A Strategy for Automatic Verification of Stabilization of Distributed Algorithms

Ritwika Ghosh(✉) and Sayan Mitra

University of Illinois, Urbana Champaign, USA
{rghosh9,mitras}@illinois.edu

Abstract. Automatic verification of convergence and stabilization properties of distributed algorithms has received less attention than verification of invariance properties. We present a semi-automatic strategy for verification of stabilization properties of arbitrarily large networks under structural and fairness constraints. We introduce a sufficient condition that guarantees that every fair execution of any (arbitrarily large) instance of the system stabilizes to the target set of states. In addition to specifying the protocol executed by each agent in the network and the stabilizing set, the user also has to provide a measure function or a ranking function. With this, we show that for a restricted but useful class of distributed algorithms, the sufficient condition can be automatically checked for arbitrarily large networks, by exploiting the small model properties of these conditions. We illustrate the method by automatically verifying several well-known distributed algorithms including link-reversal, shortest path computation, distributed coloring, leader election and spanning-tree construction.

1 Introduction

A system is said to stabilize to a set of states $\mathcal{X}^*$ if all its executions reach some state in $\mathcal{X}^*$ [1]. This property can capture common progress requirements like absence of deadlocks and live-locks, counting to infinity, and achievement of self-stabilization in distributed systems. Stabilization is a liveness property, and like other liveness properties, it is generally impossible to verify automatically. In this paper, we present sufficient conditions which can be used to automatically prove stabilization of distributed systems with arbitrarily many participating processes.

A sufficient condition we propose is similar in spirit to Tsitsiklis' conditions given in [2] for convergence of iterative asynchronous processes. We require the user to provide a measure function, parameterized by the number of processes, such that its sub-level sets are *invariant* with respect to the transitions and there is a *progress* making action for each state.[1] Our point of departure is a

This work is supported in part by research grants NSF CAREER 1054247 and AFOSR YIP FA9550-12-1-0336.

[1] A sub-level set of a function comprises of all points in the domain which map to the same value or less.

S. Graf and M. Viswanathan (Eds.): FORTE 2015, LNCS 9039, pp. 35–49, 2015.
DOI: 10.1007/978-3-319-19195-9_3

non-interference condition that turned out to be essential for handling models of distributed systems. Furthermore, in order to handle non-deterministic communication patterns, our condition allows us to encode fairness conditions and different underlying communication graphs.

Next, we show that these conditions can be transformed to a forall-exists form with a small model property. That is, there exists a cut-off number N_0 such that if the condition(s) is(are) valid in all models of sizes up to N_0, then it is valid for all models. We use the small model results from [3] to determine the cut-off parameter and apply this approach to verify several well-known distributed algorithms.

We have a Python implementation based on the sufficient conditions for stabilization we develop in Section 3. We present precondition-effect style transition systems of algorithms in Section 4 and they serve as pseudo-code for our implementation. The SMT-solver is provided with the conditions for *invariance*, *progress* and *non-interference* as assertions. We encode the distributed system models in Python and use the Z3 theorem-prover module [4] provided by Python to check the conditions for stabilization for different model sizes.

We have used this method to analyze a number of well-known distributed algorithms, including a simple distributed coloring protocol, a self-stabilizing algorithm for constructing a spanning tree of the underlying network graph, a link-reversal routing algorithm, and a binary gossip protocol. Our experiments suggest that this method is effective for constructing a formal proof of stabilization of a variety of algorithms, provided the measure function is chosen carefully. Among other things, the measure function should be *locally computable:* changes from the measure of the previous state to that of the current state only depend on the vertices involved in the transition. It is difficult to determine whether such a measure function exists for a given problem. For instance, consider Dijkstra's self-stabilizing token ring protocol [5]. The proof of correctness relies on the fact that the leading node cannot push for a value greater than its previous unique state until every other node has the same value. We were unable to capture this in a locally computable measure function because if translated directly, it involves looking at every other node in the system.

1.1 Related Work

The motivation for our approach is from the paper by John Tsitsiklis on convergence of asynchronous iterative processes [2], which contains conditions for convergence similar to the sufficient conditions we state for stabilization. Our use of the measure function to capture stabilization is similar to the use of Lyapunov functions to prove stability as explored in [6], [7] and [8]. In [9], Dhama and Theel present a *progress monitor* based method of designing self-stabilizing algorithms with a weakly fair scheduler, given a self-stabilizing algorithm with an arbitrary, possibly very restrictive scheduler. They also use the existence of a ranking function to prove convergence under the original scheduler. Several authors [10] employ functions to prove termination of distributed algorithms, but while they may provide an idea of what the measure function can be, in general they do not translate exactly to the measure functions that our verification

strategy can employ. The notion of fairness we have is also essential in dictating what the measure function should be, while not prohibiting too many behaviors. In [7], the assumption of serial execution semantics is compatible with our notions of fair executions.

The idea central to our proof method is the small model property of the sufficient conditions for stabilization. The small model nature of certain invariance properties of distributed algorithms (eg. distributed landing protocols for small aircrafts as in [11]) has been used to verify them in [12]. In [13], Emerson and Kahlon utilize a small model argument to perform parameterized model checking of ring based message passing systems.

2 Preliminaries

We will represent distributed algorithms as transition systems. Stabilization is a liveness property and is closely related to *convergence* as defined in the works of Tsitsiklis [2]; it is identical to the concept of *region stability* as presented in [14]. We will use measure functions in our definition of stabilization. A *measure function* on a domain provides a mapping from that domain to a well-ordered set. A well-ordered set W is one on which there is a total ordering $<$, such that there is a minimum element with respect to $<$ on every non-empty subset of W. Given a measure function $C : A \to B$, there is a partition of A into sub level-sets. All elements of A which map to the same element $b \in B$ under C are in the same sub level-set L_b.

We are interested in verifying stabilization of distributed algorithms independent of the number of participating processes or nodes. Hence, the transition systems are parameterized by N—the number of nodes. Given a non-negative integer N, we use $[N]$ to denote a set of indices $\{1, 2, \ldots, N\}$.

Definition 1. *For a natural number N and a set Q, a transition system $A(N)$ with N nodes is defined as a tuple (X,A,D) where*

a) $\mathcal{X}$ is the state space of the system. If the state space of of each node is Q, $\mathcal{X} = Q^N$.
b) A is a set of actions.
c) $D : \mathcal{X} \times A \to \mathcal{X}$ is a transition function, that maps a system-state action pair to a system-state.

For any $x \in \mathcal{X}$, the i^{th} component of x is the state of the i^{th} node and we refer to it as $x[i]$. Given a transition system $\mathcal{A}(N) = (\mathcal{X}, A, D)$ we refer to the state obtained by the application of the action a on a state $x \in \mathcal{X}$ i.e, $D(x, a)$, by $a(x)$.

An execution of $\mathcal{A}(N)$ records a particular run of the distributed system with N nodes. Formally, an *execution* α of $\mathcal{A}(N)$ is a (possibly infinite) alternating sequence of states and actions $x_0, a_1, x_1, \ldots$, where each $x_i \in \mathcal{X}$ and each $a_i \in A$ such that $D(x_i, a_{i+1}) = x_{i+1}$. Given that the choice of actions is nondeterministic in the execution, it is reasonable to expect that not all executions may stabilize. For instance, an execution in which not all nodes participate, may not stabilize.

Definition 2. *A* fairness condition $\mathcal{F}$ *for* $\mathcal{A}(N)$ *is a finite collection of subsets of actions* $\{A_i\}_{i \in I}$*, where* I *is a finite index set. An action-sequence* $\sigma = a_1, a_2, \ldots$ *is* $\mathcal{F}$-Fair *if every* A_i *in* $\mathcal{F}$ *is represented in* σ *infinitely often, that is,*

$$\forall\ A' \in\ \mathcal{F},\ \forall i \in \mathbb{N},\ \exists k > i,\ a_k \in A'.$$

For instance, if the fairness condition is the collection of all singleton subsets of A, then each action occurs infinitely often in an execution. This notion of fairness is similar to action based fairness constraints in temporal logic model checking [15]. The network graph itself enforces whether an action is enabled: every pair of adjacent nodes determines a continuously enabled action. An execution is strongly fair, if given a set of actions A such that all actions in A are infinitely often enabled; some action in A occurs infinitely often in the it. An *$\mathcal{F}$-fair execution* is an infinite execution such that the corresponding sequence of actions is $\mathcal{F}$-fair.

Definition 3. *Given a system* $\mathcal{A}(N)$*, a fairness condition* $\mathcal{F}$*, and a set of states* $\mathcal{X}^* \subseteq \mathcal{X}$*,* $\mathcal{A}(N)$ *is said to* $\mathcal{F}$-stabilize to $\mathcal{X}^*$ *iff for any* $\mathcal{F}$*-fair execution* $\alpha = x_0, a_1, x_1, a_2, \ldots$*, there exists* $k \in \mathbb{N}$ *such that* $x_k \in \mathcal{X}^*$*.* $\mathcal{X}^*$ *is called a* ***stabilizing set*** *for* $\mathcal{A}$ *and* $\mathcal{F}$*.*

It is different from the definition of self-stabilization found in the literature [1], in that the stabilizing set $\mathcal{X}^*$ is not required to be an invariant of $\mathcal{A}(N)$. We view proving the invariance of $\mathcal{X}^*$ as a separate problem that can be approached using one of the available techniques for proving invariance of parametrized systems in [3], [12].

Example 1. (Binary Gossip) We look at binary gossip in a ring network composed of N nodes. The nodes are numbered clockwise from 1, and nodes 1 and N are also neighbors. Each node has one of two states : $\{0, 1\}$. A pair of neighboring nodes communicates to exchange their values, and the new state is set to the binary Or ($\vee$) of the original values. Clearly, if all the interactions happen infinitely often, and the initial state has at least one node state 1, this transition system stabilizes to the state $x = 1^N$. The set of actions is specified by the set of edges of the ring. We first represent this protocol and its transitions using a standard precondition-effect style notation similar to one used in [16].

```
Automaton Gossip[N : ℕ]
type indices : [N]
type  values : {0, 1}
variables
        x[indices → values]
transitions
        step(i: indices, j: indices)
        pre True
        eff x[i] = x[j] = x[i] ∨ x[j]
measure
        func C : x ↦ Sum(x)
```

The above representation translates to the transition system $\mathcal{A}(N) = (\mathcal{X}, A, D)$ where

1. The state space of each node is $Q = \{0, 1\}$, i.e $\mathcal{X} = \{0, 1\}^N$.
2. The set of actions is $A = \{step(i, i+1) \mid 1 \leq i < N\} \cup \{(N, 1)\}$.
3. The transition function is $D(x, step(i, j)) = x'$ where $x'[i] = x'[j] = x[i] \vee x[j]$.

We define the stabilizing set to be $X^* = \{1^N\}$, and the fairness condition is $\mathcal{F} = \{\{(i, i+1\} \mid 1 < i < N\} \cup \{1, N\}$, which ensures that all possible interactions take place infinitely often. In Section 3 we will discuss how this type of stabilization can be proven automatically with a user-defined measure function.

3 Verifying Stabilization

3.1 A Sufficient Condition for Stabilization

We state a sufficient condition for stabilization in terms of the existence of a *measure function*. The measure functions are similar to Lyapunov stability conditions in control theory [17] and well-founded relations used in proving termination of programs and rewriting systems [18].

Theorem 1. *Suppose $\mathcal{A}(N) = \langle \mathcal{X}, A, D \rangle$ is a transition system parameterized by N, with a fairness condition $\mathcal{F}$, and let $\mathcal{X}^*$ be a subset of $\mathcal{X}$. Suppose further that there exists a measure function $C : \mathcal{X} \to W$, with minimum element $\perp$ such that the following conditions hold for all states $x \in X$:*

- (invariance) $\forall\ a \in A,\ C(a(x)) \leq C(x)$,
- (progress) $\exists\ A_x \in \mathcal{F},\ \forall a \in A_x,\ C(x) \neq \perp \Rightarrow C(a(x)) < C(x)$,
- (noninterference) $\forall a, b \in A,\ C(a(x)) < C(x) \Rightarrow C(a(b(x))) < C(x)$, *and*
- (minimality) $C(x) = \perp\ \Rightarrow\ x \in \mathcal{X}^*$.

Then, $\mathcal{A}[N]$ $\mathcal{F}$-stabilizes to $\mathcal{X}^*$.

Proof. Consider an $\mathcal{F}$-fair execution $\alpha = x_0 a_1 x_1 \ldots$ of $\mathcal{A}(N)$ and let x_i be an arbitrary state in that execution. If $C(x_i) = \perp$, then by *minimality*, we have $x_i \in \mathcal{X}^*$. Otherwise, by the *progress* condition we know that there exists a set of actions $A_{x_i} \in \mathcal{F}$ and $k > i$, such that $a_k \in A_{x_i}$, and $C(a_k(x_i)) < C(x_i)$. We perform induction on the length of the sub-sequence $x_i a_{i+1} x_{i+1} \ldots a_k x_k$ and prove that $C(x_k) < C(x_i)$. For any sequence β of intervening actions of length n,

$$C(a_k(x_i)) < C(x_i) \Rightarrow C(a_k(\beta(x_i))) < C(x_i).$$

The base case of the induction is $n = 0$, which is trivially true. By induction hypothesis we have: for any $j < n$, with length of β equal to j,

$$C(a_k(\beta(x_i)) < C(x_i).$$

We have to show that for any action $b \in A$,

$$C(a_k(\beta(b(x_i)))) < C(x_i).$$

There are two cases to consider. If $C(b(x_i)) < C(x_i)$ then the result follows from the *invariance* property. Otherwise, let $x' = b(x_i)$. From the invariance of b we have $C(x') = C(x_i)$. From the noninterference condition we have

$$C(a(b(x_i)) < C(x_i),$$

which implies that $C(a(x')) < C(x')$. By applying the induction hypothesis to x' we have the required inequality $C(a_k(\beta(b(x_i)))) < C(x_i)$. So far we have proved that either a state x_i in an execution is already in the stabilizing set, or there is a state $x_k, k > i$ such that $C(x_k) < C(x_i)$. Since $<$ is a well-ordering on $C(\mathcal{X})$, there cannot be an infinite descending chain. Thus

$$\exists j(j > i \wedge C(j) = \perp).$$

By minimality , $x_j \in X^*$. By invariance again, we have $\mathcal{F}$-stabilization to X^* □

We make some remarks on the conditions of Theorem 1. It requires the measure function C and the transition system $\mathcal{A}(N)$ to satisfy four conditions. The *invariance* condition requires the sub-level sets of C to be invariant with respect to all the transitions of $\mathcal{A}(N)$. The *progress* condition requires that for every state x for which the measure function is not already $\perp$, there exists a fair set of actions A_x that takes x to a lower value of C.

The *minimality* condition asserts that $C(x)$ drops to $\perp$ only if the state is in the stabilizing set $\mathcal{X}^*$. This is a part of the specification of the stabilizing set.

The *noninterference* condition requires that if a results in a decrease in the value of the measure function at state x, then application of a to another state x' that is reachable from x also decreases the measure value below that of x. Note that it doesn't necessarily mean that a decreases the measure value at x', only that either x' has measure value less than x at the time of application of a or it drops after the application. In contrast, the progress condition of Theorem 1 requires that for every sub-level set of C there is a fair action that takes *all* states in the sub-level set to a smaller sub-level set.

To see the motivation for the noninterference condition, consider a sub-level set with two states x_1 and x_2 such that $b(x_1) = x_2$, $a(x_2) = x_1$ and there is only one action a such that $C(a(x_1)) < C(x_1)$. But as long as a does not occur at x_1, an infinite (fair) execution $x_1bx_2ax_1bx_2\ldots$ may never enter a smaller sub-level set.

In our examples, the actions change the state of a node or at most a small set of nodes while the measure functions succinctly captures global progress conditions such as the number of nodes that have different values. Thus, it is often impossible to find actions that reduce the measure function for all possible states in a level-set. In Section 4, we will show how a candidate measure function can be checked for arbitrarily large instances of a distributed algorithm, and hence, lead to a method for automatic verification of stabilization.

3.2 Automating Stabilization Proofs

For finite instances of a distributed algorithm, we can use formal verification tools to check the sufficient conditions in Theorem 1 to prove stabilization. For transition systems with invariance, progress and noninterference conditions that can be encoded appropriately in an SMT solver, these checks can be performed automatically. Our goal, however, is to prove stabilization of algorithms with an arbitrary or unknown number of participating nodes. We would like to define a parameterized family of measure functions and show that $\forall N \in \mathbb{N}, \mathcal{A}(N)$ satisfies the conditions of Theorem 1. This is a parameterized verification problem and most of the prior work on this problem has focused on verifying invariant properties (see Section 1 for related works). Our approach will be based on exploiting the small model nature of the logical formulas representing these conditions.

Suppose we want to check the validity of a logical formula of the form $\forall\ N \in \mathbb{N}, \phi(N)$. Of course, this formula is valid iff the negation $\exists\ N \in \mathbb{N}, \neg\phi(N)$ has no satisfying solution. In our context, checking if $\neg\phi(N)$ has a satisfying solution over all integers is the (large) search problem of finding a counter-example. That is, a particular instance of the distributed algorithm and specific values of the measure function for which the conditions in Theorem 1 do not hold. The formula $\neg\phi(N)$ is said to have a small model property if there exists a cut-off value N_0 such that if there is no counter-example found in any of the instances $\mathcal{A}(1), \mathcal{A}(2), \ldots, \mathcal{A}(N_0)$, then there are no counter-examples at all. Thus, if the conditions of Theorem 1 can be encoded in such a way that they have these small model properties then by checking them over finite instances, we can infer their validity for arbitrarily large systems.

In [3], a class of $\forall\exists$ formulas with small model properties were used to check invariants of timed distributed systems on arbitrary networks. In this paper, we will use the same class of formulas to encode the sufficient conditions for checking stabilization. We use the following small model theorem as presented in [3]:

Theorem 2. *Let $\Gamma(N)$ be an assertion of the form*

$$\forall i_1, \ldots, i_k \in [N] \exists j_1, \ldots, \jmath_m \in [N], \phi(i_1, \ldots, i_k, j_1, \ldots, j_m)$$

where ϕ is a quantifier-free formula involving the index variables, global and local variables in the system. Then, $\forall N \in \mathbb{N} : \Gamma(N)$ is valid iff for all $n \leq N_0 = (e+1)(k+2)$, $\Gamma(n)$ is satisfied by all models of size n, where e is the number of index array variables in ϕ and k is the largest subscript of the universally quantified index variables in $\Gamma(N)$.

3.3 Computing the Small Model Parameter

Computing the small model parameter N_0 for verifying a stability property of a transition system first requires expressing all the conditions of Theorem 1 using formulas which have the structure specified by Theorem 2. There are a few important considerations while doing so.

Translating the sufficient conditions. In their original form, none of the conditions of Theorem 1 have the structure of $\forall\exists$-formulas as required by Theorem 2. For instance, a leading $\forall x \in \mathcal{X}$ quantification is not allowed by Theorem 2, so we transform the conditions into formulas with implicit quantification. Take for instance the invariance condition: $\forall x \in \mathcal{X}, \forall a \in A, (C(a(x)) \leq C(x))$. Checking the validity of the invariance condition is equivalent to checking the satisfiability of $\forall a \in A, (a(x) = x' \Rightarrow C(x') \leq C(x))$, where x' and x are free variables, which are checked over all valuations. Here we need to check that x and x' are actually states and they satisfy the transition function. For instance in the binary gossip example, we get

$$
\begin{aligned}
\textit{Invariance}: \ & \forall x \in \mathcal{X}, \forall a \in A,\ C(a(x)) \leq C(x) \text{ is verified as} \\
& \forall a \in A,\ x' = a(x) \Rightarrow C(x') \leq C(x). \\
\equiv \ & \forall i, j \in [N],\ x' = \textit{step}(i,j)(x) \Rightarrow \textit{Sum}(x') \leq \textit{Sum}(x). \\
\textit{Progress}: \ & \forall x \in \mathcal{X}, \exists a \in A,\ C(x) \neq \bot \Rightarrow C(a(x)) < C(x) \\
\text{is verified as} \ & C(x) \neq 0 \\
& \Rightarrow \exists i, j \in [N], x' = \textit{step}(i,j)(x) \wedge \textit{Sum}(x)' < \textit{Sum}(x). \\
\textit{Noninterference}: \ & \forall x \in \mathcal{X}, \forall a, b \in A,\ (C(a(x)) < C(x) \equiv C(a(b(x))) < C(x)) \\
\text{is verified as} \ & \forall i, j, k, l \in [N],\ x' = \textit{step}(i,j)(x) \wedge x'' = \textit{step}(k,l)(x) \\
& \wedge x''' = \textit{step}(i,j)(x'') \Rightarrow (C(x') < C(x) \Rightarrow C(x''') < C(x)).
\end{aligned}
$$

Interaction graphs. In distributed algorithms, the underlying network topology dictates which pairs of nodes can interact, and therefore the set of actions. We need to be able to specify the available set of actions in a way that is in the format demanded by the small-model theorem. In this paper we focus on specific classes of graphs like complete graphs, star graphs, rings, k-regular graphs, and k-partite complete graphs, as we know how to capture these constraints using predicates in the requisite form. For instance, we use *edge predicates* $E(i,j) : i$ and j are node indices, and the predicate is true if there is an undirected edge between them in the interaction graph. For a complete graph, $E(i,j) = \textit{true}$. In the Binary Gossip example, the interaction graph is a ring, and $E(i,j) = (i < N \wedge j = i+1) \vee (i > 1 \wedge j = i-1) \vee i = 1 \wedge j = N)$. If the graph is a d-regular graph, we express use d arrays, $reg_1, \ldots, reg_d$, where $\exists i, reg_i[k] = l$ if there is an edge between k and l, and $i \neq j \equiv reg_i[k] \neq reg_j[k]$. This only expresses that the degree of each vertex is d, but there is no information about the connectivity of the graph. For that, we can have a separate index-valued array which satisfies certain constraints if the graph is connected. These constraints need to be expressed in a format satisfying the small model property as well. Other graph predicates can be introduced based on the model requirements, for instance, $\textit{Parent}(i,j)$, $\textit{Child}(i,j)$, $\textit{Direction}(i,j)$. In our case studies we verify stabilization under the assumption that all pairs of nodes in E interact infinitely often. For the progress condition, the formula simplifies to $\exists a \in A, C(x) \neq \bot \Rightarrow C(a(x)) < C(x))$. More general fairness constraints can be encoded in the same way as we encode graph constraints.

4 Case Studies

In this section, we will present the details of applying our strategy to various distributed algorithms. We begin by defining some predicates that are used in our case studies. Recall that we want wanted to check the conditions of Theorem 1 using the transformation outlined in Section 3.3 involving x, x' etc., representing the states of a distributed system that are related by the transitions. These conditions are encoded using the following predicates, which we illustrate using the binary gossip example given in Section 2:

- $isState(x)$ returns true iff the array variable x represents a state of the system. In the binary gossip example, $isState(x) = \forall i \in [N], x[i] = 0 \vee x[i] = 1$.
- $isAction(a)$ returns true iff a is a valid action for the system. Again, for the binary gossip example $isAction(step(i,j)) = \text{True}$ for all $i, j \in [N]$ in the case of a complete communication graph.
- $isTransition(x, step(i,j), x')$ returns true iff the state x goes to x' when the transition function for action $step(i,j)$ is applied to it. In case of the binary gossip example, $isTransition(x, step(i,j), x')$ is

$$(x'[j] = x'[i] = x[i] \vee x[j]) \wedge (\forall p,\ p \notin \{i,j\} \Rightarrow x[p] = x'[p]).$$

- Combining the above predicates, we define $P(x, x', i, j)$ as

$$isState(x) \wedge isState(x') \wedge isTransition(x, step(i,j), x') \wedge isAction(step(i,j)).$$

Using these constructions, we rewrite the conditions of Theorem 1 as follows:

$$\begin{aligned}
Invariance: &\ \forall i, j,\ P(x, x', i, j) \Rightarrow C(x') \leq C(x). && (1)\\
Progress: &\ C(x) \neq \bot \Rightarrow \exists i, j,\ P(x, x', i, j) \wedge C(x') < C(x). && (2)\\
Noninterference: &\ \forall p, r, s, t,\ P(x, x', p, q) \wedge P(x, x'', s, t) \wedge P(x'', x''', p, q)\\
&\ \Rightarrow (C(x') < C(x) \Rightarrow C(x''') < C(x)). && (3)\\
Minimality: &\ C(x) = \bot \Rightarrow x \in X^*. && (4)
\end{aligned}$$

4.1 Graph Coloring

This algorithm colors a given graph in $d + 1$ colors, where d is the maximum degree of a vertex in the graph [10]. Two nodes are said to have a conflict if they have the same color. A transition is made by choosing a single vertex, and if it has a conflict with any of its neighbors, then it sets its own state to be the least available value which is *not* the state of any of its neighbours. We want to verify that the system stabilizes to a state with no conflicts. The measure function is chosen as the set of pairs with conflicts.

```
Automaton Coloring[N : ℕ]
type indices : [N]
type  values : {1, ..., N}
variables
        x[indices ↦ values]
transitions
        internal step(i: indices)
        pre ∃j ∈ [N](E(j, i) ∧ x[j] = x[i])
        eff x[i] = min(values \{c | j ∈ [N] ∧ E(i, j) ∧ x[j] = c})
measure
        func C : x ↦ {(i, j) | E(i, j) ∧ x[i] = x[j]}
```

Here, the ordering on the image of the measure function is set inclusion.

$$
\begin{aligned}
\textit{Invariance}: \ & \forall i \in [N],\ P(x, x', i) \Rightarrow C(x') \subseteq C(x). \qquad \text{(From (1))} \\
& \equiv \forall i, j, k \in [N],\ P(x, x', i) \Rightarrow ((j, k) \in C(x') \\
& \quad \Rightarrow (j, k) \in C(x)). \\
& \equiv \forall i, j, k \in [N],\ P(x, x', i) \\
& \quad \Rightarrow (E(j, k) \wedge x[j] \neq x[k] \Rightarrow x'[j] \neq x'[k]). \\
& (E \text{ is the set of edges in the underlying graph}) \\
\textit{Progress}: \ & \exists m \in [N],\ C(x) \neq \emptyset \Rightarrow C(\mathit{step}(m)(x)) < C(x). \\
& \equiv \forall i, j \in [N], \exists m, n \in [N],\ (E(i, j) \wedge x[i] \neq x[j]) \vee \\
& (P(x, x', m) \wedge E(m, n) \wedge x[m] = x[n] \wedge x'[m] \neq x'[n]). \\
\textit{Noninterference}: \ & \forall q, r, s, t \in [N],\ (P(x, x', q) \wedge P(x, x'', s) \wedge P(x'', x''', q)) \\
& \Rightarrow (E(q, r) \wedge x[q] = x[r] \wedge x'[q] \neq x'[r] \Rightarrow E(s, t) \\
& \wedge (x'[s] \neq x'[t] \Rightarrow x'''[s] \neq x'''[t]) \wedge x'''[r] \neq x'''[q])). \\
& (\text{from (3 and expansion of ordering}) \\
\textit{Minimality}: \ & C(x) = \emptyset \Rightarrow x \in X^*.
\end{aligned}
$$

From the above conditions, using Theorem 2 N_0 is calculated to be 24.

4.2 Leader Election

This algorithm is a modified version of the Chang-Roberts leader election algorithm [10]. We apply Theorem 1 directly by defining a straightforward measure function. The state of each node in the network consists of a) its own uid, b) the index and uid of its proposed candidate, and c) the status of the election according to the node (0 : the node itself is elected, 1 : the node is not the leader, 2 : the node is still waiting for the election to finish). A node i communicates its state to its clockwise neighbor j ($i + 1$ if $i < N$, 0 otherwise) and if the UID of i's proposed candidate is greater than j, then j is out of the running. The proposed candidate for each node is itself to begin with. When a node gets back its own index and uid, it sets its election status to 0. This status, and the correct leader identity propagates through the network, and we want to verify that the system stabilizes to a state where a leader is elected. The measure function is the number of nodes with state 0.

```
Automaton Leader[N : ℕ]
type indices    : [N]
variables
          uid[indices↦ [N]]
          candidate[indices↦ [N]]
          leader[indices↦ {0, 1, 2}]
transitions
         internal step(i: indices, j: indices)
         pre leader[i] = 1 ∧ uid[candidate[i]] > uid[candidate[j]]
         eff leader[j] = 1 ∧ candidate[j] = candidate[i]
         pre leader[j] = 2 ∧ candidate[i] = j
         eff leader[j] = 0∧candidate[j] = j
         pre leader[i] = 0
         eff leader[j] = 1 ∧ candidate[j] = i
measure
         func C : x ↦ Sum(x.leader[i])
```

The function $Sum()$ represents the sum of all elements in the array, and it can be updated when a transition happens by just looking at the interacting nodes. We encode the sufficient conditions for stabilization of this algorithm using the strategy outlined in Section 3.2.

$$
\begin{aligned}
\textit{Invariance}: \ & \forall i,j \in [N],\ P(x,x',i,j) \Rightarrow (Sum(x'.leader) \leq Sum(x.leader)).\\
\equiv\ & \forall i,j \in [N],\ (P(x,x',i,j) \Rightarrow (Sum(x.leader) - x.leader[i] - \\
& x.leader[j] + x'.leader[i] + x'.leader[j] \leq Sum(x.leader)).\\
& \text{(difference only due to interacting nodes)}\\
\equiv\ & \forall i,j \in [N],\ P(x,x',i,j)\\
& \Rightarrow (x'.leader[i] + x'.leader[j] \leq x.leader[i] + x.leader[j])\\
\textit{Progress}: \ & \exists m,n \in [N],\ Sum(x.leader) \neq N-1\\
& \Rightarrow Sum(step(m,n)(x).leader) < Sum(x.leader)).\\
\equiv\ & \forall p \in [N], x.leader[p] = 2 \Rightarrow\\
& \exists m,n \in [N],\ (P(x,x',m,n) \wedge E(m,n) \wedge\\
& x'.leader[m] + x'.leader[n] < x.leader[m] + x.leader[n]).\\
& \text{(one element still waiting for election to end)}\\
\textit{Noninterference}: \ & \forall q,r,s,t \in [N],\ P(x,x',q,r) \wedge P(x,x'',s,t) \wedge P(x'',x''',q,r)\\
& \Rightarrow (x'[q] + x'[r] < x[q] + x[r]\\
& \Rightarrow (x'''[q] + x'''[r] + x'''[s] + x'''[t] < x[q] + x[r] + x[s] + x[t])).\\
& \text{(expanding out } Sum)\\
\textit{Minimality}: \ & C(x) = N-1 \Rightarrow x \in X^*.
\end{aligned}
$$

From the above conditions, using Theorem 2, N_0 is calculated to be 35.

4.3 Shortest Path

This algorithm computes the shortest path to every node in a graph from a root node. It is a simplified version of the Chandy-Misra shortest path algorithm [10]. We are allowed to distinguish the nodes with indices 1 or N in the formula

structure specified by Theorem 2. The state of the node represents the distance from the root node. The root node (index 1) has state 0. Each pair of neighboring nodes communicates their states to each other, and if one of them has a lesser value v, then the one with the larger value updates its state to $v + 1$. This stabilizes to a state where all nodes have the shortest distance from the root stored in their state. We don't have an explicit value of $\perp$ for the measure function for this, but it can be seen that we don't need it in this case. Let the interaction graph be a $d-$regular graph. The measure function is the sum of distances.

```
Automaton Shortest[N : ℕ]
type indices : [N]
type values : {1, . . . , N}
variables
        x[indices ↦ values]
transitions
        internal step(i: indices, j: indices)
        pre x[j] > x[i] + 1
        eff x[j] = x[i] + 1
        pre x[i] = 0
        eff x[j] = 1
measure
        func C : x ↦ Sum(x[i])
```

Ordering on the image of measure function is the usual one on natural numbers.

$$
\begin{aligned}
\textit{Invariance}: \ & \forall i,j \in [N],\ P(x,x',i,j) \Rightarrow Sum(x') \leq Sum(x). \\
\equiv\ & \forall, j \in [N],\ P(x,x',i,j) \\
& \quad \Rightarrow Sum(x) - x[i] - x[j] + x'[i] + x'[j] \leq Sum(x). \\
\equiv\ & \forall i,j \in [N],\ P(x,x',i,j) \Rightarrow x'[i] + x'[j] \leq x[i] + x[j]. \\
\textit{Progress}: \ & \exists m,n \in [N],\ C(x) \neq \perp \Rightarrow P(x,x',m,n) \wedge Sum(x)' < Sum(x). \\
\equiv\ & \forall k,l \in [N],\ (E(k,l) \Rightarrow x[k] \leq x[l] + 1) \\
& \vee \exists m,n \in [N](P(x,x',m,n) \wedge E(m,n) \\
& \quad \wedge x[m] + x[n] > x'[m] + x'[n]). \\
& (C(x) = \perp \text{ if there is no pair of neighboring} \\
& \text{vertices more than 1 distance apart from each other)} \\
\textit{Noninterference}: \ & \forall q,r,s,t \in [N],\ P(x,x',q,r) \wedge P(x,x'',s,t) \wedge P(x',x'',q,r) \\
& \Rightarrow (x'[q] + x'[r] < x[q] + x[r] \\
& \Rightarrow (x'''[q] + x'''[r] + x'''[s] + x'''[t] < x[q] + x[r] + x[s] + x[t])). \\
\textit{Minimality}: \ & C(x) \neq \perp \Rightarrow x \in X^* \\
\equiv\ & \forall i,j (E(i,j) \Rightarrow x[i] - x[j] \leq 1 \Rightarrow x \in X^*) \qquad \text{(definition)}
\end{aligned}
$$

N_0 is $7(d+1)$ where the graph is d-regular.

4.4 Link Reversal

We describe the full link reversal algorithm as presented by Gafni and Bertsekas in [19], where, given a directed graph with a distinguished sink vertex, it outputs

a graph in which there is a path from every vertex to the sink. There is a distinguished sink node(index N). Any other node which detects that it has only incoming edges, reverses the direction of all its edges with its neighbours. We use the vector of reversal distances (the least number of edges required to be reversed for a node to have a path to the sink, for termination. The states store the reversal distances, and the measure function is identity.

```
Automaton Reversal[N : ℕ]
type   indices : [N]
type    values : [N]
variables
        x[indices↦ values]
transitions
        internal step(i: indices)
        pre i ≠ N ∧ ∀j ∈ [N](E(i, j) ∧ (direction(i, j) = −1)
        eff ∀j ∈ [N](E(i, j) ⇒ (Reverse(i, j)) ∧ x(i) = min(x(j)))
measure
        func C: x ↦ x
```

The ordering on the image of the measure function is component-wise comparison:

$$\boldsymbol{V_1} < \boldsymbol{V_2} \Leftrightarrow \forall i(V_1[i] < V_2[i])$$

We mentioned earlier that the image of C has a well-ordering. That is a condition formulated with the idea of continuous spaces in mind. The proposed ordering for this problem works because the image of the measure function is discrete and has a lower bound (specifically, 0^N). We elaborate a bit on P here, because it needs to include the condition that the reversal distances are calculated accurately. The node N has reversal distance 0. Any other node has reversal distance $rd(i) = min(rd(j_1), \ldots rd(j_m), rd(k_1) + 1, \ldots rd(k_n) + 1)$ where $j_p(p = 1 \ldots m)$ are the nodes to which it has outgoing edges, and $k_q(q = 1 \ldots n)$ are the nodes it has incoming edges from. P also needs to include the condition that in a transition, reversal distances of no other nodes apart from the transitioning nodes change. The interaction graph in this example is complete.

$$\begin{aligned}
\textit{Invariance}: &\ \forall i, j \in [N],\ P(x, x', i) \Rightarrow x'[j] \leq x[j] \qquad \text{(ordering)} \\
\textit{Progress}: &\ \exists m \in [N],\ C(x) \neq \perp \Rightarrow (C(step(m)(x)) < C(x)). \\
\equiv &\ \forall n \in [N],\ (x[n] = 0) \vee \exists m \in [N](P(x, x', m) \wedge x'[m] < x[m]). \\
\textit{Noninterference}: &\ \forall i, j \in [N],\ P(x, x', i) \wedge P(x', x'', j) \wedge P(x', x''', i) \\
&\ \Rightarrow (x'[i] < x[i] \wedge x'''[i] < x[i]). \qquad \text{(decreasing measure)} \\
\textit{Minimality}: &\ C(x) = 0^N \Rightarrow x \in X^*.
\end{aligned}$$

From the above conditions, using Theorem 2, N_0 is calculated to be 21.

5 Experiments and Discussion

We verified that instances of the aforementioned systems with sizes less than the small model parameter N_0 satisfy the four conditions(*invariance, progress, non-interference, minimality)* of Theorem 1 using the Z3 SMT-solver [4]. The models are checked by symbolic execution.

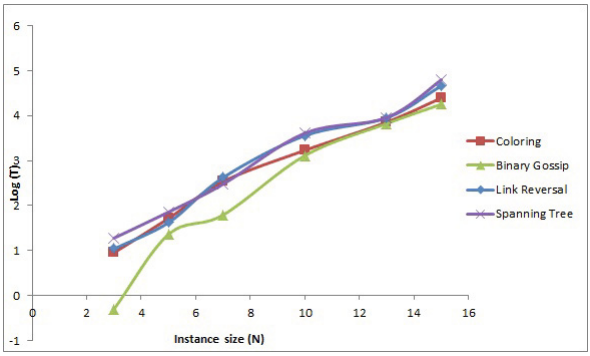

Fig. 1. Instance size vs $\log_{10}(T)$, where T is the running time in seconds

The interaction graphs were complete graphs in all the experiments. In Figure 5, the x-axis represents the problem instance sizes, and the y-axis is the log of the running time (in seconds) for verifying Theorem 1 for the different algorithms. [2]

We observe that the running times grow rapidly with the increase in the model sizes. For the binary gossip example, the program completes in $\sim$ 17 seconds for a model size 7, which is the N_0 value. In case of the link reversal, for a model size 13, the program completes in $\sim$ 30 mins. We have used complete graphs in all our experiments, but as we mentioned earlier in Section 3.2, we can encode more general graphs as well. This method is a general approach to automated verification of stabilization properties of distributed algorithms under specific fairness constraints, and structural constraints on graphs. The small model nature of the conditions to be verified is crucial to the success of this approach. We saw that many distributed graph algorithms, routing algorithms and symmetry-breaking algorithms can be verified using the techniques discussed in this paper. The problem of finding a suitable measure function which satisfies Theorem 2, is indeed a non-trivial one in itself, however, for the problems we study, the natural measure function of the algorithms seems to work.

References

1. Dolev, S.: Self-stabilization. MIT Press (2000)
2. Tsitsiklis, J.N.: On the stability of asynchronous iterative processes. Mathematical Systems Theory 20(1), 137–153 (1987)
3. Johnson, T.T., Mitra, S.: A small model theorem for rectangular hybrid automata networks. In: Giese, H., Rosu, G. (eds.) FMOODS/FORTE 2012. LNCS, vol. 7273, pp. 18–34. Springer, Heidelberg (2012)
4. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)

[2] The code is available at http://web.engr.illinois.edu/rghosh9/code.

5. Dijkstra, E.W.: Self-stabilization in spite of distributed control. In: Selected Writings on Computing: A Personal Perspective, pp. 41–46. Springer (1982)
6. Theel, O.: Exploitation of ljapunov theory for verifying self-stabilizing algorithms. In: Herlihy, M. (ed.) DISC 2000. LNCS, vol. 1914, pp. 209–222. Springer, Heidelberg (2000)
7. Oehlerking, J., Dhama, A., Theel, O.: Towards automatic convergence verification of self-stabilizing algorithms. In: Tixeuil, S., Herman, T. (eds.) SSS 2005. LNCS, vol. 3764, pp. 198–213. Springer, Heidelberg (2005)
8. Theel, O.E.: A new verification technique for self-stabilizing distributed algorithms based on variable structure systems and lyapunov theory. In: HICSS (2001)
9. Dhama, A., Theel, O.: A tranformational approach for designing scheduler-oblivious self-stabilizing algorithms. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 80–95. Springer, Heidelberg (2010)
10. Ghosh, S.: Distributed systems: an algorithmic approach. CRC Press (2010)
11. Umeno, S., Lynch, N.: Safety verification of an aircraft landing protocol: A refinement approach. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 557–572. Springer, Heidelberg (2007)
12. Johnson, T.T., Mitra, S.: Invariant synthesis for verification of parameterized cyber-physical systems with applications to aerospace systems. In: Proceedings of the AIAA Infotech at Aerospace Conference (AIAA Infotech 2013), Boston, MA. AIAA (August 2013)
13. Allen Emerson, E., Kahlon, V.: Reducing model checking of the many to the few. In: McAllester, D. (ed.) CADE-17. LNCS (LNAI), vol. 1831, pp. 236–254. Springer, Heidelberg (2000)
14. Duggirala, P.S., Mitra, S.: Abstraction refinement for stability. In: 2011 IEEE/ACM International Conference on Cyber-Physical Systems (ICCPS), pp. 22–31. IEEE (2011)
15. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and reasoning about systems. Cambridge University Press (2004)
16. Mitra, S.: A verification framework for hybrid systems. PhD thesis, Massachusetts Institute of Technology (2007)
17. Khalil, H.K., Grizzle, J.W.: Nonlinear systems, vol. 3. Prentice Hall, Upper Saddle River (2002)
18. Dershowitz, N.: Termination of rewriting. Journal of Symbolic Computation 3(1), 69–115 (1987)
19. Gafni, E.M., Bertsekas, D.P.: Distributed algorithms for generating loop-free routes in networks with frequently changing topology. IEEE Transactions on Communications 29(1), 11–18 (1981)