Parameterized Synthesis for Distributed Applications with Consensus

Nouraldin Jaber[1], Swen Jacobs[2], Milind Kulkarni[1], and Roopsha Samanta[1]

[1] Purdue University
[2] CISPA Helmholtz Center for Information Security

Abstract. Distributed systems are hard to get right. Efforts in reasoning about distributed systems have primarily focused on verification and language design, with some limited efforts in synthesis for systems with a *fixed* number of processes. In this paper, we target *parameterized synthesis* of distributed systems that use *consensus protocols*, such as Paxos, as a building block to provide higher-level functionality. This paper makes several contributions. First, to enable scalability of reasoning about such complex distributed systems, we propose encapsulating the details of consensus into a simple abstraction, `choose`. Second, we prove that parameterized verification of safety properties for such systems with `choose` is decidable. We then show that many interesting classes of systems with `choose` have tractable *cutoffs* for parameterized verification, yielding a decision procedure for parameterized synthesis. Finally, we use our tool to demonstrate feasibility of synthesis for several examples of distributed applications that build on consensus, including a model of the Small Aircraft Tracking System (SATS) and a Distributed Mobile Robotics (DMR) problem.

1 Introduction

Distributed systems, and especially distributed systems that require some form of consensus, are hard to build, and hard to get right. Consequently, verification of distributed systems is a long-standing and fruitful research area [3, 4, 2, 68, 16, 17, 71, 18, 61, 14, 19, 59], with substantial efforts [58, 9, 49, 12, 22, 21, 54, 52, 72] in verifying intricate consensus protocols, such as Paxos [48] and Raft [56]. These efforts, however, do not directly simplify design and analysis of distributed applications that use consensus as merely a step in ensuring global properties. For instance, consider a distributed smoke detector (Fig. 1 shows a constituent process) whose intended behavior is as follows. Upon detecting smoke, the processes coordinate using a consensus protocol to choose at most two processes to report the smoke to the fire department. More precisely, the *safety specification* is: (1) at most two processes report to the fire department and (2) processes that do not detect smoke do not report to the fire department. The *liveness specification* is: if smoke was detected by some process, then the application eventually reports to the fire department. Thus, the application uses consensus as a building block to provide interesting higher-level behavior.

This paper exploits the fact that numerous consensus protocols already have verified implementations [52, 58, 72], and targets parameterized synthesis of distributed applications built on top of such verified consensus implementations. Our overall approach is based on *abstracting consensus into a primitive building block* and reasoning about the behavior of the rest of the application around it.

As our first contribution, we introduce a simple abstraction for consensus protocols that can be integrated into models of distributed systems. In this abstraction, the processes trying to reach consensus simply move from a global state where consensus would naturally start to a global state where consensus is achieved, using a special *atomic* global transition, denoted `choose`. Such `choose` transitions encapsulate the *implementation details* of how consensus is achieved and simply provide a consistency guarantee: all processes that participate in the consensus round agree on the participating and the "chosen" processes. Additionally, we define the `choose` model: an expressive model of distributed protocols that permits `choose` transitions to abstract consensus rounds, accommodates broadcast and rendezvous communication, and lends itself to symmetry-based reductions for verification.

To synthesize distributed applications described in the `choose` model, this paper makes further contributions: *parameterized verification and synthesis* for systems in the `choose` model with an *arbitrary* number of processes. We note that parameterized verification is only decidable for restricted classes of distributed systems. For example, Esparza et al. [34] showed that for (instantaneous) broadcast communication in a clique, safety properties are decidable, while liveness properties are not. Unfortunately, neither this decidability result, nor any other result we are aware of, is applicable to systems in the `choose` model.

Our second contribution, then, is a decidability result for parameterized verification w.r.t. safety properties in an extension of the broadcast model of Esparza et al. [34]. This extended model, called the *guarded broadcast model*, is obtained by augmenting broadcast protocols with additional global conditions necessary for modeling consensus. We show that we can map systems in our `choose` model, that satisfy some conditions, to the guarded broadcast model, while preserving safety properties. All these results can be integrated to yield a semi-decision procedure for parameterized synthesis of systems in the `choose` model, based on a parameterized verifier.

To avoid the practical challenges of parameterized verification, we make our third contribution: we derive *cutoffs* for parameterized verification for many interesting classes of systems in the guarded broadcast model and characterize the conditions under which the cutoffs hold for parameterized verification of systems in the `choose` model. These cutoffs enable reduction of parameterized verification to verification using a standard, *non-parameterized* verification engine.

All these contributions put together yield an algorithm for parameterized synthesis of systems in the `choose` model (under certain conditions) based on a standard, non-parameterized CEGIS (counterexample-guided inductive synthesis) loop [64].

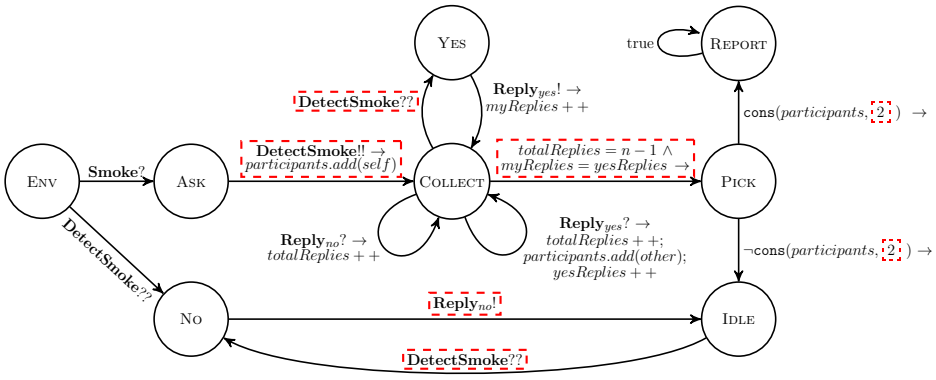

Fig. 1. A smoke detector process.

We build our abstractions and procedures on top of Kinara, a recent verification and synthesis engine for distributed systems [3, 4, 2]. We demonstrate the utility of our `choose` abstraction and cutoffs by first showing that several sensor network-like applications can be verified and synthesized by our tool. We then turn to two larger case studies: (i) the Small Aircraft Tracking System (SATS), a protocol for managing air traffic that requires several rounds of consensus between distributed processes (planes), and (ii) Distributed Mobile Robotics (DMR) that requires several rounds of synchronization to ensure that a set of distributed mobile robots generate action plans that do not interfere. We show that given a skeleton of the system we can synthesize guards that use multiple rounds of consensus to guarantee the correct operation of these systems.

Example. Fig. 1 shows a finite-state input-output machine describing a smoke detector process from our distributed smoke detector example from before. If there are n smoke detectors, there will be n instances of this state machine. The state machine refers to the current instance using the variable *self*. Each smoke detector starts in state Env. If any detector receives the message **Smoke**? from the environment, it moves to Ask, and broadcasts **DetectSmoke**!! to all other processes. Any process still in Env when it receives the broadcast (**DetectSmoke**??) moves to No, where it responds to any messages saying it did not see smoke (**Reply**$_{no}$!). At this point, all processes that *have* detected smoke collaborate to determine who will report the incident. The loop between Collect and Yes involves receiving **Reply** messages from other processes and responding to other **DetectSmoke** messages (with **Reply**$_{yes}$!) such that each detector builds a set *participants* that captures all detectors that saw smoke (those that did not see smoke respond from the No state and will not be in *participants*).

Once a process has built its participant set and has responded to all other detectors that saw smoke (*i.e.* $totalReplies = n-1 \wedge myReplies = yesReplies$), the detectors that detected smoke move to Pick to `choose` at most two detectors to contact the fire department. This is the step that abstracts consensus: all detectors in Pick will receive a consistent view of which detectors were chosen,

reflecting the results of a consensus protocol. Hence, two detectors will contact the fire department, and all the others move to the IDLE state.

Systems of this type can be verified using our framework, and also *synthesized* to ensure the safety properties mentioned earlier. In our experiments, we omitted the guards and updates in the dashed red boxes and were able to synthesize the correct completions to satisfy the specification. The omitted parts correspond to interesting questions the developer of such a system might have. For example, what should a detector do upon receiving a message from another detector? After exchanging the messages, should the detector attempt to contact the fire department?

2 Preliminaries: Distributed Protocol Completion

We present an adaptation of the formal model of distributed protocols and the counterexample-guided inductive synthesis (CEGIS) procedure for protocol completion from [3, 4].

2.1 Formal Model

Basic Model of Distributed Systems. We consider distributed systems consisting of a set of n identical *system processes* and an *environment process*, all communicating asynchronously. We fix a collection of finite types, including the type `bool` of Boolean values and the special type $\mathcal{I} = \{1, 2, \ldots, n\}$ of process indices. We also permit enumerated types, arrays and records. Processes communicate by synchronizing on input and output actions. We support two types of communication: pairwise rendezvous and broadcast. In (pairwise) rendezvous communication, two processes synchronize to communicate a value and in broadcast communication, one process synchronizes with all other processes to communicate a value[3].

Processes. Formally, a process P is a finite-state, input-output machine described by the tuple $\langle I, O, L, \ell_0, V, T\rangle$ where I and O are disjoint, finite sets of input and output actions, respectively, L is a finite set of locations, $\ell_0 \in L$ is the initial location, V is a set of (typed) process-local variables and T is a set of transitions. Each transition in T is denoted by $\ell \xrightarrow{a,\, g \to u} \ell'$ such that:

1. $\ell, \ell' \in L$ are the source and target locations, respectively,
2. $a \in I \cup O \cup \{\epsilon\}$ is the (*communicating*) action
3. $\langle g, u\rangle$ (or, $g \to u$) is a guarded command where the guard g is a Boolean expression and the command u is a sequence of updates of the form `lhs`:=`rhs`. In general, `lhs` is a local variable and g, `rhs` are expressions over local variables. If a is an input action, g and `rhs` may include a. If a is an output action, `lhs` may update a.

[3] Note that our current model supports *asynchronous* operation (non-communicating processes can take steps independent of one another and different processes can communicate at different times), but not *non-blocking* communication (sending and receiving processes must block until they can exchange the message).

An action $a \in O$ ($a \in I$, resp.) is either a broadcast output (input) message $\texttt{msg}!!v$ ($\texttt{msg}??v$, resp.) or a rendezvous output (input) message $\texttt{msg}!v$ ($\texttt{msg}?v$, resp.), where $\texttt{msg}$ is the message label and v is the value communicated for some associated variable var. A transition with $a = \epsilon$ is called an *internal* transition and is not observable by other processes.

A *state* s of process P is a pair (ℓ, σ) where $\ell \in L$ and σ is a valuation of the local variables in V. Let S be the set of all states of process P. Let σ_0 denote the initial valuation and $s = (\ell_0, \sigma_0)$ denote the initial state of P. An execution/run τ of process P is a possibly infinite sequence $(\ell_0, \sigma_0), (\ell_1, \sigma_1), \ldots$ of states in S obtained by *executing enabled transitions*: for each $j \geq 0$, there is a transition $\ell_j \xrightarrow{a,\, g \to u} \ell_{j+1} \in T$ such that σ_j satisfies the guard g and σ_{j+1} is obtained by applying the update u to σ_j.

A system process is required to satisfy the following conditions.

1. T is deterministic: at most one transition is enabled at every state along any execution.
2. There is no race between input and output transitions: the set of locations L is partitioned into the set L_I of *input locations* and the set L_O of output locations. All outgoing transitions from L_I and L_O are input/internal and output/internal transitions, respectively.

Unless necessary, we refer to a system process simply as a process. Note that an environment process is not required to satisfy any of the above conditions. In particular, it can be non-deterministic.

Composition of Processes. We define the asynchronous (interleaving-based) composition $P_1 \parallel \ldots \parallel P_n$ of n identical [4] processes $P_1, \ldots, P_n$ as a global transition system[5], $\mathcal{M} = \langle Q, q_0, R \rangle$, where

1. $Q = S^n$ is the set of global states,
2. $q_0 = (s_{0,1}, \ldots, s_{0,n})$ is the initial global state, and,
3. $R \subseteq Q \times Q$ is the set of global transitions (corresponding to two processes synchronizing via rendezvous communication, or, all processes synchronizing via broadcast communication, or, a single process making an internal transition). For rendezvous communication, this can be formalized as follows. If there exist processes P_i, P_j with local transitions $\ell_i \xrightarrow{\texttt{msg}!v,\, g_i \to u_i} \ell_i'$ and $\ell_j \xrightarrow{\texttt{msg}?v,\, g_j \to u_j} \ell_j'$, respectively, such that $g_i \wedge g_j$ evaluates to $\texttt{true}$ in global state q, and global state q' is obtained from q by updating the locations of P_i, P_j to ℓ_i', ℓ_j', respectively and applying the sequence of updates u_i; $var{:=}v$; u_j, then there exists a corresponding global transition (q, q') in R. Global transitions corresponding to broadcast communication and internal transitions can be formalized similarly.

An execution/run of the global transition system $\mathcal{M}$ is a (possibly infinite) sequence of states, $q_0, q_1, \ldots$, in Q such that for each $j \geq 0$, $(q_j, q_{j+1}) \in R$. A state

[4] We compose with the environment process, too, but elide this for simplicity.

[5] Ideally, the composition should be defined as an input-output process to ensure associativity of the composition operator. We choose to keep the presentation simple by instead defining the composition as an (unlabeled) transition system.

q is reachable if there exists a finite execution of $\mathcal{M}$ that reaches q. A state q is a *deadlock* state if no transition is enabled at q. A global transition system that does not contain any deadlock states is called *deadlock-free*.

Specifications. The expected behavior of (executions of) the global transition system $\mathcal{M}$ is specified using *safety* and *liveness* requirements in *linear temporal logic (LTL)* [53]. The global transition system can be augmented with error states $E \subseteq Q$ and accepting states $A \subseteq Q$ to capture violations of safety and liveness specifications, respectively. An infinite execution of $\mathcal{M}$ is *accepting* if it visits accepting states in A infinitely often. The system $\mathcal{M}$ is *safe* if it has no reachable error states and *live* if it has no accepting infinite execution. Given a safety (liveness) specification, the notation $\mathcal{M} \models \phi$ denotes that $\mathcal{M}$ is safe (live).

Symmetry-based Reduction. Since $\mathcal{M}$ is constructed by composing identical processes, it is possible to greatly improve its verification time complexity by *collapsing* symmetric behaviors in $\mathcal{M}$. We say $\mathcal{M}$ is *fully symmetric* if its transition relation R is invariant under permutations over the set of process indices $\mathcal{I}$: $\forall \pi \in G, \pi(R) = R$, where G is the set of all permutations over $\mathcal{I}$ and $\pi(R) = \{(\pi(q_1), \pi(q_2)) : (q_1, q_2) \in R\}$. Appendix A.1 formalizes the notion of permuting global states (i.e., defines $\pi(q)$) and presents syntactic conditions on the guarded commands in $\mathcal{M}$ that entail full symmetry. It has been shown in [29] that, under full symmetry, one can "reduce" $\mathcal{M}$ into a quotient structure $\overline{\mathcal{M}}$ such that for any specification ϕ: $\mathcal{M} \models \phi \iff \overline{\mathcal{M}} \models \phi$.

2.2 Distributed Protocol Completion

Problem Definition. A *process sketch* $P^{??}$ is a partial process with incomplete guarded commands. In particular, expressions in guarded commands in a process sketch may also include function symbols with unknown interpretations, along with local variables, actions and function symbols with known interpretations. We assume that each known or unknown function symbol is equipped with a signature $d_1 \times \ldots \times d_k \to r$ identifying the types of its arguments and return values. Given an interpretation A that assigns an appropriate interpretation to each uninterpreted function symbol in $P^{??}$, we obtain a process P that we call a *completion* of $P^{??}$ under A.

Definition 1 (Distributed Protocol Completion). *Given an environment process E with no uninterpreted functions and a set of identical process sketches $P_1^{??}, \ldots, P_n^{??}$ with sets of uninterpreted functions $U_1, \ldots, U_n$, find an interpretation A of $U = U_1 \cup \ldots \cup U_n$ that yields completions $P_1, \ldots, P_n$ such that:*

1. *$P_1, \ldots, P_n$ satisfy the conditions for system processes and*
2. *the global transition system $\mathcal{M} = P_1 \parallel \ldots \parallel P_n \parallel E$ is safe, live, deadlock-free and fully symmetric.*

Synthesis algorithm. We present a simplified version of the distributed protocol completion algorithm from [3]. The CEGIS algorithm iteratively invokes a *synthesizer* and a *verifier*. The synthesizer attempts to generate an interpretation A for all uninterpreted functions while enforcing conditions (1) and (2)

from Def. 1. If the synthesizer succeeds, then the verifier attempts to check if the completed protocol is safe, live and deadlock-free. If verification succeeds, the algorithm terminates. Otherwise, the verifier returns a set of counterexamples that are used to generate further constraints on the uninterpreted functions.

The synthesizer is based on an SMT solver and the verifier is a model checker that uses symmetry reduction.

3 Parameterized Completion of Distributed Applications with Consensus

We extend the formal model in Sec. 2.1 to enable us to effectively reason about distributed applications based on consensus protocols and present the parameterized completion problem for such systems.

3.1 The `choose` Model for Distributed Applications with Consensus

Many distributed applications are built on top of complex consensus protocols. An explicit encoding of such consensus protocols within a larger application can lead to *state space explosion* and prevent reasoning about higher-level properties of the application. To address this problem, we assume correctness of the consensus protocol and extend the basic model to allow encapsulation of such protocols into an *abstraction* that comes with precondition obligations and postcondition guarantees. Then, to prove correctness of distributed applications with consensus protocols it is sufficient to show correctness of their *simpler* counterparts in this extended model, called the `choose` model.

Consensus protocols enable their participants to reach agreement on a set of values in a distributed setting. Many different strategies exist to implement this [48, 10, 56, 62]. However, at its core, consensus boils down to *choosing* a subset of *winners* (or winning proposals) from the set of *participants* in a way that is globally consistent. Breaking this down further, any correct consensus protocol implementation satisfies the following precondition and postcondition.

C1 *Consistent Participants Precondition.* The consensus round starts when all participating processes agree on *who* to reach consensus with.

C2 *Consistent Winners Postcondition.* When the consensus round ends, each participant's local result is globally consistent with all other participants.

Thus, to encapsulate consensus rounds, the `choose` model extends the basic model with a special *atomic* global transition, denoted `choose`, that comes with a guarantee of expected behavior, expressed as the precondition-postcondition pair $(C1, C2)$. We make this more precise in what follows.

Let us fix a round of consensus with participants[6] $\mathcal{S} \subseteq \mathcal{I}$ and cardinality k that chooses some set of k winners from $\mathcal{S}$. Let $\mathcal{W}^*$ denote the set of all possible winner sets. We model this consensus round with a nondeterministic,

[6] While consensus protocols naturally agree on values, we abstract this using process identifiers without loss of generality.

atomic global `choose` transition from a `GlobalConsensusStart` state $q_{\texttt{start}}$ that satisfies precondition C1 to `GlobalConsensusEnd` states $q^{\mathcal{W}}_{\texttt{end}}$ (corresponding to winner sets $\mathcal{W} \in \mathcal{W}^*$) that satisfy postcondition C2. Let C denote the local state of each participant in which the consensus protocol is invoked. Let $\mathcal{S}_i$ denote the local set of consensus participants constructed by process P_i before it transitions into C[7]. Finally, let C^W (C^L) denote the local state into which each winning (losing) participant transitions after the consensus round ends.

The `GlobalConsensusStart` state, $q_{\texttt{start}}$, is defined as a global state where all the participants of a consensus round are in C *and* have consistent $\mathcal{S}_i$ sets:
(1) $\exists i \in \mathcal{I} :\ q_{\texttt{start}}[i] = C$,
(2) $\forall i, j \in \mathcal{I} :\ q_{\texttt{start}}[i] = C \wedge \mathbb{1}_{\mathcal{S}_i}(j) \iff q_{\texttt{start}}[j] = C \wedge \mathbb{1}_{\mathcal{S}_j}(i)$, and
(3) $\forall i, j \in \mathcal{I} :\ (q_{\texttt{start}}[i] = C \wedge q_{\texttt{start}}[j] = C) \implies (\mathcal{S}_i = \mathcal{S}_j)$
where $q[i]$ denotes the local state of process P_i in global state q and $\mathbb{1}_A(x)$ is an indicator function that evaluates to `true` if $x \in A$ and `false` otherwise. Since each process maintains the set of participants locally, these conditions enforce consistency between those sets. Condition (1) checks that at least one process is in C. This filters out all the states where no process is attempting to invoke consensus, and helps trigger conditions (2) and (3) on the right states. Condition (2) ensures that if two processes are in C, each process is in the other's local set of participants. Condition (3) ensures that if two processes are in C, they must agree on their sets of participants. Note that if a process is done constructing its set of participants and is now in C, it must wait until every other process (in its set of participants) has transitioned to C.

The `GlobalConsensusEnd` state $q^{\mathcal{W}}_{\texttt{end}}$ for winner set $\mathcal{W} \in \mathcal{W}^*$ is defined as a global state satisfying the following:
(1) $\forall i \in \mathcal{W} :\ q^{\mathcal{W}}_{\texttt{end}}[i] = C^W$,
(2) $\forall i \in \mathcal{S} - \mathcal{W} :\ q^{\mathcal{W}}_{\texttt{end}}[i] = C^L$, and
(3) $\forall i \in \mathcal{I} - \mathcal{S} : q^{\mathcal{W}}_{\texttt{end}}[i] = q_{\texttt{start}}[i]$.
This definition ensures that all processes locally transition into states that consistenly reflect the results of the consensus round.

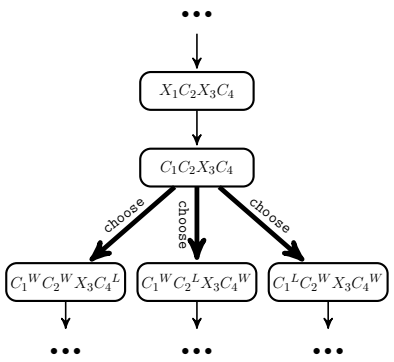

Fig. 2. Part of the global transition system in the `choose` model encapsulating one round of consensus.

Example. Consider a consensus round in a distributed system with 4 processes, where processes 1, 2, and 4 participate to choose two winners and process 3 is not interested. Thus, the set $\mathcal{S}$ of participants is $\{1, 2, 4\}$ and the set $\mathcal{W}^*$ of all possible winner sets is $\{\{1, 2\}, \{1, 4\}, \{2, 4\}\}$. The behavior of the global transition system in

[7] We place no restrictions on how these sets are constructed. In some situations, constructing the set may require exchanging messages with other processes to determine which ones are involved in consensus (e.g. Fig. 1); in others, such sets can be predefined or locally inferred with no need to exchange messages to construct them.

the `choose` model, relevant to this consensus round, is shown in Fig. 2. The `choose` transition abstracting this consensus round originates in the `GlobalConsensusStart` state $C_1C_2X_3C_4$; here, process 3 is in a *don't care* local state X. The `choose` transition nondeterministically ends in one of three possible `GlobalConsensusEnd` states, each encoding one of the winner sets. Note that the `choose` transition cannot originate in the state $X_1C_2X_3C_4$ as it does not satisfy the *Consistent Participants* precondition.

Symmetry-based Reduction. In Appendix A.2, we show that a global transition system $\mathcal{M}$ in the `choose` model is *fully symmetric*. Specifically, we show that a `choose` transition (and hence, the transition relation R) is invariant under a set of permutations G over the process indices $\mathcal{I}$, i.e., $\forall q_{\texttt{start}}, q_{\texttt{end}} \in Q, \pi \in G : (q_{\texttt{start}}, q_{\texttt{end}}) \in R \iff (\pi(q_{\texttt{start}}), \pi(q_{\texttt{end}})) \in R$, where $q_{\texttt{start}}$ and $q_{\texttt{end}}$ are the start and end states of some `choose` transition. Intuitively, the proof is based on the observation that `choose` transitions are oblivious to the *actual* identities of the participants.

Remarks. Note that a global, atomic `choose` transition encapsulates a bunch of interleaved messages and states within a consensus protocol and hence has no clear mapping to process-local transitions. Also, to keep the presentation simple, we assumed one consensus round. Our framework and tool support multiple consensus rounds, each encapsulated in its own global `choose` transition(s).

3.2 Parameterized Completion of Distributed Applications

Problem Definition. The parameterized completion problem for distributed applications with consensus differs from the distributed protocol completion problem in Def. 1 in two respects. First, our global transition systems are now defined in the `choose` model. Second, we target correctness of distributed systems with an *arbitrary* number of processes, i.e., an arbitrarily-sized set $\mathcal{I}$ of process indices. Thus, we effectively aim to synthesize a *family* of correct distributed systems, parameterized by $\mathcal{I}$, also referred to as a *parameterized system*. In contrast, distributed protocol completion targets synthesis of a single, correct distributed system with a *fixed* number of processes.

To be able to define the parameterized completion problem, we need to extend some of our existing definitions with their parameterized versions.

A *parameterized uninterpreted function* is an uninterpreted function that is parameterized by $\mathcal{I}$. An interpretation $A_{\mathcal{I}}$ for a parameterized uninterpreted function is defined over $\mathcal{I}$ and is called a *parameterized interpretation*. Given a specific set of process indices, $\mathcal{J}$, and a parameterized interpretation $A_{\mathcal{I}}$ for the parameterized uninterpreted functions of a process sketch $P^{??}$, we can obtain completed processes $P_1, \ldots, P_{|\mathcal{J}|}$ under the instantiated interpretation $A_{\mathcal{I}}(\mathcal{J})$.

Definition 2 (Parameterized Completion of Distributed Applications). *Given an environment process E with no uninterpreted functions, and a process sketch $P^{??}$ with a set $U_{\mathcal{I}}$ of parameterized uninterpreted functions, find a parameterized interpretation $A_{\mathcal{I}}$ of $U_{\mathcal{I}}$ such that: for any given set of process indices $\mathcal{J}$ if $P_1, \ldots, P_{|\mathcal{J}|}$ are the completed processes under $A_{\mathcal{I}}(\mathcal{J})$, then*

1. *$P_1, \ldots, P_{|\mathcal{J}|}$ satisfy the condition for system processes and*
2. *the global transition system $\mathcal{M} = P_1 \parallel \ldots \parallel P_{|\mathcal{J}|} \parallel E$ in the* `choose` *model is safe, live, deadlock-free and fully symmetric.*

4 Solution Sketch

In this section, we describe our solution to the problem of parameterized completion of distributed applications, combining an algorithm for distributed protocol completion with techniques for obtaining parameterized safety guarantees.

The first ingredient of our solution is an algorithm for synthesis of distributed applications in the `choose` model, for a fixed number of processes. To this end, we have extended the CEGIS algorithm of [3, 4] to support `choose` transitions.

To lift this synthesis technique to parameterized systems, we need a way to reason about the *parameterized verification* problem that checks correctness of a parameterized system. Since none of the existing decidability results for parameterized verification support the `choose` model, we extend an existing model to support `choose` while still admitting decidable parameterized verification.

Guarded Broadcast Model. The model of broadcast protocols of Esparza et al. [34], which supports broadcasts, internal transitions and rendezvous, admits decidable parameterized verification of safety properties. However, the transitions cannot communicate values, in particular process indices, and the model does not support processes with a state space that grows with the number of processes (as is the case in the `choose` model). Moreover, it assumes that broadcasts are always enabled, whereas both the basic and `choose` models have blocking broadcasts that are only enabled if all other processes are ready to receive. Therefore, this model cannot be used for paramaterized verification of systems in the basic model, much less the `choose` model.

Our extension of the model, which we call the *guarded broadcast (GBC) model*, also cannot communicate values and requires a fixed state space of processes, but it supports global guards on transitions that require every process to be in a certain subset of its local state space. Thus, it directly supports blocking broadcasts, and also allows us to capture the conditions of a `GlobalConsensusStart` state and the effect of a `choose` transition at a slightly more abstract level: the guard allows us to check the *Consistent Participants* precondition (by implicitly encoding that all processes have common knowledge of the participants), and a `choose` transition is modeled as a GBC transition that moves one of the participants to the winning state C^W and all other processes to the losing state C^L.

Formally, a process in the GBC model, referred to as a *GBC protocol*, is defined as a tuple $\langle A, S, s_0, T \rangle$, where A is a finite set of action names that gives rise to the input actions $In = \{a?? \mid a \in A\}$ and output actions $Out = \{a!! \mid a \in A\}$, S is a finite set of local states, $s_0 \in S$ is the initial state, and

$T \subseteq S \times In \cup Out \times \mathcal{P}(S) \times S$ is the transition relation.[8] Given a transition (s, l, G, s'), we call l the *label* and G the *guard* of the transition. Semantically, a local transition $(s, a!!, G, s')$ can only be taken if all processes are currently in a state in G; if the transition is taken, all other processes have to take a transition labeled with $a??$. We assume that exactly one such transition exists for all states in G, and its guard is S, i.e., it can always be taken. As further extensions, we also allow *k-broadcasts*, i.e., transitions with k broadcast senders, for $k \in \mathbb{N}$, which allows us to directly model `choose` transitions with cardinality k. Finally, we introduce *negotiations*, a communication primitive involving *matching broadcast transitions* $(s, a!!, G, s')$ and $(s, a??, S, s')$. Thus, a negotiation can be initiated by any participating process, a beneficial property that general broadcasts do not enjoy. A global state is represented as a vector $\mathbf{q} \in \mathbb{N}^{|S|}$. It stores, for every local state s, the number of processes currently in s, denoted $\mathbf{q}(s)$.

To prove decidability of parameterized verification for this model, we use similar machinery as Esparza et al. [34]. One main difference is that we need a more fine-grained well-quasi-ordering (wqo) on global states in order to ensure that only states that satisfy the same guards are related by the ordering. Similar to their proof, we can show that in most cases a transition in a small system can be simulated by a single corresponding transition in any system that is bigger (in the wqo). However, there is one special case in which multiple transitions in the bigger system may be required to simulate a single transition in the small system. To ensure that a repetition of transitions is possible, we require the following *well-behavedness* property.

Definition 3 (Well-behaved GBC protocol). *A GBC protocol is defined to be well-behaved if: for every pair of transitions $(s, a!!, G, s')$ with $s' \neq s$ and $(s, a??, S, s)$ in the system, and for any state $s_r \in G$ with a receiving transition $(s_r, a??, S, s'_r)$, the post-state s'_r is in G and has a self-loop $(s'_r, a??, S, s'_r)$.*

Note that the well-behavedness requirement is quite natural: it essentially requires processes to be insensitive to repetitions of the same broadcast action. Since processes have a finite state-space, unbounded counting of the number of identical actions is impossible anyway. Moreover, note that GBC protocols that correspond to a system in the `choose` model (as described later in this section) are *always* well-behaved, since separation between input and output locations implies that pairs of transitions as defined above cannot exist.

We can now state our key result about decidability of parameterized verification in the GBC model (see Appendix B.3 for the proof):

Theorem 1. *For well-behaved GBC protocols, the parameterized verification problem for safety properties is decidable.*

The decidability result is constructive, i.e., it yields an algorithm for parameterized verification. Furthermore, an analysis of this parameterized verification algorithm enables computation of *cutoffs* for different classes of GBC protocols.

[8] Without loss of generality, we assume that all actions are broadcasts, since this allows us to simulate internal transitions and rendezvous.

Formally, a cutoff for a class of parameterized systems and a class of specifications is a number $c \in \mathbb{N}$ such that for every process P and specification φ from the respective classes,

$$\forall n \geq c : (P_1 \parallel \ldots \parallel P_c \models \varphi \iff P_1 \parallel \ldots \parallel P_n \models \varphi) .$$

Thus, cutoffs directly reduce parameterized verification to verification of a c-process system, where c is the cutoff for the system and specification under consideration.

While one can show that, in general, cutoffs for GBC protocols can grow impractically large, we determine sufficient conditions that guarantee small cutoffs. We define a local transition to be *free* if it is an internal transition, a broadcast send transition, or a negotiation. A path from one state to another is free if all transitions on the path are free. Suppose we want to check reachability of an error state $\mathbf{q}$ with $\mathbf{q}(s) = m$. A sufficient condition for obtaining small cutoffs, abbreviated as $\mathsf{cond}^{s}_{\mathsf{free}}$, is that all paths from s_0 to s in the GBC protocol must be free.

Lemma 1. *Given a well-behaved GBC protocol that satisfies* $\mathsf{cond}^{s}_{\mathsf{free}}$*,* $c = m$ *is a cutoff for reachability of an error state* $\mathbf{q}$ *with* $\mathbf{q}(s) = m$*.*

While the condition may seem restrictive, it directly applies to four out of our five example benchmarks, and with slight modifications also to the fifth example. For more details on cutoffs, we refer the reader to Appendix B.4.

Relation between `choose` model and GBC model. To support an abstraction into the GBC model, a system $\mathcal{M}_{\texttt{choose}}$ in the `choose` model needs to satisfy certain conditions. For this work, we assume that (i) any communication of process indexes in $\mathcal{M}_{\texttt{choose}}$ is confined to a substructure of the state-space that is only used for establishing common knowledge about distributed state, e.g., to ensure the *Consistent Participants* precondition; (ii) the values of variables that store process indexes have no further effect after leaving the substructure; and (iii) our safety properties do not refer to the internals of such a substructure. We abbreviate these three conditions as $\mathsf{cond}_{\mathsf{abs}}$ and obtain the following lemma.

Lemma 2. *Under* $\mathsf{cond}_{\mathsf{abs}}$*, one can map an arbitrary-sized system* $\mathcal{M}_{\texttt{choose}}$ *in the* `choose` *model to a GBC protocol* $\mathcal{M}_{GBC}$ *such that for any* $\mathsf{cond}_{\mathsf{abs}}$*-compliant safety property* ϕ*:*

$$\forall n. (\mathcal{M}_{GBC}(n) \models \phi \iff \mathcal{M}_{\texttt{choose}}(n) \models \phi) .$$

To support this abstraction in our synthesis approach, we need to guarantee that the synthesized solution will satisfy $\mathsf{cond}_{\mathsf{abs}}$. This is, in part, enforced by the process sketches we consider, and by preventing the completion from re-using process indexes outside of the substructure. For more details on the relation between the two models, we refer to Appendix B.5.

Parameterized completion of distributed applications in the `choose` model. The parameterized verifier obtained as a result of Theorem 1 can be used in conjunction with the distributed protocol completion algorithm from Sec. 2, to yield a semi-decision procedure for parameterized completion of systems in the `choose` model for ensuring safety. To avoid the practical challenge of implementing a parameterized verifier, we present an alternative solution based on cutoff results for the GBC model.

The main requirements that we need to guarantee are that (i) condition $\mathsf{cond}_{\mathsf{abs}}$ holds, enabling the abstraction of a completed system $\mathcal{M}_{\texttt{choose}}$ to a GBC protocol $\mathcal{M}_{GBC}$ for which we know how to compute cutoffs, and (ii) condition $\mathsf{cond}^s_{\mathsf{free}}$ holds for any state s that appears in an error state $\mathbf{q}$, enabling the use of Lemma 6 to compute a cutoff to ensure safety. Since the abstraction preserves safety properties for systems of arbitrary size (Lemma 2), satisfaction of these two conditions implies that we can synthesize a solution of the cutoff size in the `choose` model, and directly get parameterized correctness.

5 Implementation

We implemented our approach in Kinara [3, 4, 2], a finite-state model checking and synthesis framework. We briefly highlight some details of how we extend Kinara to support systems in the `choose` model; we call this new system Kinara$_{ch}$.

Kinara$_{ch}$ ensures that the given process sketch is completed in a way that respects $\mathsf{cond}_{\mathsf{abs}}$ and $\mathsf{cond}^s_{\mathsf{free}}$ (see Sec. 4). Kinara$_{ch}$ encodes these conditions as constraints to the SMT solver that exclude possible completions if they result in a violation of the conditions.

Kinara$_{ch}$ models the participant set used in `GlobalConsensusStart` as an array that is indexed by the index set $\mathcal{I}$ with values `true` or `false` to indicate membership. For example, in a system of 4 processes, the input set $\{1, 3\}$ is represented by the array `[false,true,false,true]`.

To ensure that we generate truly parameterized completions that work for an arbitrary number of processes, we impose additional syntactic constraints on the generated guards and updates. For example, we do not allow uninterpreted functions to evaluate to integer constants. Instead, the completions have to use terms that depend on the size of the index set, typically $|\mathcal{I}|$ or $|\mathcal{I}| - 1$ to refer to all (other) processes in the system. Similarly, completions can check for equality of two indices, but cannot compare individual indices against constant numbers.

As discussed in Sec. 3.1, each `choose` transition encodes a possible winning set by annotating the participants as winners and losers (C^L and C^W). Kinara$_{ch}$ stores that annotation in a variable, `ChDes`, which can be 0, 1, or 2, representing unknown result, losing process, or winning process, respectively. A `choose` transition sets `ChDes` on each participating process in a globally-consistent way. Kinara$_{ch}$ creates a variant of this transition for each possible outcome of `choose`.

Example. Assume we have 4 smoke detectors modeled as shown in Fig. 1. A global state where only 3 detectors (say 1, 2, and 4) have detected fire, communicated the information among each other, and reached their PICK local state is a

valid `GlobalConsensusStart` state. With a cardinality of 2, a `choose` transition that encodes the winning set $\{1, 4\}$ would have the following updates:
`global transition { true -> Detector[1].ChDes := 2 ;`
`Detector[2].ChDes := 1 ; Detector[4].ChDes := 2}`
Similar commands are added and executed for alternative winning sets $\{1, 2\}$ and $\{2, 4\}$. From a process-local perspective, the local guard `cons` $(S, 2)$ (shown in Fig. 1) is implemented as `ChDes = 2?` while the local guard $\neg$`cons` $(S, 2)$ is implemented as `ChDes = 1?`. Both transitions also set `ChDes` back to 0.

Note that a process participating in consensus round cannot advance beyond a local choose state if its `ChDes` is set to 0. And since `ChDes` can only be set to 1 or 2 by a `choose` transition, which is only triggered out of a proper `GlobalConsensusStart` state, all participating processes will block until they have the results on consensus, and then they can proceed in an interleaving semantics style—this captures the atomic nature of our consensus abstraction.

6 Evaluation

This section evaluates Kinara$_{ch}$, on several case studies. First, we study whether Kinara$_{ch}$ can model, verify, and complete (synthesize) different distributed applications in the `choose` model. Second, we use the cutoffs established in Lemma 1 to perform synthesis for these applications and evaluate Kinara$_{ch}$'s performance.

6.1 Case Study Descriptions

We use the `choose` model to build and verify five case studies: the example from the introduction, as well as four more, which we describe here. For each case study, we describe the application and delegate the safety and liveness properties for the problem and its state machine description to Appendix C.

Chubby Application. Chubby [7] is a distributed lock service. Applications can interface with Chubby as a file system where they send reads and writes to the Chubby system, and the data is replicated safely on different servers. The Chubby system starts by picking a *leader* server which is responsible for receiving clients' requests. The leader can safely serve read requests directly and is responsible for committing writes to all the replicas before sending an acknowledgment to the client. The leader periodically times out, and a new round of election should happen to pick a new leader. Chubby's use of consensus is quite simple, but it demonstrates our broader point, which is that consensus is not an application in and of itself. Rather, it is a building block that can be leveraged for more complex applications (serving files, in this case).

Smoke Detectors with Reset. We next examine a variant of the smoke detector example that uses a "reset" environment signal to move all detectors back to the initial state to start new rounds of detection. The presence of the reset signal makes the problem more interesting as it simulates infinite rounds of consensus.

Distributed Mobile Robotics (DMRs). As a larger case study, we model the system presented in [19] where a set of robots share a workspace with obstacles,

Benchmark	#Local Vars	#Locations	Cutoff	#Iterations	Time(s)
Chubby application	3	9	2	106	4.6
Smoke detector (SD)	9	9	3	57	406.4
SD with reset	9	9	3	53	380.2
DMR	9	11	3	52	358.4
SATS	17	14	5	6	306.1

Table 1. Performance numbers for the benchmarks.

and need to coordinate their movements. The robots coordinate to create a motion plan by successively choosing each robot to make a motion plan taking into account the previous robots' plans. We model this system in our `choose` model: the robots choose one robot to make a plan, then the remaining robots re-enter consensus to choose a second robot and so on.

Small Aircraft Transportation System (SATS). Our final case study addresses the landing operation of the Small Aircraft Transportation System (SATS) [1], a frequent target for verification [42, 8, 55]. The idea is that the aircraft should coordinate with each other to figure out how to land safely and avoid collisions. The planes use consensus to choose successive subsets of planes to progress to the next phase of landing, until just one plane at a time is chosen to land.

6.2 Quantitative Evaluation of Kinara$_{ch}$

Table 1 quantitatively evaluates Kinara$_{ch}$'s synthesis performance. For each benchmark, we give the number of local variables in the process definition, the number of locations, the cutoff (number of processes) used in synthesis, the number of CEGIS iterations before the loop terminated with the correct completions, and finally the total execution time. We conducted the experiments on an Intel core i7 machine with 4GB RAM. The "Smoke detector" row in the table is for the case study in Fig. 1. We see in all cases that Kinara$_{ch}$ is able to perform synthesis quite quickly—taking less than seven minutes to perform synthesis for any example. Note that synthesis time is dependent on how complex the holes are, the size of the local state space, and the number of constraints that must be solved in each iteration. In all cases, we leverage the cutoffs provided by Appendix B.4, which allow us to perform synthesis only for a small number of processes while providing guarantees for unbounded numbers.

7 Related Work

Non-parameterized verification and synthesis. There is a tremendous amount of work on checking whether a distributed system with a *fixed* number of processes meets a given set of correctness specifications [4, 3, 2, 68, 16–18, 14, 19, 52, 73, 13]. Fundamentally, these approaches are not sufficient since they only provide guarantees for fixed size systems.

Parameterized Verification. The Parameterized Model Checking Problem (PMCP) instead tries to solve the verification problem for systems of any size. Works that

tackle PMCP (which is undecidable in the most general case [66, 27]) take a variety of different approaches.

One main approach to tackle the undecidability of the PMCP and the state space explosion is to include the user in the loop with *interactive theorem provers*. Such tools describe the distributed system is either logic [71, 61, 70, 20, 72, 59] or a DSL [60]. Typically, the user needs to specify an inductive invariant that is used to make an argument about the correctness of the system for any size. Disel [61] leverages the same observation we do—that many distributed applications build on lower-level primitives like consensus—and builds abstractions of distributed primitives to provide compositional verification using Coq. However, such approaches require a non-trivial knowledge of the system and the underlying logic to provide the inductive invariant and use the tools properly.

Another attempt to address the PMCP problem with user help is using *deductive reasoning*. For example [58, 57, 67] use deductive techniques like rewire rules and quantifier elimination in order to map the distributed systems written in uninterpreted first-order logic (FOL) into a decidable fragment of FOL. Most reasoning in this track depends heavily on the specific distributed system being addressed (e.g. [22, 54, 58] for consensus and [51] for key-value stores). Hence, is hard to generalize and automate.

Several works attempt to identify cutoff bounds for different classes of distributed systems. For example, cutoffs were obtained for cache coherence protocols [24], guarded protocols [40, 25, 23], consensus protocols [54], and self-stabilizing systems [5]. PSync [22] is a specialized programming language for the development of verifiable consensus algorithms, but the user has to provide invariants and ranking functions. Maric et al. [54] show cutoffs directly for consensus problems. However, their results are restricted to the consensus problem itself, and do not extend to applications that use consensus to obtain other goals.

A final class of solutions attempts to find decidability results by restricting the generality of the problem in various ways. Most results consider a fully connected network (a clique), either with rendezvous communication [37], broadcasts [34], local updates with global guards [23], or several variants or combinations thereof [26, 24]. Some communication primitives have also been considered in more complex networks, for example token-passing [28, 11], or broadcasts [15]. Decidability results for systems that are composed of identical components have recently been surveyed by Bloem et al. [6] as well as Esparza [32]. Works based on Threshold Automata [45, 43, 41, 47, 44, 50, 65, 46] build on the intuition that most fault-tolerance strategies depend on some form of counting replies and checking if they are above some threshold.

Parameterized Synthesis. There has recently also been an increased interest in synthesizing parameterized systems. Jacobs and Bloem [39] introduced a general approach based on cutoff results, which allows to use any underlying synthesis algorithm for the given class of systems. Other approaches are more specialized, such the approach of Lazic et al. [50] for synthesis of fault-tolerant algorithms that takes a sketch of the algorithm and generates parameter valuations that guarantee correctness by solving a set of arithmetic constraints.

Verifying Consensus Protocols. Due to their importance, several works have targeted reasoning about consensus protocols like Paxos [48, 10] and Raft [56]. Some implemented DSLs to reason about consensus [22, 54, 52], others mapped consensus protocols to decidable logic fragments [21, 58], while others used interactive tools to reason about consensus [72]. This reinforces the need to move from reasoning about the correctness of consensus protocols, and instead focus on applications that use consensus as a black-box component.

References

1. Nasa - small aircraft transportation system, https://www.nasa.gov/centers/langley/news/factsheets/SATS.html
2. Alur, R., Martin, M., Raghothaman, M., Stergiou, C., Tripakis, S., Udupa, A.: Synthesizing finite-state protocols from scenarios and requirements. In: Haifa Verification Conference. pp. 75–91. Springer (2014)
3. Alur, R., Raghothaman, M., Stergiou, C., Tripakis, S., Udupa, A.: Automatic completion of distributed protocols with symmetry. In: International Conference on Computer Aided Verification. pp. 395–412. Springer (2015)
4. Alur, R., Tripakis, S.: Automatic synthesis of distributed protocols. SIGACT News **48**(1), 55–90 (Mar 2017). https://doi.org/10.1145/3061640.3061652, http://doi.acm.org/10.1145/3061640.3061652
5. Bloem, R., Braud-Santoni, N., Jacobs, S.: Synthesis of self-stabilising and byzantine-resilient distributed systems. In: International Conference on Computer Aided Verification. pp. 157–176. Springer (2016)
6. Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers (2015). https://doi.org/10.2200/S00658ED1V01Y201508DCT013, https://doi.org/10.2200/S00658ED1V01Y201508DCT013
7. Burrows, M.: The chubby lock service for loosely-coupled distributed systems. In: Proceedings of the 7th symposium on Operating systems design and implementation. pp. 335–350. USENIX Association (2006)
8. Carreño, V., Muñoz, C.: Safety verification of the small aircraft transportation system concept of operations. In: AIAA 5th ATIO and16th Lighter-Than-Air Sys Tech. and Balloon Systems Conferences. p. 7423 (2005)
9. Chand, S., Liu, Y.A., Stoller, S.D.: Formal verification of multi-paxos for distributed consensus. In: International Symposium on Formal Methods. pp. 119–136. Springer (2016)
10. Chandra, T.D., Griesemer, R., Redstone, J.: Paxos made live: an engineering perspective. In: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing. pp. 398–407. ACM (2007)
11. Clarke, E.M., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: Gardner, P., Yoshida, N. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 3170, pp. 276–291. Springer (2004). https://doi.org/10.1007/978-3-540-28644-8_18, https://doi.org/10.1007/978-3-540-28644-8_18
12. Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., Vanzetto, H.: Tla+ proofs. In: International Symposium on Formal Methods. pp. 147–154. Springer (2012)

13. Damm, W., Finkbeiner, B.: Automatic compositional synthesis of distributed systems. In: International Symposium on Formal Methods. pp. 179–193. Springer (2014)
14. Deligiannis, P., McCutchen, M., Thomson, P., Chen, S., Donaldson, A.F., Erickson, J., Huang, C., Lal, A., Mudduluru, R., Qadeer, S., Schulte, W.: Uncovering bugs in distributed storage systems during testing (not in production!). In: Proceedings of the 14th Usenix Conference on File and Storage Technologies. pp. 249–262. FAST'16, USENIX Association, Berkeley, CA, USA (2016), http://dl.acm.org/citation.cfm?id=2930583.2930602
15. Delzanno, G., Sangnier, A., Traverso, R., Zavattaro, G.: On the complexity of parameterized reachability in reconfigurable broadcast networks. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India. LIPIcs, vol. 18, pp. 289–300. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012). https://doi.org/10.4230/LIPIcs.FSTTCS.2012.289, https://doi.org/10.4230/LIPIcs.FSTTCS.2012.289
16. Desai, A., Gupta, V., Jackson, E., Qadeer, S., Rajamani, S., Zufferey, D.: P: safe asynchronous event-driven programming. ACM SIGPLAN Notices **48**(6), 321–332 (2013)
17. Desai, A., Jackson, E., Phanishayee, A., Qadeer, S., Seshia, S.A.: Building reliable distributed systems with p. Tech. rep. (2015)
18. Desai, A., Phanishayee, A., Qadeer, S., Seshia, S.A.: Compositional programming and testing of dynamic distributed systems. Proc. ACM Program. Lang. **2**(OOPSLA), 159:1–159:30 (Oct 2018). https://doi.org/10.1145/3276529, http://doi.acm.org/10.1145/3276529
19. Desai, A., Saha, I., Yang, J., Qadeer, S., Seshia, S.A.: Drona: A framework for safe distributed mobile robotics. In: Proceedings of the 8th International Conference on Cyber-Physical Systems. pp. 239–248. ICCPS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3055004.3055022, http://doi.acm.org/10.1145/3055004.3055022
20. Doenges, R., Wilcox, J.R., Woos, D., Tatlock, Z., Palmskog, K.: Verification of implementations of distributed systems under churn
21. Drăgoi, C., Henzinger, T.A., Veith, H., Widder, J., Zufferey, D.: A logic-based framework for verifying consensus algorithms. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 161–181. Springer (2014)
22. Drăgoi, C., Henzinger, T.A., Zufferey, D.: Psync: A partially synchronous language for fault-tolerant distributed algorithms. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 400–415. POPL '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2837614.2837650, http://doi.acm.org/10.1145/2837614.2837650
23. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: McAllester, D.A. (ed.) CADE. Lecture Notes in Computer Science, vol. 1831, pp. 236–254. Springer (2000). https://doi.org/10.1007/10721959_19, https://doi.org/10.1007/10721959_19
24. Emerson, E.A., Kahlon, V.: Exact and efficient verification of parameterized cache coherenceprotocols. In: CHARME. Lecture Notes in Computer Science, vol. 2860, pp. 247–262. Springer (2003). https://doi.org/10.1007/978-3-540-39724-3_22

25. Emerson, E.A., Kahlon, V.: Model checking guarded protocols. In: 18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings. pp. 361–370. IEEE Computer Society (2003). https://doi.org/10.1109/LICS.2003.1210076, https://doi.org/10.1109/LICS.2003.1210076
26. Emerson, E.A., Kahlon, V.: Rapid parameterized model checking of snoopy cache coherence protocols. In: TACAS. Lecture Notes in Computer Science, vol. 2619, pp. 144–159. Springer (2003). https://doi.org/10.1007/3-540-36577-X_11
27. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 85–94. POPL '95, ACM, New York, NY, USA (1995). https://doi.org/10.1145/199448.199468, http://doi.acm.org/10.1145/199448.199468
28. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Int. J. Found. Comput. Sci. **14**(4), 527–550 (2003). https://doi.org/10.1142/S0129054103001881, https://doi.org/10.1142/S0129054103001881
29. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Formal methods in system design **9**(1-2), 105–131 (1996)
30. Emerson, E.A., Wahl, T.: On combining symmetry reduction and symbolic representation for efficient model checking. In: Advanced Research Working Conference on Correct Hardware Design and Verification Methods. pp. 216–230. Springer (2003)
31. Emerson, E.A., Wahl, T.: Dynamic symmetry reduction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 382–396. Springer (2005)
32. Esparza, J.: Parameterized verification of crowds of anonymous processes. In: Esparza, J., Grumberg, O., Sickert, S. (eds.) Dependable Software Systems Engineering, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 45, pp. 59–71. IOS Press (2016). https://doi.org/10.3233/978-1-61499-627-9-59, https://doi.org/10.3233/978-1-61499-627-9-59
33. Esparza, J., Desel, J.: On negotiation as concurrency primitive. In: D'Argenio, P.R., Melgratti, H.C. (eds.) CONCUR. Lecture Notes in Computer Science, vol. 8052, pp. 440–454. Springer (2013). https://doi.org/10.1007/978-3-642-40184-8_31, https://doi.org/10.1007/978-3-642-40184-8_31
34. Esparza, J., Finkel, A., Mayr, R.: On the verification of broadcast protocols. In: 14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999. pp. 352–359. IEEE Computer Society (1999). https://doi.org/10.1109/LICS.1999.782630, https://doi.org/10.1109/LICS.1999.782630
35. Finkel, A.: A generalization of the procedure of karp and miller to well structured transition systems. In: ICALP. Lecture Notes in Computer Science, vol. 267, pp. 499–508. Springer (1987). https://doi.org/10.1007/3-540-18088-5_43
36. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theor. Comput. Sci. **256**(1-2), 63–92 (2001). https://doi.org/10.1016/S0304-3975(00)00102-X
37. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681, http://doi.acm.org/10.1145/146637.146681
38. Ip, C.N., Dill, D.L.: Better verification through symmetry. Formal methods in system design **9**(1-2), 41–75 (1996)

39. Jacobs, S., Bloem, R.: Parameterized synthesis. Logical Methods in Computer Science **10**(1) (2014). https://doi.org/10.2168/LMCS-10(1:12)2014, https://doi.org/10.2168/LMCS-10(1:12)2014
40. Jacobs, S., Sakr, M.: Analyzing guarded protocols: Better cutoffs, more systems, more expressivity. In: International Conference on Verification, Model Checking, and Abstract Interpretation. pp. 247–268. Springer (2018)
41. John, A., Konnov, I., Schmid, U., Veith, H., Widder, J.: Towards modeling and model checking fault-tolerant distributed algorithms. In: International SPIN Workshop on Model Checking of Software. pp. 209–226. Springer (2013)
42. Johnson, T.T., Mitra, S.: Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In: Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on. pp. 161–170. IEEE (2012)
43. Konnov, I., Lazić, M., Veith, H., Widder, J.: Para2: parameterized path reduction, acceleration, and smt for reachability in threshold-guarded distributed algorithms. Formal Methods in System Design **51**(2), 270–307 (2017)
44. Konnov, I., Lazić, M., Veith, H., Widder, J.: A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. ACM SIGPLAN Notices **52**(1), 719–734 (2017)
45. Konnov, I., Veith, H., Widder, J.: On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. vol. 252 (09 2014). https://doi.org/10.1007/978-3-662-44584-6_10
46. Konnov, I., Veith, H., Widder, J.: Smt and por beat counter abstraction: parameterized model checking of threshold-based distributed algorithms. In: International Conference on Computer Aided Verification. pp. 85–102. Springer (2015)
47. Konnov, I., Widder, J.: ByMC: Byzantine Model Checker. In: ISoLA 2018 - 8th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Lecture Notes in Computer Science, vol. 11246, pp. 327–342. Limassol, Cyprus (Oct 2018). https://doi.org/10.1007/978-3-030-03424-5_22, https://hal.inria.fr/hal-01909653
48. Lamport, L.: The part-time parliament. ACM Transactions on Computer Systems (TOCS) **16**(2), 133–169 (1998)
49. Lamport, L.: Specifying systems: the TLA+ language and tools for hardware and software engineers. Addison-Wesley Longman Publishing Co., Inc. (2002)
50. Lazic, M., Konnov, I., Widder, J., Bloem, R.: Synthesis of distributed algorithms with parameterized threshold guards. In: Aspnes, J., Bessani, A., Felber, P., Leitão, J. (eds.) OPODIS. LIPIcs, vol. 95, pp. 32:1–32:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017). https://doi.org/10.4230/LIPIcs.OPODIS.2017.32, https://doi.org/10.4230/LIPIcs.OPODIS.2017.32
51. Lesani, M., Bell, C.J., Chlipala, A.: Chapar: Certified causally consistent distributed key-value stores. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 357–370. POPL '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2837614.2837622, http://doi.acm.org/10.1145/2837614.2837622
52. Liu, Y.A., Stoller, S.D., Lin, B., Gorbovitski, M.: From clarity to efficiency for distributed algorithms. In: ACM SIGPLAN Notices. vol. 47, pp. 395–410. ACM (2012)
53. Manna, Z., Pnueli, A.: The temporal logic of reactive and concurrent systems: Specification. Springer Science & Business Media (2012)

54. Marić, O., Sprenger, C., Basin, D.: Cutoff bounds for consensus algorithms. In: International Conference on Computer Aided Verification. pp. 217–237. Springer (2017)
55. Munoz, C.A., Dowek, G., Carreño, V.: Modeling and verification of an air traffic concept of operations. ACM SIGSOFT Software Engineering Notes **29**(4), 175–182 (2004)
56. Ongaro, D., Ousterhout, J.K.: In search of an understandable consensus algorithm. In: USENIX Annual Technical Conference. pp. 305–319 (2014)
57. Padon, O., Hoenicke, J., Losa, G., Podelski, A., Sagiv, M., Shoham, S.: Reducing liveness to safety in first-order logic. Proc. ACM Program. Lang. **2**(POPL), 26:1–26:33 (Dec 2017). https://doi.org/10.1145/3158114, http://doi.acm.org/10.1145/3158114
58. Padon, O., Losa, G., Sagiv, M., Shoham, S.: Paxos made epr: Decidable reasoning about distributed protocols. Proc. ACM Program. Lang. **1**(OOPSLA), 108:1–108:31 (Oct 2017). https://doi.org/10.1145/3140568, http://doi.acm.org/10.1145/3140568
59. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. In: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 614–630. PLDI '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2908080.2908118, http://doi.acm.org/10.1145/2908080.2908118
60. Rahli, V.: Interfacing with proof assistants for domain specific programming using eventml (2012)
61. Sergey, I., Wilcox, J.R., Tatlock, Z.: Programming and proving with distributed protocols. Proc. ACM Program. Lang. **2**(POPL), 28:1–28:30 (Dec 2017). https://doi.org/10.1145/3158116, http://doi.acm.org/10.1145/3158116
62. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Symposium on Self-Stabilizing Systems. pp. 386–400. Springer (2011)
63. Sistla, A.P., Gyuris, V., Emerson, E.A.: Smc: a symmetry-based model checker for verification of safety and liveness properties. ACM Transactions on Software Engineering and Methodology (TOSEM) **9**(2), 133–166 (2000)
64. Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. In: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 404–415. ASPLOS XII, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1168857.1168907, http://doi.acm.org/10.1145/1168857.1168907
65. Stoilkovska, I., Konnov, I., Widder, J., Zuleger, F.: Verifying Safety of Synchronous Fault-Tolerant Algorithms by Bounded Model Checking (Nov 2018), https://hal.inria.fr/hal-01925653, working paper or preprint
66. Suzuki, I.: Proving properties of a ring of finite-state machines. Inf. Process. Lett. **28**(4), 213–214 (Jul 1988). https://doi.org/10.1016/0020-0190(88)90211-6, http://dx.doi.org/10.1016/0020-0190(88)90211-6
67. Taube, M., Losa, G., McMillan, K.L., Padon, O., Sagiv, M., Shoham, S., Wilcox, J.R., Woos, D.: Modularity for decidability of deductive verification with applications to distributed systems. In: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 662–677. PLDI 2018, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3192366.3192414, http://doi.acm.org/10.1145/3192366.3192414

68. Udupa, A., Raghavan, A., Deshmukh, J.V., Mador-Haim, S., Martin, M.M., Alur, R.: Transit: specifying protocols with concolic snippets. ACM SIGPLAN Notices **48**(6), 287–296 (2013)
69. Wahl, T.: Adaptive symmetry reduction. In: International Conference on Computer Aided Verification. pp. 393–405. Springer (2007)
70. Wilcox, J.R., Sergey, I., Tatlock, Z.: Programming Language Abstractions for Modularly Verified Distributed Systems. In: Lerner, B.S., Bodík, R., Krishnamurthi, S. (eds.) 2nd Summit on Advances in Programming Languages (SNAPL 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 71, pp. 19:1–19:12. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2017). https://doi.org/10.4230/LIPIcs.SNAPL.2017.19, http://drops.dagstuhl.de/opus/volltexte/2017/7126
71. Wilcox, J.R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M.D., Anderson, T.: Verdi: A framework for implementing and formally verifying distributed systems. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 357–368. PLDI '15, ACM, New York, NY, USA (2015). https://doi.org/10.1145/2737924.2737958, http://doi.acm.org/10.1145/2737924.2737958
72. Woos, D., Wilcox, J.R., Anton, S., Tatlock, Z., Ernst, M.D., Anderson, T.: Planning for change in a formal verification of the raft consensus protocol. In: Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs. pp. 154–165. CPP 2016, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2854065.2854081, http://doi.acm.org/10.1145/2854065.2854081
73. Yang, J., Chen, T., Wu, M., Xu, Z., Liu, X., Lin, H., Yang, M., Long, F., Zhang, L., Zhou, L.: Modist: Transparent model checking of unmodified distributed systems. In: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation. pp. 213–228. NSDI'09, USENIX Association, Berkeley, CA, USA (2009), http://dl.acm.org/citation.cfm?id=1558977.1558992

A Symmetry Reduction

In order to improve scalability of verification and synthesis of systems with many similar processes, symmetry reductions are used to reduce a global transition system $\mathcal{M}$ into a smaller transition system $\overline{\mathcal{M}}$ that preserves relevant behavior [29, 38, 3, 63, 30, 69, 31]. In this section, we first formalize the notion of symmetry and identify conditions for enabling symmetry reduction in the basic model of Sec. 2.1. We then show that the `choose` model of Sec. 3.1 also lends itself to symmetry reduction.

A.1 Symmetry Reduction in the Basic Model

Full Symmetry. Let $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be a permutation acting on the set $\mathcal{I}$ of process indices. Let G denote the set of all permutations over $\mathcal{I}$. A permutation of a global state $q = (s_0, s_1, \ldots, s_n)$ can then be defined as: $\pi(q) = (\pi(s_0), \pi(s_1), \ldots, \pi(s_n))$, where $\pi(s_i) = (\ell_{\pi(i)}, \pi(\sigma_i))$. Note that $\pi(\sigma_i)$ depends on the type of the local variable being permuted: if it is of type $\mathcal{I}$, then $\pi(\sigma_i) = \sigma_{\pi(i)}$, otherwise $\pi(\sigma_i) = \sigma_i$ [9].

Definition 4 ([30, 69, 31]). *A global transition system $\mathcal{M}$ composed of identical processes with index set $\mathcal{I}$ is fully symmetric if its transition relation R is invariant to permutations: $\forall \pi \in G : \pi(R) = R$, where $\pi(R) = \{(\pi(q_1), \pi(q_2)) : (q_1, q_2) \in R\}$.*

To enable the use of symmetry reduction in the verification of our distributed systems, we require the guarded commands of processes to be *index-independent* [29, 30]: the value of the guard and the effect of the update should not depend on the specific values of process indices, i.e., should be invariant to permutation of process indices.

Definition 5. *Given a distributed system consisting of identical processes with index set $\mathcal{I}$, a guard or an update f in a process is index-independent if $\forall \pi \in G : \pi(f) \iff f$.*

Lemma 3 ([29, 30, 38]). *For a distributed system consisting of identical processes with index set $\mathcal{I}$, if the guards and updates in all processes are index-independent, then the global transition system $\mathcal{M}$ is fully symmetric.*

Sufficient syntactic constraints for index-independence. While guards and updates over variables independent of $\mathcal{I}$ are naturally index-independent (since permutations only affect $\mathcal{I}$), those over variables of type $\mathcal{I}$ must be proven to be invariant under permutations. Prior approaches [29, 30, 38] propose easy-to-check *syntactic constraints* on expressions over variables of type $\mathcal{I}$ that suffice for index-independence. These constraints restrict guards/updates over variables

[9] In case the variable was of an enumerated type (e.g. set, array, or record) containing values of type $\mathcal{I}$, then the permutation is applied recursively to all the elements. If the array has an index type $\mathcal{I}$, then we permute the array elements themselves, too.

of type $\mathcal{I}$ to expressions checking equality (or disequality). For example, given local variables $i_1, i_2 \in V$ of type $\mathcal{I}$, these syntactic constraints permit guards that check if $i_i = i_2$ or $i_1 \neq i_2$, but do not permit guards that check $i_1 < i_2$. In the latter case, the guard values will differ if the values of i_1 and i_2 are permuted.

Symmetry Reduction via Quotient Structures. As proposed by Emerson and Sistla [29], it is possible to exploit the symmetries present in a global transition system $\mathcal{M}$ in model checking by constructing a compressed *quotient* structure $\overline{\mathcal{M}}$ such that $\mathcal{M} \models \phi \iff \overline{\mathcal{M}} \models \phi$, where ϕ is any (CTL*) specification. It follows from their result that such a quotient structure can be constructed for any $\mathcal{M}$ that is fully symmetric and can enable symmetry reduction for model checking w.r.t. any LTL specification. We refer the interested reader to [29] for further details.

A.2 Symmetry Reduction in the `choose` Model

Let $\mathcal{M} = (Q, q_0, R)$ be a global transition system in the basic model, modeling a distributed application with (possibly multiple rounds of) consensus. Let $\mathcal{M}_{\texttt{choose}} = (Q, q_0, R')$ be the global transition system in the `choose` model corresponding to $\mathcal{M}$, obtained by abstracting away multiple interleaved paths between `GlobalConsensusStart` and `GlobalConsensusEnd` states in Q with (nondeterministic and atomic) `choose` transitions. Thus, $R' \setminus R$, denoted R_{ch}, is a set of `choose` transitions of the form $(q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}})$, where $q_{\texttt{start}}$ is a `GlobalConsensusStart` state and $q^{\mathcal{W}}_{\texttt{end}}$ is a possible `GlobalConsensusEnd` state where $\mathcal{W} \in \mathcal{W}^*$.

Further, let us assume that all guards and updates of process-local transitions involved in the global transitions in $R \cap R_{\texttt{choose}}$ are index-independent. It then follows from the definitions and lemmas outlined in Appendix A.1 that if $R_{\texttt{choose}}$ is invariant under permutations over $\mathcal{I}$, $\mathcal{M}_{\texttt{choose}}$ is fully symmetric and thereby enables symmetry reduction.

Lemma 4. *R_{choose} is invariant under permutations over $\mathcal{I}$:*

$$\forall \pi \in G, (q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}}) \in R_{choose} : (q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}}) \in R_{choose} \iff (\pi(q_{\texttt{start}}), \pi(q^{\mathcal{W}}_{\texttt{end}})) \in R_{choose}$$

Proof. Let $q_1, q_2, \ldots, q_{\texttt{start}}$ be a path in $\mathcal{M}_{\texttt{choose}}$ with no `choose` transitions. By constraints placed on such transitions (ref. Lemma 3) we know that if state $q_{\texttt{start}}$ is reachable, then all states $\{\pi(q_{\texttt{start}}) | \pi \in G\}$ are reachable, too.

Let us examine one $q_{\texttt{start}}$ and its possible $q_{\texttt{end}}$ states, then generalize to other $q_{\texttt{start}}$ states. Recall that we characterize such $q_{\texttt{start}}$ state by the set of participants it encodes, say $\mathcal{S}$, and the cardinality of the intended winning sets, say k. For each of the resulting winning set $\mathcal{W} \in \mathcal{W}^*$, we create a $q^{\mathcal{W}}_{\texttt{end}}$ state.

We distinguish two disjoint subsets of permutations in G:
- $G_m = \{\pi \mid \pi(\mathcal{S}) = \mathcal{S}\}$[10] (i.e. permutations preserving the membership of $\mathcal{S}$).
- $G_n = \{\pi \mid \pi(\mathcal{S}) \neq \mathcal{S}\}$.

[10] The permutation of a set A is the set of the permuted elements: $\pi(A) = \{\pi(m) \mid m \in A\}$.

By the construction process of `choose` transitions (ref. Sec. 3.1), we know that $\mathcal{W}^*$ contains *all subsets* of $\mathcal{S}$ of size k. Hence, $\mathcal{W}^*$ is invariant to permutations in G_m (i.e. $\forall \pi_m \in G_m,\ \pi_m(\mathcal{W}^*) = \mathcal{W}^*$). Since $\mathcal{S}$ is also invariant to permutations in G_m, we can lift those observations to states:

$$\forall \pi_m \in G_m,\ (q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}}) \in R_{\texttt{choose}} \iff (\pi_m(q_{\texttt{start}}), \pi_m(q^{\mathcal{W}}_{\texttt{end}})) \in R_{\texttt{choose}}$$

For any permutation $\pi_n \in G_n$ we obtain a new global state $q'_{\texttt{start}} = \pi_n(q_{\texttt{start}})$ encoding the set $\mathcal{S}' = \pi_n(\mathcal{S})$. However, since any such $q'_{\texttt{start}}$ is reachable, and by the above reasoning, we conclude that:

$$\forall \pi_n \in G_n,\ (q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}}) \in R_{\texttt{choose}} \iff (\pi_n(q_{\texttt{start}}), \pi_n(q^{\mathcal{W}}_{\texttt{end}})) \in R_{\texttt{choose}}$$

Since $G = G_m \cup G_n$, we obtain:

$$\forall \pi \in G, (q_{\texttt{start}}, q^{\mathcal{W}}_{\texttt{end}}) \in R_{\texttt{choose}} \iff (\pi(q_{\texttt{start}}), \pi(q^{\mathcal{W}}_{\texttt{end}})) \in R_{\texttt{choose}}$$

Paths to $q_{\texttt{start}}$ with `choose` transitions. Above, we assumed that no `choose` transitions is on the path to $q_{\texttt{start}}$. Recall that we `choose` transitions are *atomic*. So one can inductively argue that the above reasoning holds as we, effectively, just showed that `choose` transitions are symmetry-compliant.

B Parameterized Correctness Arguments and Cutoffs

In this section, we show how to obtain parameterized correctness arguments for our systems. To this end, we provide a model-checking procedure based on a counter representation, as well as a way to obtain cutoffs. We use a system model that is at a slightly higher level than the one in the previous sections, and is based on the model for broadcast protocols by Esparza [34]. After introducing this basic model, we extend it such that we can model the essential properties of `choose` and certain kinds of transition guards. Then, we provide conditions for decidability of the PMCP of these systems, as well as a model-checking procedure and a way to derive cutoffs.

B.1 Basic System Model: Broadcast Protocols

We consider processes $P = \langle A, S, s_0, T \rangle$, where A is a finite set of action names that gives rise to the input actions $In = \{a?? \mid a \in A\}$ and output actions $Out = \{a!! \mid a \in A\}$, S is a finite set of states, $s_0 \in S$ is the initial state, and $T \subseteq S \times In \cup Out \times S$ is the transition relation[11]. Without loss of generality, we assume that all actions are broadcasts, since this allows us to simulate local

[11] In contrast to Sec. 2.1, we do not separate a state into a location and a valuation of variables in this model, and processes do not have identifiers.

transitions, pairwise rendezvous and global synchronization. The semantics of broadcasts is: if some process takes a transition with $a!!$, then all other processes have to take a transition with $a??$ (if they have one in the current state; we can assume wlog that this is always the case, by adding self-loops). This part is exactly the model from Esparza et al. [34].

Formally, the parameterized model of Esparza et al. is the following:

- the state space of the parameterized system is $\mathbf{q} = \mathbb{N}^{|S|}$, i.e., a state is a vector of natural numbers, representing the number of processes that are in any given state $s \in S$.
- in a broadcast transition, one process makes a transition $s_i \xrightarrow{a!!} s_j$ and all other processes make a transition $s_k \xrightarrow{a??} s_l$, depending on their current state s_k. Thus, the successor configuration $\mathbf{q}' \in \mathbb{N}^{|S|}$ of a given configuration $\mathbf{q} \in \mathbb{N}^{|S|}$ can be computed in the following way (where $\mathbf{u}_i$ is the unit vector with $\mathbf{u}_i(s_i) = 1$):

$$\begin{aligned} \mathbf{p} &= \mathbf{q} - \mathbf{u}_i \\ \mathbf{p}'(s_l) &= \sum\nolimits_{\{s_k \mid s_k \xrightarrow{a??} s_l\}} \mathbf{p}(s_k) \\ \mathbf{q}' &= \mathbf{p}' + \mathbf{u}_j. \end{aligned}$$

  As noted by Esparza et al., the computation of $\mathbf{q}'$ can be denoted more succinctly as $\mathbf{q}' = M_a \cdot \mathbf{q} + \mathbf{v}_a$, where M_a is a *broadcast matrix* that models the second line above and $\mathbf{v}_a$ is a *broadcast vector* that models the effect of the other two lines. Under the assumption of deterministic transitions, in a broadcast matrix each column is a unit vector.

Our goal is to use broadcasts to model `choose` transitions on a slightly more abstract level. While the effect of a `choose`(1) transition can be modeled by a broadcast (where the chosen process is the sender and all other processes are receivers), we face two problems: (i) `choose` transitions are only enabled when *all* processes have decided whether they want to participate, and (ii) broadcasts do not allow us to model the effect of a `choose`(k) with $k > 1$. Therefore, we need to significantly extend the broadcast model in order to capture the semantics of the systems we consider.

B.2 Extended Model: Guarded Broadcast Protocols

In the basic model, a broadcast transition a is enabled whenever there is a process that can take a local transition labeled with $a!!$ for sending the broadcast. I.e., this model implicitly assumes that all processes are always ready to receive the broadcast. To model `choose` transitions that are only enabled when all processes have made the decision whether they want to participate, we extend this model by adding support for *guarded* transitions that are only enabled if all processes are ready, i.e., they are in a specified subset $G \subseteq S$ of their local statespace. G is also called the *guard* of the transition. Moreover, to model `choose` transitions that select $k > 1$ winners, we extend the model with transitions that are similar to a broadcast, but have up to k processes execute the sending transition at the

same time. Finally, we also consider *negotiations*, which are essentially broadcasts without a distinguished sender, i.e., transitions where all processes (that are in a state where the transition is enabled) synchronize and take the same action.[12]

Formally, our systems are now given by $\langle A, S, s_0, T\rangle$, where all components are as before, except that now we have $T \subseteq S \times In \cup Out \times \mathcal{P}(S) \times S$. The update of global state $\mathbf{q}$ by a transition is defined as before, except that

- a transition is only enabled if $\mathbf{q}$ satisfies the guard G, i.e., if $\mathbf{q}(s) = 0$ for all $s \notin G$. We write such a transition as $\mathbf{q} \xrightarrow{G} \mathbf{p}$.
- a broadcast transition may allow a fixed number $k \in \mathbb{N}$ of processes to take the sending transition simultaneously. We call such a transition a *k-broadcast*. In this case, the global successor configuration is computed by subtracting $k \cdot \mathbf{u}_i$ (instead of $\mathbf{u}_i$) in the first step, and adding $k \cdot \mathbf{u}_j$ (instead of $\mathbf{u}_j$) in the last step. If there are only $k' < k$ processes in the sending state, then the transition fires with k' senders instead of k.
- a negotiation step is computed simply by applying a broadcast matrix M_a, without an additional broadcast vector.

B.3 Model Checking for Guarded Broadcast Protocols

Finkel [35] introduced the notion of a well-structured transition system (WSTS): an infinite-state transition system that is equipped with a *well-quasi order (WQO)* on its state space and that has some additional properties. We will use Finkel and Schnoebelen [36] as a main reference, which gives a survey of existing results and puts them into a common framework. In particular, these results show how we can effectively represent systems with infinite sets of states in order to decide safety/reachability properties.

In this section we show that, under some additional conditions, guarded broadcast protocols are WSTSs, and we can decide the parameterized model checking problem (PMCP) for safety properties.

Compatibility and Effective Computability of Predecessors To prove decidability of the PMCP, we need some additional definitions.

Definition 6. *A WQO $\preceq$ is* compatible *with a transition relation $\to$ if for every $\mathbf{q} \preceq \mathbf{q}'$ and $\mathbf{q} \to \mathbf{p}$ there exists $\mathbf{p}'$ with $\mathbf{p} \preceq \mathbf{p}'$ and $\mathbf{q}' \to^* \mathbf{p}'$, and* strongly compatible *if $\mathbf{q}' \to \mathbf{p}'$.*

A transition system equipped with a WQO that is compatible with its transition relation is called a *well-structured transition system (WSTS)*.

Fix an infinite set of states Q and a transition relation $\to$. For a (possibly infinite) subset $U \subseteq Q$, let $Pred(U)$ denote the predecessor states of U with respect to $\to$. Furthermore, let $\uparrow U$ denote the *upwards closure* of U, i.e., the set of all states $\mathbf{p}$ such that there exists $\mathbf{q} \in U$ with $\mathbf{q} \preceq \mathbf{p}$. A set U is *upwards closed* if $\uparrow U = U$. Every upwards closed set U has a finite *basis*: a finite set $B \subseteq U$ such that $\uparrow B = U$.

[12] Systems that only communicate through negotiation have been investigated by Esparza and Desel [33].

Definition 7. *For a given transition relation $\to$ and WQO $\preceq$, we say that we can* effectively compute *$Pred$ if there exists an algorithm that computes a finite basis of $Pred(U)$ from any finite basis of any upwards-closed $U \subseteq Q$.*

Finkel and Schnoebelen [36] have shown that we can decide reachability of any upwards-closed set in a WSTS if we can effectively compute $Pred$.

Deciding Safety Properties for Guarded Broadcast Protocols. Esparza et al. [34] have defined the following WQO on the (global) states of the basic model of broadcast protocols:

$$\mathbf{q} \preceq \mathbf{q}' \text{ iff } \mathbf{q}(s) \leq \mathbf{q}'(s) \text{ for all } s \in S.$$

Based on this, they have shown:

Theorem 2 ([34]). *For the model of broadcast protocols and the WQO above, the following hold:*

- *Broadcast transitions are* strongly compatible *with the WQO, i.e., if $\mathbf{q} \preceq \mathbf{q}'$ and $\mathbf{q} \to \mathbf{p}$, then $\exists \mathbf{p}'$ with $\mathbf{p} \preceq \mathbf{p}'$ and $\mathbf{q}' \to \mathbf{p}'$. This implies that for a given upward-closed set of states C, the set of predecessor states $Pred(C)$ is also upward-closed.*[13]
- *For broadcast transition systems we can effectively compute Pred.*

We quickly recapitulate their proof idea, because it will inform our proof for guarded broadcast protocols.

Proof. For both items, it is sufficient to show them for every action $a \in A$ separately. With our formulation, the first item is easy to show based on the matrix notation of broadcast transitions. For the second, observe that for a transition based on $s_k \xrightarrow{a!!} s_l$, any element $\mathbf{q}$ of $Pred(C)$ must satisfy $\mathbf{q}(s_k) \geq 1$ and $M_a \cdot \mathbf{q} + \mathbf{v}_a = \mathbf{q}'$, for some $\mathbf{q}' \in C$. The basis of $Pred(C)$ consists of the minimal elements that satisfy these conditions, and thus is computable.

In the following, we consider the extended model of guarded broadcast protocols and introduce conditions under which we can obtain a similar result.

To this end, for a global state $\mathbf{q}$ let $\mathsf{supp}(\mathbf{q}) = \{s \in S \mid \mathbf{q}(s) > 0\}$, i.e., the set of local states that appear at least once in $\mathbf{q}$. Then we consider the following WQO:[14]

[13] In fact, Esparza et al. have only proven the latter, but one can show that the two statements are equivalent.

[14] To see that this is really a WQO, we need to show that every infinite sequence $\mathbf{q}_1, \mathbf{q}_2, \ldots$ of global states contains contains $\mathbf{q}_i, \mathbf{q}_j$ with $i < j$ and $\mathbf{q}_i \trianglelefteq \mathbf{q}_j$. To see this, consider an arbitrary infinite sequence $\overline{\mathbf{q}} = \mathbf{q}_1, \mathbf{q}_2, \ldots$. Then there is at least one set S of local states such that there are infinitely many $\mathbf{q}_i$ with $\mathsf{supp}(\mathbf{q}_i) = S$. Then let $\overline{\mathbf{q}'}$ be the infinite subsequence of $\overline{conf}$ where all elements have $\mathsf{supp}(\mathbf{q}'_i) = S$. Since $\preceq$ is a WQO, there exist $\mathbf{q}'_i, \mathbf{q}'_j$ with $i < j$ and $\mathbf{q}'_i \preceq \mathbf{q}'_j$, and since $\mathsf{supp}(\mathbf{q}'_i) = \mathsf{supp}(\mathbf{q}'_j) = S$, we also get $\mathbf{q}'_i \trianglelefteq \mathbf{q}'j$. Since $\mathbf{q}'_i = \mathbf{q}_k$ and $\mathbf{q}'_j = \mathbf{q}_l$ for some $k < l$, we get $\mathbf{q}_k \trianglelefteq \mathbf{q}_l$ for $k < l$, and thus $\trianglelefteq$ is a WQO.

$$\mathbf{q} \trianglelefteq \mathbf{q}' \text{ iff } \mathbf{q} \preceq \mathbf{q}' \wedge \mathsf{supp}(\mathbf{q}) = \mathsf{supp}(\mathbf{q}').$$

Furthermore, we require the following restriction on guarded broadcast protocols. We call a guarded broadcast protocol *well-behaved* if every broadcast transition $\mathbf{q} \xrightarrow{G} \mathbf{p}$ based on a local transition $s_k \xrightarrow{a!!} s_l$ of the sender and a matrix M_a for the receivers satisfies at least one of the following:

1. the sender remains in its state, i.e., $s_k = s_l$,
2. receivers that are in s_k leave the state, i.e., $M_a(s_k) \neq s_k$, or
3. for all states in G, M_a is idempotent, i.e., $M_a(s) = M_a(M_a(s))$, and all states are mapped into G by the transition.

Note that these are not strong restrictions, and these conditions are naturally satisfied by many classes of transitions. E.g., the second condition is satisfied by `choose` transitions, the third condition is necessary (and sufficient) to model internal transitions as guarded broadcasts, and a negotiation transition will always satisfy one of the first two conditions.

Then, we get the following:

Theorem 3. *For well-behaved guarded broadcast protocols and the WQO $\trianglelefteq$, the following hold:*

- *transitions are compatible with $\trianglelefteq$, i.e., if $\mathbf{q} \trianglelefteq \mathbf{q}'$ and $\mathbf{q} \to \mathbf{p}$, then $\exists \mathbf{p}'$ with $\mathbf{p} \trianglelefteq \mathbf{p}'$ and $\mathbf{q}' \to^* \mathbf{p}'$. They are strongly compatible if they satisfy the first or the second condition of well-behavedness.*
- *we can effectively compute Pred.*

Proof. As above, it is sufficient to prove the statements for every action $a \in A$ separately.

For the first item, we start by considering a broadcast transition with exactly one sender. Let $\mathbf{q} \trianglelefteq \mathbf{q}'$ and $\mathbf{q} \xrightarrow{a,G} \mathbf{p}$ based on the local transition $s_k \xrightarrow{a!!} s_l$ of the sender. Note that this local transition is also enabled in $\mathbf{q}'$, and distinguish two cases: (1) if $s_k = s_l$ or $M_a(s_k) \neq s_k$, then by the argument from the proof of Theorem 2 we get $\mathbf{p} \preceq \mathbf{p}'$ for $\mathbf{q}' \xrightarrow{a,G} \mathbf{p}'$. Furthermore, we also get $\mathsf{supp}(\mathbf{p}) = \mathsf{supp}(\mathbf{p}')$, and therefore $\mathbf{p} \trianglelefteq \mathbf{p}'$. (2) if $s_k \neq s_l$ and $M_a(s_k) = s_k$, the second part of the argument above does not hold: if $\mathbf{q}(s_k) = 1$ and $\mathbf{q}'(s_k) > 1$, then we can have $s_k \in \mathsf{supp}(\mathbf{p}')$ while $s_k \notin \mathsf{supp}(\mathbf{p})$. However, in this case we know that the third condition of well-behavedness must hold. Since the same transition is still enabled for other processes remaining in s_k, we can repeat it a often as needed to arrive in a state with $\mathsf{supp}(\mathbf{p}) = \mathsf{supp}(\mathbf{p}')$. For negotiation transitions, note that the argument is even simpler: we know that either the first or the second condition of well-behavedness applies. Finally, for k-broadcasts the argument is essentially the same as above, noting that if fewer than k processes remain in the state after repeated application, then the k-broadcast can still be executed.

For the second item, observe that for a transition based on $s_k \xrightarrow{a!!,G} s_l$, any element $\mathbf{q}$ of $Pred(C)$ must satisfy (i) $\mathbf{q}(s_k) \geq 1$, (ii) $\mathbf{q}(s) = 0$ for $s \notin G$, and

(iii) $M_a \cdot \mathbf{q} + \mathbf{v}_a = \mathbf{q}'$, for some $\mathbf{q}' \in C$. The basis of $Pred(C)$ consists of the minimal elements (wrt. $\trianglelefteq$) that satisfy these conditions, and thus is computable. Again, a similar argument applies for negotiations and k-broadcasts.

B.4 Cutoffs for (Guarded) Broadcast Protocols

Finally, we want to investigate the connection between the decidability result in Theorem 3 and cutoff results for parameterized systems. While Theorem 3 allows us to decide parameterized model checking problems and can be integrated into a semi-procedure for parameterized synthesis, a cutoff result is more general and directly reduces both parameterized verification and synthesis to a problem over a given, fixed number of processes.

Basic Observations. Formally, a *cutoff* for a class of parameterized systems and a class of specifications is a number $c \in \mathbb{N}$ such that for every system component P and specification φ from the respective classes,

$$\forall n \geq c : (P^c \models \varphi \iff P^n \models \varphi) .$$

The basic idea is that, based on the proofs of the second items of Theorems 2 and 3, we can determine a number c of processes in our system that is sufficient to reach a set of error states (that we assume to be upwards-closed in our WQO), i.e., an error state is reachable with any number $n \geq c$ of processes if and only if it is reachable with c processes.

To this end, consider the conditions on the basis of $Pred(C)$ that are given in the proof: the only case in which these conditions require us to increase the number of processes is if we want to reach one or more states through a broadcast-receive (and there is no or there are not enough processes in the state from which the corresponding broadcast-send can be fired). If we look at the special case of `choose`, we can see that the transition on `choose`(k), for arbitrary k, can be seen as a broadcast-send, while the transition on $\neg$`choose`(k) has to be considered a broadcast-receive with k senders. On the other hand, the special case of negotiations never requires us to add processes, as the whole transition can be described only by a broadcast matrix M_a.

Note that this means that in general the cutoff can grow very big. Consider the example shown in Fig. 3, where we want to reach state E: to reach E from s_0 with a single process, we need 2 other processes that can fire the necessary broadcast-sends $a_1!!$ and $a_2!!$. To reach E with m processes, we need to take m rounds through the cycle $s_0 \to s_1 \to s_2 \to s_0$, and in each round need 2 other processes to fire the broadcast-sends. Thus, we need $m \cdot 2$ additional processes. Finally, consider the case were we increase the size of the cycle: if we have k states in the cycle (plus sink and E) and want to reach E with m processes, then we will need $m \cdot (k-1)$ additional processes. In all of these cases, this observation gives us a lower bound on the cutoff for the respective properties. In particular, a cutoff for reachability properties in broadcast systems in general

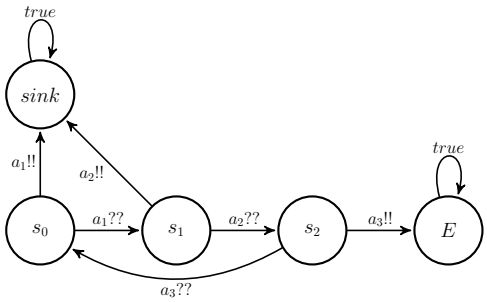

Fig. 3. Example for a linear lower bound on cutoffs

is at least in the order of $m \cdot k$, where m is the number of processes that should reach the state, and k is the number of states in the process.

Since cutoffs that grow with the size of the state-space make synthesis (and verification) very difficult in practice, in the following we identify local conditions on processes that allow us to compute small cutoffs.

Conditions for Small Cutoffs. We begin with a very simple case, where systems are restricted to only internal transitions and negotiation transitions.

Lemma 5. *For systems that only have internal transitions and negotiations, the cutoff for reachability of any state s by m processes is m.*

Proof. To see this, first consider a system with $n > m$ processes, where eventually m of them reach s. We can simulate this run in a system with m processes by simply keeping the m processes that reach s, and remove all others. Similarly, if all processes in a system of size m eventually reach s, then we can simulate this run in a bigger system by adding processes (that need to take negotiation steps when the other processes do so, and otherwise can behave arbitrarily).

While we will in general not be interested in systems that can *only* communicate through internal transitions and negotiations, we can refine this observation based on the states that we are interested in, and allow some different kinds of communication in our system.

To this end, define a transition of a process P to be *free* if it does not depend on other processes, i.e., it is an internal transition, a sending transition, or a negotiation. A path from one state to another is free if all transitions on the path are free.

Lemma 6. *Given a well-behaved guarded broadcast protocol, if all paths from s_0 to s are free, then $c = m$ is a cutoff for reachability of a state $\mathbf{q}$ with $\mathbf{q}(s) = m$.*

Proof. The argument follows the same line as the one above for systems with only internal transitions and negotiations, noting that transitions along a free path can also be taken regardless of the other processes in the system.

B.5 Connection between the Models

In this section, we show how to map a system description in the `choose` model (as introduced in Sec. 3.1) to a system in the GBC model (as introduced in Appendix B.2). The goal is to obtain parameterized safety guarantees for systems in the `choose` model, and ultimately to use cutoff results for the GBC model in the synthesis of systems in the `choose` model.

Mapping the `choose` model to the GBC model. The `choose` model is very similar to the GBC model in many aspects: both models support finite-state processes, and many of the communication primitives are shared, and have the same semantics. When mapping a system from the `choose` model to the GBC model, the main obstacle is that the `choose` model supports communication based on process ids, e.g., a process can pass its id in a broadcast, or answer to a process with a specific id from which it has received a message before. Moreover, processes can store id values as well as sets of id values in local variables. None of this is supported in the GBC model, where the local state space has to be fixed and cannot depend on the number of processes in the system.

Therefore, in general we will lose information when mapping a process in the `choose` model to the GBC model. However, we can show that for certain use cases, which include the accumulation of information that is necessary to execute `choose` transitions, substructures of a process that use id-based communication can be mapped to structures in the GBC model such that safety/reachability properties are preserved.

Abstracting the collection of id-based information. In order to perform a consensus round, the system needs to reach a `GlobalConsensusStart` state. To this end, processes can use id-based communication to exchange information about who wants to participate. However, if we assume that we are not interested in *how* this happens, then any structure that ensures that `GlobalConsensusStart` will be reached can be modeled in the GBC model: we simply let participants and non-participants move to different states, and block the execution of `choose` transitions unless all participants have reached these states, see Fig. 4.

Based on these observations, we assume that the systems we are interested in only use id-based communication in order to ensure that `GlobalConsensusStart` is reached, and that this information will not affect the behavior of the system after the corresponding `choose` transitions are executed. I.e., whenever id-based communication is used, it happens in a clearly separated part of the local state space that is entered and left by all processes at the same time.

As an example, consider Fig. 1, and note that the structure in the figure naturally satisfies the conditions: with all processes starting from Env, we will eventually reach a `GlobalConsensusStart` state. Before that, executing `choose` transitions is not possible, and after that the collected information will not be used anymore.

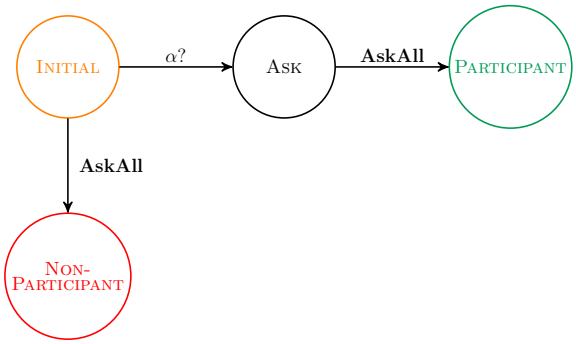

Fig. 4. A construct to abstract (possibly multiple) communications in `choose` model. Processes start at the Initial state, and are partitioned into two groups according to an event α (e.g. an environment signal). The two groups of processes (in Initial and Ask states) take a negotiation step, **AskAll** which moves the processes to Participant and Non-Participant states.

Our assumptions on substructures that use id-based communication then allow us to map a system description from the `choose` model to the GBC model in the following way:

- a substructure of the state space with entry state Init, positive exit state Participant and negative exit state Non-Participant is mapped to the component displayed in Fig. 4, and the `choose` transition that leaves Participant is guarded with the set $G = \{\text{Participant}, \text{Non-Participant}\}$
- every state and transition outside of such substructures is mapped without change to the GBC model.

Remark. The idea of substructures can be generalized to, for example, have more exit states. The substructure in Fig. 4 is one instantiation that is sufficient for our benchmarks.

Properties of the Mapping and Correctness Argument. We claim that the mapping introduced above preserves reachability properties in parameterized systems.

Target Specifications. We consider specifications that express the reachability of a combination of locations by a fixed number of processes, while all other processes can be in arbitrary states (e.g. no more than two smoke detectors can be in Report state). More precisely, specifications can refer to any state in the `choose` model that is not abstracted away by the mapping.

Simulation Equivalence. If ϕ is such a specification, then we get the property

$$\forall n.\, (\mathcal{M}_{GBC}(n) \models \phi \iff \mathcal{M}_{\texttt{choose}}(n) \models \phi)\,.$$

This can easily be seen since (i) entry to and exit from the substructures that are abstracted happen under the same conditions, (ii) within the substructures, all states that are not abstracted away are reachable for all processes (regardless of the size of the system), and (iii) outside of these substructures the systems are identical.

As a consequence of this property, we know that for our target specifications, cutoffs from the GBC model can also be used in the `choose` model. In particular, if we can guarantee that the synthesized system will satisfy the properties defined above, then we can use the cutoffs also in synthesis of process implementations in the `choose` model.

Preserving cutoffs in `choose` *model.* Recall that the cutoffs depend on the notion of free paths discussed in Appendix B.4, hence, in order to use the same cutoffs in the `choose` model, we need to ensure that the synthesized solution has free paths equivalent to those in the GBC model. We achieve this by encoding the notion of free paths as constraints to the synthesis engine.

C Evaluation Benchmarks

In this section, we provide specifications, completions, and figures for out benchmarks from Sec. 6. Kinara$_{ch}$ can successfully verify all of our case studies. The transition diagrams demonstrate the *complete* implementation, with the holes we synthesize marked in dashed red boxes.

C.1 Chubby Application

Specifications. The safety properties for this system are: (1) there is at most one leader at a time; and (2) each write is committed to all replicas before it is acknowledged. The liveness (progress) properties are: (1) client reads and writes must eventually be acknowledged; and (2) on a timeout, all servers must go back to the *Candidate* state where a new server is elected.

Synthesis Completions. Since there are no guards and updates for Chubby, the completion here had a different style: for the actions in the dashed red boxes (Fig. 5), the *next state* is left unspecified. The job of the synthesis to answer questions like: in the case of a timeout, where should the leader transition to, and how should the replicas react? Also, how should the leader server react to a read or write? How should the replicas react to a write?. In addition to that, we also allow the synthesis procedure here to figure out the right number of winners for consensus. Note that he cutoff is 2 according to Lemma 6.

C.2 Smoke Detector with Reset

Specifications The safety and liveness properties are the same as those of the example from Fig. 1. With the presence of a reset signal, the verifier must make

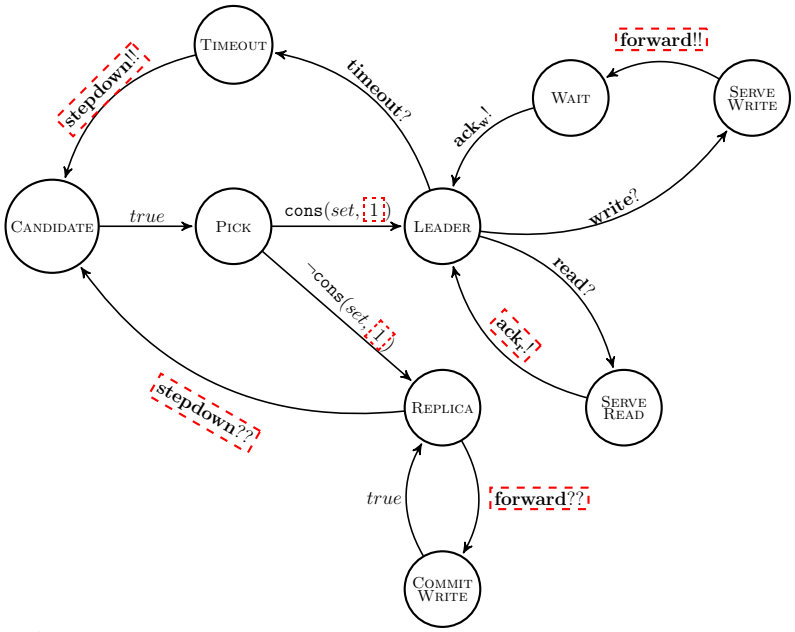

Fig. 5. A description of an application using Chubby lock service to simulate a distributed consistent file system. The *set* here trivially contains all the servers since, all servers try to become leaders. All servers start at the Candidate state where they are eventually partitioned into one leader and $n-1$ replicas.

sure the the specifications are met for each *round* of the system: before the detectors go back to the detecting a new fire, they must correctly handle the current one.

Completions. We omitted the guards and updates in the dashed red boxes and were able to synthesize the correct completions to satisfy the specification. The omitted parts correspond to interesting questions the developer of such system might have. For example, should the detector attempt to build a participant set of detectors that sensed a fire? what should a detector do upon receiving a message from another detector? After the message exchange is done, should the detector attempt to contact the fire department or is other processes going to do so?. Since all detectors go back to the initial state at the same time, this system is around-based and the cutoff is 3 according to Lemma 6.

C.3 Distributed Mobile Robotics(DMRs)

Specifications. The safety property of this system, is that exactly one robot can be planning at a given time, hence no collision will happen[15]. The liveness properties enforce that any submitted task should eventually be executed.
Completions. As shown in Fig. 6, the holes are similar to those of the smoke

[15] We abstract away from the mechanics of creating the plan itself.

detector. Additionally, a hole corresponding to how should the robot handle a message from another robot that is done planning. The cutoff for this example is 3, according to Lemma 6[16].

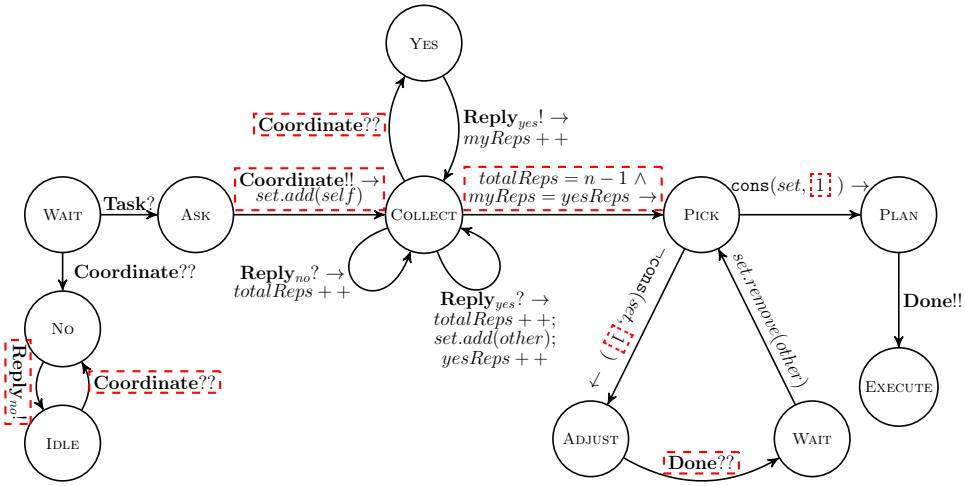

Fig. 6. A process description for a robot. A reset signal from the environment that sends all robots from EXECUTE and IDLE back to the WAIT STATE, is not shown in the figure.

C.4 Small Aircraft Transportation System (SATS)

Protocol Description. As shown in Fig. 7, an aircraft starts in FLY state. The aircraft then approaches the airport and head to a holding zone that can be either on the *left* or the *right*, represented by HOLDLEFT and HOLDRIGHT, respectively. Each holding zone can accommodate two aircraft. From either the left or the right, an aircraft may try to approach a BASE where it attempts to approach the runway and land. In that BASE zone, only one aircraft may move to the FINAL APPROACH zone where it can safely attempt to land, entering the LANDED state. However, due to unforeseen reasons (e.g. ice on the runway), the pilot may decide to abort landing. At that point, the aircraft needs to head to a miss zone either on the left or the right, MISSLEFT and MISSRIGHT, respectively. From there, the aircraft goes back to the assigned holding zone and attempts the landing again.

This is an instance where it is useful to allow the `choose` transitions to also return the winner and loser sets to the participants. For example, set_2 is the

[16] With some relaxation of the free path requirement: we allow broadcast-receive transitions that lead to a state with a free path.

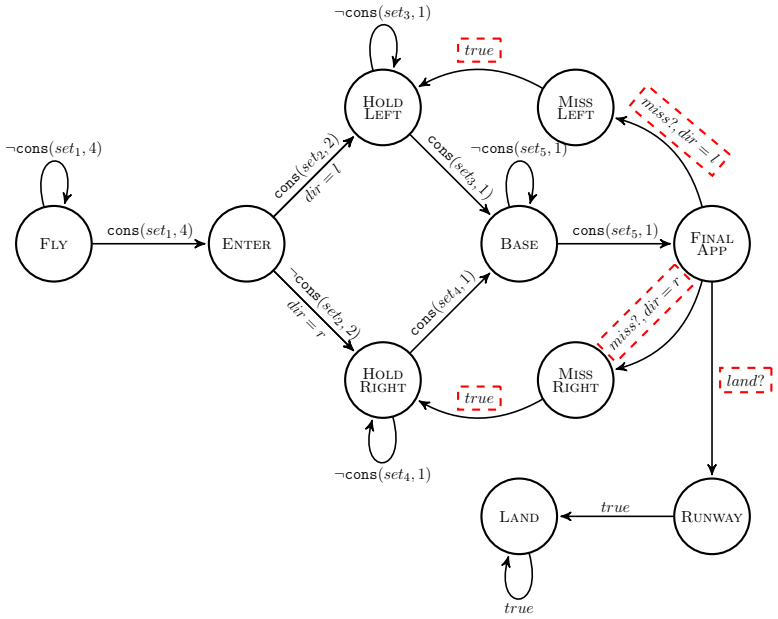

Fig. 7. SATS protocol description. If a transition has updates, they show up under the arrow.

winning set of `cons` $(set_1, 4)$ and set_3 and set_4 are the winning and losing sets of `cons` $(set_2, 2)$, respectively. Building set_5 requires the processes to communicate as in the smoke detector examples, we omit the communication states to keep the presentation simpler. Note that Table 1 includes the four states necessary to build set_5, but not shown here, in its characterization of SATS.

Specifications The safety properties, given by NASA, are as follows:

- While any number of aircraft can be in the Fly and Landed states, no more than four aircraft total can be in any of the other states.
- Only one aircraft is allowed on the FinalApproach state.
- No more than two aircraft allowed in HoldLeft (HoldRight) state.
- No more than two aircraft allowed in MissLeft (MissRight) state.
- At most two aircraft are assigned to miss left (right) at any time.

For liveness, the protocol description states that all aircraft land in order of sequence number. However, enforcing this property breaks symmetry. Instead, we relax this property to "an airplane entering the airport will land", and leave enforcing sequential ordering to the non-deterministic results of consensus, outside of our verification scope.

Completions. The completions in this system were to synthesize the actions of the pilot attempting to land. In particular, which direction should the pilot take in case of an abnormal condition. The cutoff value for this example, is 5, according to Lemma 6.