value and sequence number from the port shared with $P_m$
- $\cup_{m\in nbr(n)}\{s_m\}$, where $s_m$ represents that $P_n$ selects the port shared with $P_m$ as the next read port

- $\cup_{m \in nbr(n)}\{u_m\}$, where $u_m$ represents that the value and sequence number of the port shared with $P_m$ are updated
- $w_n$ represents that $P_n$ writes to the ports shared with all its neighbors
- $m_n$ represents that $P_n$ finds a port mismatch between two reads
- $o_n$ stands for the first pass of reading
- $s_n$ represents that $P_n$ takes a snapshot

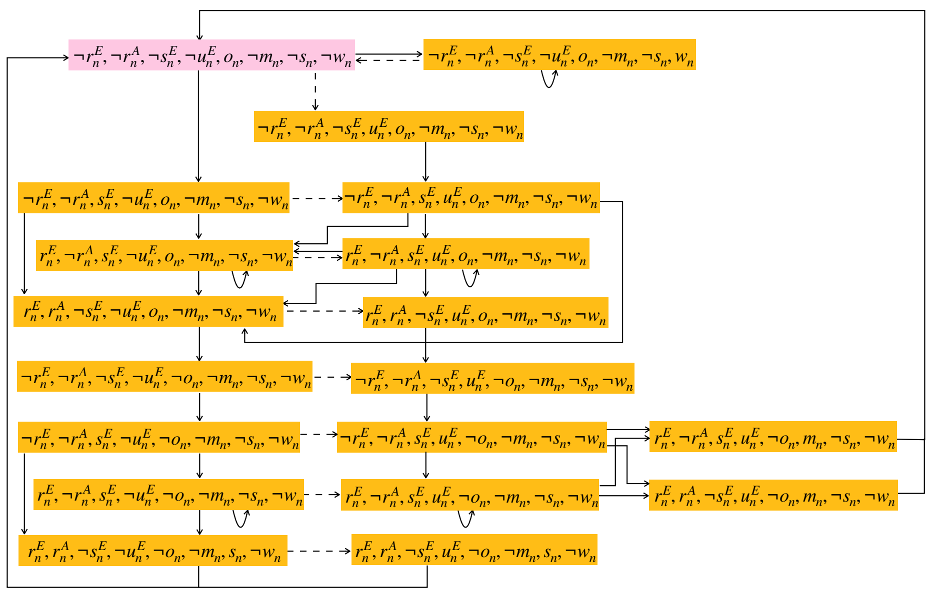



**Fig. 3.** The abstract model for the atomic snapshot protocol.

***The Dining Philosophers Model.*** Fig. 4 shows an abstract model constructed for the dining philosophers protocol. The states in Fig. 4 are labeled by assigning a value to each variable in the abstract alphabet $\Sigma^a_n$. The parametric alphabet of the protocol contains variables listed as follows. We convert the parametric shared variables into $f^{\mathtt{E}}_n$, $f^{\mathtt{A}}_n$, $d^{\mathtt{E}}_n$, $d^{\mathtt{A}}_n$, $t^{\mathtt{E}}_n$, and $t^{\mathtt{A}}_n$ to form $\Sigma^a_n$.

- $\cup_{m \in nbr(n)}\{f_{nm}\}$, where $f_{nm}$ represents that $P_n$ holds the shared fork
- $\cup_{m \in nbr(n)}\{d_{nm}\}$, where $d_{nm}$ represents that the fork shared with $P_m$ is dirty
- $\cup_{m \in nbr(n)}\{t_{nm}\}$, where $t_{nm}$ represents that $P_n$ holds the request token
- $T_n$ represents that $P_n$ is thinking
- $H_n$ represents that $P_n$ is hungry
- $E_n$ represents that $P_n$ is eating

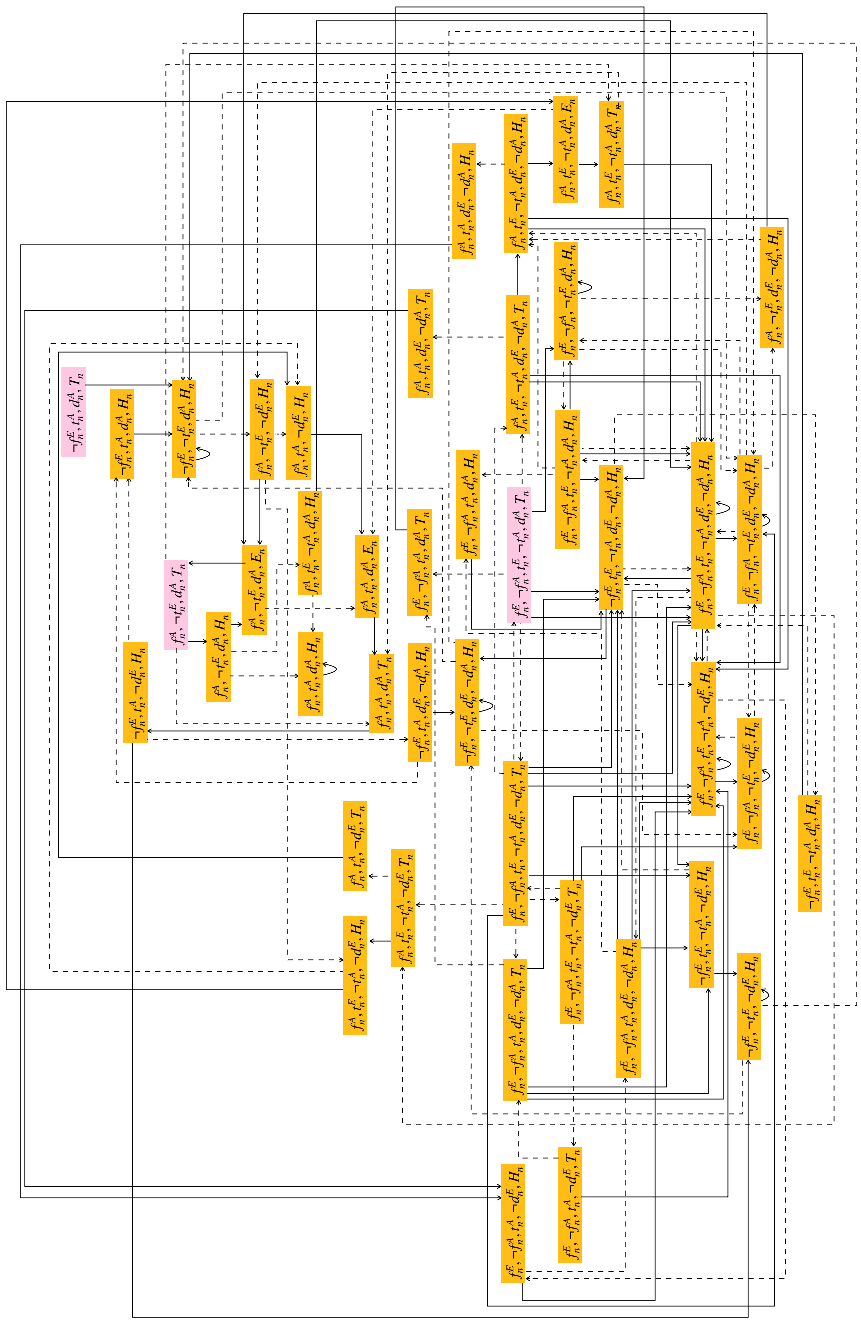


**Fig. 4.** The abstract model for the dining philosophers protocol.