DOI 10.1007/s00165-010-0150-7
BCS © 2010
Formal Aspects of Computing (2011) 23: 649–679

Formal Aspects of Computing

Verification of distributed systems with local–global predicates

K. Mani Chandy[1], Brian Go[1], Sayan Mitra[2], Concetta Pilotto[1] and Jerome White[1]

[1] California Institute of Technology, Pasadena, USA.
E-mails: mani@cs.caltech.edu; bgo@cs.caltech.edu; pilotto@cs.caltech.edu; jerome@cs.caltech.edu
[2] University of Illinois at Urbana Champaign, Urbana, USA. E-mail: mitras@crhc.uiuc.edu

Abstract. This paper describes a methodology for developing and verifying a class of distributed systems in which the state space may be discrete or continuous. Our focus is on systems where changes are local in that a small number of components change state while the remainder of the system is unchanged. A proof methodology is developed that ensures global properties, such as invariants and convergence, by guaranteeing local properties within subsystems. This methodology is used to prove the correctness of concrete examples. We present a PVS library of theorems and proofs that can be used to reduce the work required to develop and verify programs in this class. A transformation of these libraries to Java is also outlined.

Keywords: Concurrency · Convergence · Local–global · Stability · Theorem prover · Verification

1. Introduction

Verifying correctness of distributed systems is difficult due to the large state space of the system and inherent nondeterminism. Model checkers and test suites are intractable when the system has unbounded data structures, such as stacks or queues, or has real-valued variables. Systems, such as sensor networks and autonomous vehicle systems have continuous state spaces and continuous transitions that are not easily represented by model checkers. This paper presents an approach to verification of distributed systems, with continuous or discrete state spaces, that addresses some of these challenges.

Some progress properties are expressed as either convergence to, or termination in, a particular system state. Informally, a system converges to a desired value if the state gets arbitrarily close to the desired value as time, or the number of steps in a computation, tends to infinity [Lue79]. Termination is a related property which states that the desired state is reached in a finite number of steps. These properties are formalized using Linear Temporal Logic

Correspondence and offprint requests to: J. White, E-mail: jerome@cs.caltech.edu
This paper is dedicated to Brian Go, 1988–2009.

[Pnu77]. Proof obligations regarding convergence appear in many continuous state problems, including distributed control algorithms for tracking [OS07], pattern formation [OSFM07] and flocking [BHOT05], while proof obligations about termination appear in many distributed computing problems such as consensus [Lyn96, AW04]. The *model-based* approach [Tau05, Hor03] develops algorithms at an abstract mathematical level and then refines models to actual implementations in a provably correct manner by a process known as *stepwise refinement*. A great deal of work has been carried out on stepwise refinement over almost 40 years including sequential programming [Dij68, Dij76, Wir71, Mor87], and concurrent systems [LT87, Spi08, Spi07, Abr96, BS96, CM88, LS91].

The contribution of this paper is to propose a verification method for convergence or termination of continuous or discrete-state systems. This builds upon the earlier work on reuse and refinement. Our method uses properties of interactions among small numbers of agents, or between an agent and a message channel, to prove global properties of systems—we define these properties as *local–global*. We use local–global properties in conjunction with standard techniques, such as proving invariants and demonstrating the existence of variant functions or Lyapunov functions [CS77, BHOT05, Lib03], for proving correctness of distributed systems. As an example of such a system, consider a collection of analog sensors that use continuous dynamics to converge to the average of their initial values [JLM03]. A faulty mechanism within the system may partition the set of sensors into multiple connected components. In this case, the dynamics of each connected component is independent of the dynamics of other connected components. However, by ensuring that the dynamics for each connected component maintains the average and reduces the variance of that component, we ensure that the global dynamics maintains the average and reduces the variance of the entire system. Furthermore, local–global properties are used to prove that if the sensor network is not permanently partitioned (though it can be temporarily partitioned) then the mean values for all sensors in the network converge to the global mean. The sensor network is an example of a larger class of problems in which local–global properties are used to verify systems. Our methodology is also applied to *message passing* systems in which messages may be lost, delayed and reordered. As an example we present a message passing algorithm, and its proof of correctness, for solving systems of linear equations that extends the work of [GM80, CM69, CMP08].

We encode theorems, specifications, and property abstractions in the theorem prover PVS [ORS92], developing refinement libraries for proving convergence and termination of distributed systems. Our general theory for convergence and termination builds on the PVS formalization of I/O automata [LT89, AHS98] and its extensions to timed and hybrid I/O automata [KLSV06, MA05, Mit07]. These theories formalize reachable states, invariant properties, acceptance conditions, and abstraction relations. Convergence and termination of general sequences has been formalized in the PVS and Isabelle libraries for differential calculus [Got00], real-analysis [Har98], and topology [Les], along with programs and recursive functions [BKN07, RP96]. Proof hierarchies within a theorem prover has been studied for specific cases [Jac00, dJvdPH00, MB97].

Finally, our PVS library is transformed into a library of Java classes. Using our methodology, we derive correctness of our programs from the correctness of the PVS specification. Our proof obligation is to show that operators implemented in Java are equivalent to their corresponding PVS operators. For example, a step from PVS to Java might be the implementation of the arithmetic max. In this case, our proof obligation is to show that the function max defined in PVS and implemented in Java are equivalent. We employ JML [BCC+05] to help with such checks.

The paper is organized as follows. Section 2 presents basic definitions for automata and proves sufficient conditions for termination and convergence. Section 3 provides a formal definition of local–global properties and how they are used to prove correctness. Section 4 applies local–global properties to two applications from control theory, proving their termination and convergence, respectively. Section 5 extends our theory to message passing systems with bounded delay. Section 6 discusses our PVS meta-theory for convergence and termination. Section 7 presents a Java implementation of our examples. We conclude with Sect. 8.

2. Preliminary theory

In this section we provide basic definitions for automata, reachability, stability, and convergence and state sufficient conditions for proving the latter properties. We refer to [AHS98, Arc00, LKLM05, GMPJ09] for their corresponding PVS meta-theories.

2.1. Automata, executions, and invariants

We denote the set of Boolean constants by $\mathbb{B} = \{true, false\}$, the set of natural numbers by $\mathbb{N} = \{0, 1, \ldots, \}$, and the set of reals by $\mathbb{R}$. For a set A, A^ω is defined as the set of infinite sequences of elements in A indexed by $\mathbb{N}$. We denote by $[N]$ with $N \in \mathbb{N}$ the set $\{0, 1, \ldots N-1\}$; $n, m \in \mathbb{N}$ denote arbitrary natural numbers.

An automaton $\mathscr{A}$ is a nondeterministic state machine or a labeled transition system defined as follows.

Definition 1 An *automaton* $\mathscr{A}$ is a quintuple consisting of:

(a) a nonempty set of states S,
(b) a nonempty set of actions A,
(c) a nonempty set of start states $S_0 \subseteq S$,
(d) an enabling predicate $E : [S, A \to \mathbb{B}]$, and
(e) a transition function $T : [S, A \to S]$.

For $s \in S$ and $a \in A$, $E(s, a)$ holds if and only if the transition labeled by a can be applied to s. In this case, a is said to be *enabled* at s. At any state s, multiple actions may be enabled. However, once an enabled action a is fixed the post-state of the transition s' is uniquely determined. Specifically, $s' = T(s, a)$, if a is enabled at s. The set of actions can be uncountably infinite. We refer the reader to [LKLM05] for models of timed and hybrid systems in this formalism.

Hereafter, $\mathscr{A}$ denotes an automaton with parameters $\langle S, A, S_0, E, T\rangle$; $s, \hat{s} \in S$, $s_0 \in S_0$ denote arbitrary (initial) states of the automaton; $\hat{S}$ a nonempty subset of S; $a \in A$ an arbitrary action of the automaton. The semantics of an automaton $\mathscr{A}$ are defined in terms of its executions. An *execution fragment* of $\mathscr{A}$ is a possibly infinite alternating sequence of states and actions $s_0, a_0, s_1, a_1, s_2, \ldots$, such that for each i, $E(s_i, a_i)$ holds and $s_{i+1} = T(s_i, a_i)$. An execution fragment is an *execution* if s_0 is a starting state. The *length* of a finite execution is the number of actions it contains. Given an execution, we denote by $s^{(n)}$ the state of the automaton after n steps of that execution. Finally, an execution is *complete* if and only if (a) the execution is infinite, or (b) the execution is finite and terminates in a state in which no action is enabled.

Given two states s_i, s_j, we say that s_i is *reachable* from s_j, denoted by $reach(s_i, s_j)$, if there exists a finite, possibly empty, sequence of actions $a_0, a_1, \ldots, a_n$ which when applied to s_j, result in s_i. We denote $reach(s, s_0)$ with $s_0 \in S_0$ by *reachable*(s). We define *ReachFrom*(S_i) with $S_i \subseteq S$ as the set of states reachable from states in S_i; more formally,

$$\mathit{ReachFrom}(S_i) = \{s : \exists s_1 \in S_i \; reach(s, s_1)\}.$$

The following theorem formalizes Floyd's induction principle for proving invariants predicates.

Theorem 2.1 (Floyd's Induction Principle [Flo67]) *Let $G : [S \to \mathbb{B}]$ be a predicate on S. Then G is invariant if the following holds:*

A1. $\forall s \in S_0$*:* $G(s)$*, and*
A2. $\forall s \in S, a \in A$*:* $G(s) \wedge E(s, a) \Rightarrow G(T(s, a))$.

This theorem has been employed for verifying safety properties of untimed [AHS98], timed [LKLM05], and hybrid automata [UL07]. Features of this verification method that make it attractive are: (a) It suffices to check that the predicate G is preserved over individual actions, and hence, the check breaks down into a case analysis of actions. (b) This structure facilitates partial automation of proofs using customized proof strategies [Arc00].

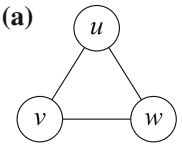
When proving convergence or termination, we consider executions in which certain important actions occur (infinitely often). Fairness conditions give us a way of restricting the class of executions.

Definition 2 A *fairness condition* $\mathcal{F}$ for the set of actions A is a finite collection $\{F_i\}_{i=1}^n$, where each F_i is a nonempty subset of A. An infinite sequence of actions $a_0, a_1, \ldots \in A^\omega$ is $\mathcal{F}$-fair if

$$\forall F \in \mathcal{F}, \; \forall n, \; \exists m, \; m > n, \; \text{such that } a_m \in F.$$

An infinite execution $\alpha = s_0, a_0, s_1, a_1, \ldots$ is said to be $\mathcal{F}$-fair if the corresponding sequence of actions $a_0, a_1, \ldots$ is $\mathcal{F}$-fair.

An execution is not $\mathcal{F}$-fair if there exists $F \in \mathcal{F}$ such that no action from F ever appears in some suffix of that execution. Most models of distributed systems assume weak fairness whereby certain actions are executed infinitely often. The model presented in Definition 2 is weaker. We use this model because we wish to prove



(a)

(b) $\mathcal{F} = \Big\{ \underbrace{\{\{u,v\},\{u,w\}\}}_{F_1}, \underbrace{\{\{u,v\},\{v,w\}\}}_{F_2}, \underbrace{\{\{u,w\},\{v,w\}\}}_{F_3} \Big\}$

$\pi = \Big\{\{u,v\},\{v,w\}\Big\}$

(c) $\mathcal{F}_w = \Big\{\{u,v\},\{u,w\},\{v,w\}\Big\}$

$\pi_w = \Big\{\{u,v\},\{v,w\},\{u,w\}\Big\}$

Fig. 1. A system consisting of three agents, u, v, and w. The fairness criteria is that the system is never permanently partitioned into non-communicating subsets. Note that any computation, where the communication links in π occur infinitely often, is not allowed in weak fairness. **a** Communication links of the system. **b** The fairness condition $\mathcal{F}$, and a set of communication links π occurring infinitely often according to Definition 2. Although there are only two communication links in π, all $F \in \mathcal{F}$ are represented. **c** The fairness condition $\mathcal{F}_w$, and a set of communication links π_w occurring infinitely often according to weak fairness

properties of networks of agents that are never partitioned permanently. For example, consider a network of three agents u, v and w (see Fig. 1a). Communication links between any pair of agents may be created and deleted in a nondeterministic manner. The network is permanently partitioned if an agent is permanently cut off from the others. For example, the network is not permanently partitioned if a communication link between u and v exists infinitely often, and a link between v and w exists infinitely often, even though there is never a link between u and w (see Fig. 1b). A fairness criterion that specified that every communication link exists infinitely often (Fig. 1c) is too strong because it would rule out computations in which there is never a link between u and w. The fairness criterion that specifies that u is not permanently cut off from the other agents is that at least one of the links $\{u, v\}$ or $\{u, w\}$ exists infinitely often.

2.2. Stability, convergence, and termination

The concepts of stability, convergence and termination have been introduced by Lyapunov [Lya66] and used for describing the behavior of dynamical systems. In this subsection, we formalize these concepts and define stability, convergence and termination of automata. In [Tsi87] sufficient conditions for stability and convergence have been provided for dynamical systems. Our contribution is to extend these conditions to automata. In [MC08], the authors present an analogous formalization together with the corresponding PVS implementation.

2.2.1. Topology

Stability, convergence, and termination of $\mathscr{A}$ to a state $\hat{s}$ are defined with respect to a topological structure around $\hat{s}$.

Definition 3 For $\epsilon > 0$ and s the *ϵ-ball around s* is the set

$$B_\epsilon(s) := \{s_1 \in S \mid d(s_1, s) \leq \epsilon\}. \tag{1}$$

where $d : [S, S \to \mathbb{R}_{\geq 0}]$ is a *distance function* for $\hat{s}$ satisfying $\forall s \neq \hat{s} : \; d(\hat{s}, s) > d(\hat{s}, \hat{s})$.

The ϵ-balls around a given state s define a topological structure around s.

2.2.2. Stability

Informally, an equilibrium state $\hat{s}$ is stable for $\mathscr{A}$ if every execution fragment that starts close to $\hat{s}$ remains close to $\hat{s}$, where closeness is defined in terms of the ϵ-balls around $\hat{s}$. Examples of stable executions are depicted in Fig. 2.

Definition 4 $\hat{s}$ is *stable* for $\mathscr{A}$ if

$$\forall \epsilon > 0, \exists \delta > 0, \quad \mathit{ReachFrom}(B_\delta(\hat{s})) \subseteq B_\epsilon(\hat{s}).$$

From this definition, it follows that $\delta \leq \epsilon$. Note that stability is independent of the starting states of the automaton. For a nonempty subset of states $\hat{S}$, the definitions for the ϵ-balls around $\hat{S}$ and stability are analogous to Definitions 3 and 4.

Sufficient conditions for stability. Let f be a function from S to a totally ordered set P. The range of f is denoted by *Range*(f), and its *p-sublevel set* is defined as $L_p := \{s \mid f(s) \leq p\}$. The following theorem gives a sufficient condition for proving stability of an automaton in terms of a Lyapunov-like function.

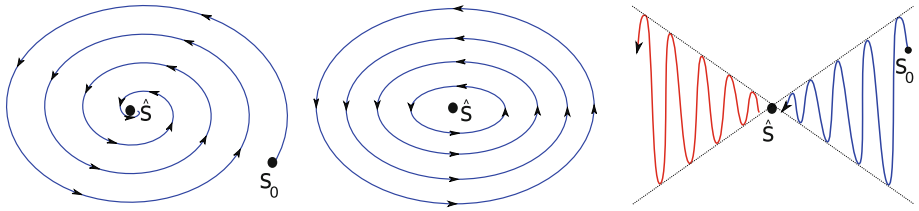


Fig. 2. Stable and convergent (*left*), stable and non-convergent (*middle*) and convergent and unstable (*right*) executions. In the last case, the execution from s_0 converges, but executions starting from the left neighborhood of $\hat{s}$ diverge

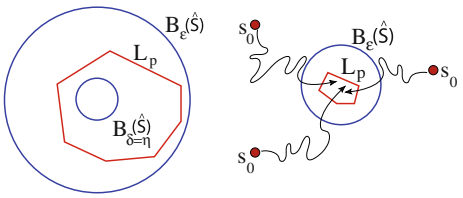


Fig. 3. A graphical representation of the proof of Theorem 2.2 (*left*) and of Theorem 2.3 (*right*)

Theorem 2.2 *If there exists* $f : [S \to P]$ *that satisfies the following conditions:*

B1. $\forall \epsilon \geq 0, \exists p \in P$, *such that* $L_p \subseteq B_\epsilon(\hat{S})$.

B2. $\forall p \in P, \exists \epsilon \geq 0$, *such that* $B_\epsilon(\hat{S}) \subseteq L_p$.

B3. $\forall s, a\ E(s, a) \Longrightarrow f(T(s, a)) \leq f(s)$.

Then $\hat{S}$ *is stable for* $\mathscr{A}$.

B1 requires that every ϵ-ball around $\hat{S}$ contains a p-sublevel set L_p. B2 is symmetric—it requires that every sublevel set contains an ϵ-ball. B3 states that the value of the function f does not increase if an action a is applied to state where it is enabled.

Proof. Let us fix an $\epsilon > 0$. We have to show that there exists a $\delta > 0$, such that any execution fragment that starts in $B_\delta(\hat{S})$ remains within $B_\epsilon(\hat{S})$.

From B1, $\exists p \in P : L_p \subseteq B_\epsilon(\hat{S})$, and from B2, $\exists \eta \geq 0 : B_\eta(\hat{S}) \subseteq L_p \subseteq B_\epsilon(\hat{S})$. We refer to Fig. 3 for a pictorial representation. Set $\delta = \eta$. From B3, $\mathit{ReachFrom}(L_p) = L_p$. Since $B_\delta(\hat{S}) \subseteq L_p$, we get that $\mathit{ReachFrom}(B_\delta(\hat{S})) \subseteq L_p \subseteq \mathit{ReachFrom}(B_\epsilon(\hat{S}))$. □

2.2.3. Convergence

An automaton $\mathscr{A}$ *converges* to $\hat{s}$, if for every fair infinite execution of $\mathscr{A}$, the automaton gets closer to $\hat{s}$. See Fig. 2 for examples of convergent and divergent executions.

Definition 5 $\mathscr{A}$ *converges* to $\hat{s}$, if $\forall \epsilon > 0 : \Diamond\Box\ s \in B_\epsilon(\hat{s})$.

For a nonempty subset of states $\hat{S} \subseteq S$, the definition of convergence to $\hat{S}$ is analogous to Definition 5.

Sufficient conditions for convergence. The next theorem gives sufficient conditions for proving convergence of automaton $\mathscr{A}$ in terms of a Lyapunov-like function. Let $\mathcal{P}$ be the set of values in P that are reached eventually: $\mathcal{P} := \{p \in P \mid \Diamond\ (s \in L_p)\}$.

Theorem 2.3 *Suppose there exists a totally ordered set* $(P, <)$ *and a function* $f : S \to P$ *that satisfy the following conditions:*

C1. $\forall p, q \in P, p < q \Longrightarrow L_p \subsetneq L_q$.

C2. $\forall \epsilon > 0, \exists p \in P,$ *such that* $L_p \subseteq B_\epsilon(\hat{S})$.

C3. $\forall s,\ a,$ $(reachable(s) \ \wedge \ E(s, a)) \Longrightarrow f(T(s, a)) \leq f(s)$.

C4. $\forall p \in P, L_p \neq \hat{S}$ *implies* $\exists F \in \mathcal{F},$ *such that* $\forall a \in F, s \in L_p, reachable(s) \Longrightarrow (E(s, a) \wedge f(T(s, a)) < f(s))$.

C5. $\forall P' \subseteq \mathcal{P},$ *if* P' *is lower bounded then it has a smallest element.*

Then $\mathscr{A}$ *converges to* $\hat{S}$

Some remarks about the hypothesis of the theorem are in order. C1 implies that every sublevel set of the function f is distinct. C2 requires that for any $\epsilon > 0$, there exists a p-sublevel set of f that is contained within the ϵ-ball around $\hat{S}$. This is identical to condition B1. C3 requires that the function f is nonincreasing over all transitions from reachable states. This is a weaker version of B3. C4 requires that for any sublevel set L_p that is not equal to the convergence set $\hat{S}$, there exists a fair set of actions $F \in \mathcal{F}$, such that any action $a \in F$ strictly decreases the value of f—possibly by some arbitrarily small amount. C5 requires that every lower-bounded subset of $\mathcal{P}$ has a smallest element. This is a weaker assumption than requiring $\mathcal{P}$ to be well-ordered. Instead of C5 it is sometimes easier to prove that the set $\mathcal{P}$ is well-ordered.

Before proving Theorem 2.3, we state an intermediate lemma used in the proof.

Lemma 2.4 f *satisfies C1–5* $\Longrightarrow$ $Range(f) = \mathcal{P}$.

This lemma follows directly from conditions C1, 4, and 5 on f.

Proof of Theorem 2.3. Let us fix $\epsilon \geq 0$, and f satisfying the conditions in the hypothesis.

By C2, $\exists p \in Range(f) : L_p \subseteq B_\epsilon(\hat{S})$. By Lemma 2.4, $\Diamond(s \in L_p)$, see Fig. 3. By C3, $ReachFrom(L_p) = L_p$. Hence, $\Diamond\Box\ s \in L_p$, which implies $\Diamond\Box\ s \in B_\epsilon(\hat{S})$. $\Box$

In certain applications the function d which defines the topological structure around $\hat{s}$ can itself be used as the Lyapunov-like function for proving convergence. In which case C2 follows automatically.

Corollary 2.5 *Let* $\hat{S}$ *be a nonempty subset of* S *and* $\mathcal{F}$ *be a fairness condition on* A. *We define* $f : S \to \mathbb{R}_{\geq 0}$ *as* $f(s) := d(\hat{S}, s)$. *Suppose there exists a strictly decreasing sequence* $p_0, p_1, \ldots \in {\mathbb{R}_{\geq 0}}^\omega$ *of valuations of* f *that converges to* 0, *such that:*

D1. $\forall i, j \in \mathbb{N} + 1, i > j \Rightarrow L_{p_i} \subsetneq L_{p_j}$.

D2. $\forall s,\ a, i \in \mathbb{N}$ $(reachable(s) \ \wedge \ E(s, a) \ \wedge \ s \in L_{p_i}) \Rightarrow T(s, a) \in L_{p_i}$.

D3. $\forall i \in \mathbb{N}, p_i \neq 0$ *implies* $\exists F \in \mathcal{F}$ *such that* $\forall a \in F, s \in L_{p_i}, reachable(s) \Rightarrow (E(s, a) \ \wedge \ T(s, a) \in L_{p_{i+1}})$.

Then $\mathscr{A}$ *converges to* $\hat{S}$.

2.2.4. Termination

An automaton $\mathscr{A}$ *terminates* in $\hat{s}$, if for every fair execution of $\mathscr{A}$, in a finite number of steps the automaton reaches and remains in $\hat{s}$.

Definition 6 $\mathscr{A}$ *terminates* in $\hat{s}$, if $\Diamond\Box\ s = \hat{s}$.

For a nonempty subset of states $\hat{S} \subseteq S$, the definition of termination in $\hat{S}$ is analogous to Definition 6.

Sufficient conditions for termination. The next theorem gives sufficient conditions for proving termination of automaton $\mathscr{A}$ in terms of a Lyapunov-like function.

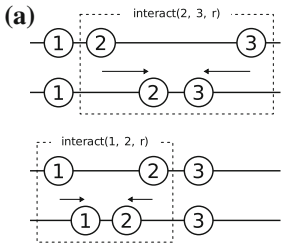
Theorem 2.6 *Let* $\hat{S}$ *be a nonempty subset of* S, *and* $\mathcal{F}$ *be a fairness condition on* A. *We define* $f : S \to \mathbb{R}_{\geq 0}$ *as* $f(s) := d(\hat{S}, s)$. *Suppose there exists a strictly finite decreasing sequence* $p_0, p_1, \ldots p_n \in \mathbb{R}_{\geq 0}$ *of valuations of* f *that reaches* $p_n = 0$, *such that:*

E1. $\forall i, j \in [n + 1], i > j \Rightarrow L_{p_i} \subsetneq L_{p_j}$.

E2. $\forall s,\ a, i \in [n + 1]$ $(reachable(s) \ \wedge \ E(s, a) \ \wedge \ s \in L_{p_i}) \Rightarrow T(s, a) \in L_{p_i}$.

E3. $\forall i < n : p_i \neq 0$ *implies* $\exists F \in \mathcal{F},$ *such that* $\forall a \in F, s \in L_{p_i}, reachable(s) \Rightarrow (E(s, a) \ \wedge \ T(s, a) \in L_{p_{i+1}})$.

Then $\mathscr{A}$ *terminates in* $\hat{S}$.



(b)
$\mathbf{L}$: Type := $\{l : real \mid 0 < l \wedge l \leq 0.5\}$
L : Const $\mathbf{L}$
S : Type := array $[I \mapsto \mathbb{R}_{\geq 0}]$
s_0 : Const S
$S_0(s : S) : bool := (s = s_0)$

A : Datatype $interact(i, j : I, r : [L, 1 - L])$

$E(s : S, a : A) : bool :=$ true

$T(s : S, a : A) : S :=$ [Case $a \equiv interact(i, j, r)$:
s With [
$(i) := s[i] + r(s[j] - s[i])$
$(j) := s[j] - r(s[j] - s[i])$]]

Fig. 4. Representations of the line convergence algorithm. **a** Two transitions in which agents atomically interact and update their location. **b** A formal specification

3. Verification of concurrent systems using local–global relations

In this section we present our methodology as applied to concurrent systems. We introduce local–global relations, a key part of our methodology. These relations describe properties of the global state of the system that are maintained when local parts of the system interact. They are used to prove system safety and progress properties.

3.1. Automaton formulation

Definition 7 A *concurrent system* is an automaton with additional structure:

(a) The system has N agents, where N is a positive integer. The agents are indexed j for $0 \leq j < N$.

(b) A state $s \in S$ is an array with N elements, where $s[j]$ is the state of the j-th agent.

We denote by $s[J]$, $J \subseteq [N]$ the subarray of s inclusive of indices in J in some predefined order. Associated with each action $a \in A$ is a nonempty set of agents whose states can be read or modified by the occurrence of action a. We denote this set of agents by $agents(a)$. We use the notation s_a for $s[agents(a)]$. The enabled predicate, E, depends only on the states of the agents that participate in the action. We define an enabling condition e for an action a as $\forall s, a : e(s_a, a) = E(s, a)$. The transition function T depends only on the states of the agents that participate in the action. We define the transition function t of an action a as $\forall s, a : t(s_a, a) = s'_a$ where $s' = T(s, a)$ where the pre-states, s, and post-states, s', are restricted to the agents that participate in the action.

An example of a concurrent system is the following.

Example 3.1 We model a distributed algorithm in which a set of N agents start at arbitrary positions on a line and, through interactions, converge to a point. Specifically, any two agents may interact at any time; when they do, their positions are atomically updated so that they move towards one another to reduce the distance between them by the same amount r. The value of $r \in [L, 1 - L]$, where L is a real constant such that $2L \in (0, 1)$. The constant $2L$ is the lower bound on the overall improvement of the two agents. An iteration of this algorithm can be seen in Fig. 4a.

Figure 4b provides concrete definitions for the types and the functions for modeling this protocol as an automaton. Briefly, I is the set of natural numbers from to 0 to $N - 1$. The states type, S, is an array of $\mathbb{R}_{\geq 0}$ indexed by I. The state s_0 is an arbitrary but constant element of S and denotes the initial starting state. The real value r regulates the distance a given agent travels. Note that $r \in [L, 1 - L]$.

In this example, two agents can always interact, because the enabling predicate, $E(s, a)$, is always true. For action a of the form $interact(i, j, r)$, the state transition function $T(s, a)$ returns a state s' that is identical to s except that the i^{th} and the j^{th} values of s' are $s[i] + r(s[j] - s[i])$ and $s[j] - r(s[j] - s[i])$, respectively. This update rule works both if $s[i] < s[j]$ and vice-versa. If $s[i] < s[j]$, then the difference $s[j] - s[i]$ is positive; in this case, the $interact$ action adds a positive value to $s[i]$ and subtracts the same amount from $s[j]$. In the other case ($s[i] \geq s[j]$), $s[j] - s[i] \leq 0$, hence, the action subtracts a non-negative value to $s[i]$ and adds the same amount to $s[j]$.

3.2. Local–global Relations

This subsection introduces a relation called local–global which is a central concept for our methodology. A local–global relation is one that is preserved throughout the execution by the entire concurrent system (globally) provided it is preserved by the individual agent interactions (locally). Because our system is represented as an array, we restrict attention to local–global relations that deal with equal-sized arrays. A local–global relation holds for two equal-sized arrays if it holds for all their equal-sized subarrays. This subsection provides sufficient conditions for proving that certain relations are local–global; the next subsection applies local–global relations to concurrent systems. In Sects. 4 and 5 we show how local–global relations can aid in verifying convergence and termination of specific systems.

For the remainder of this section x and x' are equal-sized nonempty arrays of the same type. Let M be the size of the arrays, and let the array index be j for $0 \leq j < M$. Hereafter J refers to a nonempty set of indices of array x or x'. Denote by $\bar{J}$ the complement of J. Recall that $x[J]$ is the subarray of the indices in x inclusive of the values in J, based on some predefined order. For example, consider $M = 5$ and $J = \{0, 2, 3\}$ then $\bar{J} = \{1, 4\}$, $x[J] = [x[0], x[2], x[3]]$ and $x[\bar{J}] = [x[1], x[4]]$.

A formal definition is as follows

Definition 8 Let R be a binary relation on equal-sized arrays of agent states. We define R to be a *local–global relation* if and only if:

LG1. R is transitive

LG2. $\forall x, x', J : \Big(x[J]\ R\ x'[J]\Big) \ \wedge\ \Big(x[\bar{J}] = x'[\bar{J}]\Big) \Rightarrow (x\ R\ x')$

Example 3.2 Local–global relations that we use in proving programs include the following. $\forall x, x' : x\ R\ x'$ holds exactly when

1. In the case where the elements of x are positive integers:

   $$x\ R\ x' \equiv gcd(x) = gcd(x')$$

   where gcd is the greatest common divisor. Two equal sized arrays satisfy the relation R if their gcds are the same. The relation R is local–global because it satisfies both conditions of Definition 8: equality is transitive; and for all sets of indices J, if $x[J]$ and $x'[J]$ have the same gcd, while indices in $\bar{J}$ remain the same, then $\text{gcd}(x) = \text{gcd}(x')$.
2. In the case where the elements of x are in a partial ordering $\leq$:

   $$x\ R\ x' \equiv \min(x) \leq \min(x')$$

   where $\min(x)$ is the minimum element in the array x. In this case, two equal sized arrays satisfy the relation R if the minimum element in the first array is less-than, or equal to, the minimum element in the second. The relation R is local–global because $\leq$ transitive, and for all sets of indices J, if $\min(x[J]) \leq \min(x'[J])$, while indices in $\bar{J}$ remain the same, then $\min(x) \leq \min(x')$.

Not all relations satisfy our local–global definition. Consider a relation R defined as

$$x\ R\ x' \equiv \min(x) < \min(x')$$

Then R is *not* a local–global relation, as illustrated by the case $x = [2, 0]$, $x' = [1, 0]$, and $J = \{0\}$.

The operators presented in this example are instances of a larger class of binary operators that obey our local–global relation. For this reason we define a generic operator $\circ$, along with a composition function fold.

Definition 9 Let $\circ$ be an associative, commutative, dyadic operator on some type. For an array x with $M > 0$ elements define fold $\circ\ x$ as:

$$\text{fold}\ \circ\ x = \begin{cases} x[0] & \text{if } M = 1, \\ x[0] \circ x[1] \circ \ldots \circ x[M-1] & \text{otherwise.} \end{cases}$$

The following theorem gives sufficient conditions for when this binary composability obeys our local–global relation.

Theorem 3.3 *The relation,* $x\ R\ x' \equiv \big((\text{fold}\ \circ\ x) = (\text{fold}\ \circ\ x')\big)$*, is a local–global relation if* $\circ$ *is associative and commutative.*

Proof. For all x, x', are equal-sized arrays of some type, and $J \subsetneq [M]$

$$\begin{aligned}\text{fold}\ \circ\ x &= (\text{fold}\ \circ\ x[J]) \circ (\text{fold}\ \circ\ x[\bar{J}]) \\ &= (\text{fold}\ \circ\ x'[J]) \circ (\text{fold}\ \circ\ x'[\bar{J}]) \\ &= \text{fold}\ \circ\ x'\end{aligned}$$

□

Example 3.4 The following examples are instances of fold that, by Theorem 3.3, are local–global relations:

- Where $\circ$ is +, and the elements of x are reals,

  $$x\ R\ x' \equiv \sum_j x'[j] = \sum_j x[j].$$

  This follows since addition is associative and commutative.
- Where $\circ$ is min, and the elements of x are in a partial order,

  $$x\ R\ x' \equiv \min_j x'[j] = \min_j x[j].$$

  This follows since min is associative and commutative.

In addition to Theorem 3.3, local–global relations can be constructed if $\circ$ is associative, commutative, and monotonic.

Definition 10 Let u, u' and v be members of a partial ordering $\leq$. The operator $\circ$ is *monotonic* exactly when:

$$\forall u, u', v:\ u \leq u' \Rightarrow u \circ v \leq u' \circ v$$

The operator $\circ$ is *strictly monotonic* exactly when:

$$\forall u, u', v:\ u < u' \Rightarrow u \circ v < u' \circ v$$

Likewise, monotonicity must be maintained under composition.

Theorem 3.5 *Consider arrays* x *and* x' *whose elements are members of the partial ordering* $\leq$*, then*

1. $x\ R\ x' \equiv (\text{fold}\ \circ\ x)\ \leq\ (\text{fold}\ \circ\ x')$ *is a local–global relation if* $\circ$ *is associative, commutative and monotonic.*
2. $x\ R\ x' \equiv (\text{fold}\ \circ\ x) < (\text{fold}\ \circ\ x')$ *is a local–global relation if* $\circ$ *is associative, commutative and strictly monotonic.*

Proof. For all x, x', are equal-sized arrays of some type, and $J \subsetneq [M]$

$$\begin{aligned}\text{fold}\ \circ\ x &= (\text{fold}\ \circ\ x[J]) \circ (\text{fold}\ \circ\ x[\bar{J}]) \\ &\leq (\text{fold}\ \circ\ x'[J]) \circ (\text{fold}\ \circ\ x'[\bar{J}]) \\ &= \text{fold}\ \circ\ x'\end{aligned}$$

The proof for the strictly monotonic operator is similar. □

Example 3.6 The following examples are instances of fold that, by Theorem 3.5, are local–global relations:

- In the case where the elements of x are reals, since + is associative, commutative and strictly monotonic:

  $$x\ R\ x' \equiv \sum_j x'[j] < \sum_j x[j].$$
- In the case where elements of x are sets since $\bigcup$ is associative, commutative, and monotonic (though not strictly monotonic) with respect to $\subset$:

  $$x\ R\ x' \equiv \bigcup_j x[j] \subseteq \bigcup_j x'[j].$$

- In the case where elements of x are sets since $\bigcap$ is associative, commutative, and monotonic (though not strictly monotonic) with respect to $\subset$:

$$x \; R \; x' \equiv \bigcap_j x[j] \subseteq \bigcap_j x'[j].$$

Next we give the predicate versions of the results on sets. We use the predicate versions in proofs.

- In the case where elements of x are Booleans since conjunction is associative, commutative, and monotonic with respect to implication:

$$x \; R \; x' \equiv \bigwedge_j x[j] \Rightarrow \bigwedge_j x'[j].$$

- In the case where elements of x are Booleans since disjunction is associative, commutative, and monotonic with respect to implication:

$$x \; R \; x' \equiv \bigvee_j x[j] \Rightarrow \bigvee_j x'[j].$$

3.3. Correctness of concurrent systems through local–global relations

The previous subsection considers local–global relations over arbitrary arrays. In this subsection, we apply these relations to the state space of concurrent systems. In this context, local–global relations can be used to reason about safety and progress properties of concurrent systems.

3.3.1. Sufficient conditions for proving safety

Let $\mathscr{A}$ be a concurrent system.

Definition 11 Let R be a binary relation on equal-sized arrays, and let a be an action of $\mathscr{A}$. Let $a \Longrightarrow R$ be a relation between actions and values of R. Its definition is as follows:

$$\alpha \Longrightarrow R \quad \equiv \quad \Big(\forall s, s' : e(s_a, a) \wedge \big(s'_a = t(s_a, a)\big) \Longrightarrow s_a \; R \; s'_a\Big).$$

The condition $\alpha \Longrightarrow R$ holds when the pre-states and post-states of agents participating in action a satisfy relation R. This condition deals with a local state transition rather than the global state transition—it does not deal with the agents that do not participate in action a.

Observation. If R is local–global relation, then if the pre-states and post-states of agents participating in action a satisfy R it follows that the pre-state and post-state of the global system also satisfy R.

$$a \Longrightarrow R \quad \equiv \quad \Big(\forall s, s' : e(s_a, a) \wedge \big(s'_a = t(s_a, a)\big) \Rightarrow s \; R \; s'\Big).$$

We extend the definition of $a \Longrightarrow R$ from a single action a to any set A of actions as follows:

$$A \Longrightarrow R \quad \equiv \quad \Big(\forall a \in A : a \Longrightarrow R\Big)$$

Theorem 3.7 *Given an automaton $\mathscr{A}$, and a local–global relation R, if $A \Longrightarrow R$, then for all executions of $\mathscr{A}$*

$$\forall y, z \in \mathbb{N} : y < z \Rightarrow s^{(y)} \; R \; s^{(z)}$$

Proof. Follows from property L1 of Definition 8 (transitivity of R), and from Definition 11. □

Corollary 3.8 (Conservation Law) *If $\circ$ is an associative and commutative operator, and*

$$\forall a, s : (\text{fold } \circ\ s_a) = (\text{fold } \circ\ t(s_a, a))$$

then

$$\forall t \in \mathbb{N} : (\text{fold } \circ\ s^{(t)}) = (\text{fold } \circ\ s^{(0)})$$

Proof. Follows from Theorems 3.3 and 3.7. □

Corollary 3.9 *If $\circ$ is an associative, commutative, monotonic operator, and*

$$\forall a, s : (\text{fold } \circ\ s_a) \geq (\text{fold } \circ\ t(s_a, a))$$

then

$$\forall y, z \in \mathbb{N} : y < z \Rightarrow \left(\text{fold } \circ\ s^{(y)}\right) \geq \left(\text{fold } \circ\ s^{(z)}\right)$$

Proof. Follows from Theorems 3.5 and 3.7. □

Corollary 3.8 is called a conservation law because it gives conditions for conserving fold with respect to any initial state. For example, if any action by any set of agents conserves the sum of agent values participating in the action, then the sum of all agent values is constant throughout an execution. Corollary 3.9 provides conditions for conserving the $\leq$ relation on the application of fold. For example, if any action by any set of agents does not increase the error of the agents participating in the action, then the error of the system does not increase with the execution of any sequence of actions.

3.3.2. Sufficient conditions for proving progress

Let d be a function from arrays of agent states to a partially ordered set P with $<$ ordering, G be an invariant of the system, and Q be a predicate on (global) states of the system. We are interested in sufficient conditions for proving that eventually Q holds. The following theorem lays the ground work for this proof, by showing that there are only two possible outcomes for a given fair execution: Q holds, or d strictly decreases.

Theorem 3.10 *If the following hold*

D1. $\forall a, s : G(s) \wedge E(s, a) \Longrightarrow d(s) \geq d(T(s, a))$
D2. $\exists F \in \mathcal{F} : \forall a \in F : G(s) \wedge \neg Q(s) \Longrightarrow E(s, a) \wedge d(s) > d(T(s, a))$

then for all complete executions:

E1. *either the execution is infinite, and for all $p \in P$, if $d(s) = p$ at any point in an execution, then there is an infinite suffix of the execution where $d(s) < p$ for all states in the suffix:*

$$\forall p : \Box\Big(d = p \Rightarrow \Diamond\Box\ d < p\Big)$$

or

E2. *every execution has a suffix where Q holds at every point in the suffix.*

$$\Diamond\Box\ Q$$

Proof. Proof is similar to Theorem 2.3. □

Condition D1 implies that d is monotone nonincreasing as an execution proceeds. Therefore, for any w, if $d(s) = w$ at any point in an execution then $d(s) \leq w$ forever thereafter in that execution. Condition D2 implies that in an infinite execution, an action that reduces d is executed infinitely often. From the two conditions, if $d(s) = w$ at any point in the execution then $d(s) < w$ at a later point in the execution. Theorem 3.10 is extended to deal with local–global relations:

Theorem 3.11 *If the following hold*

H1. $\forall a, s : G(s) \wedge e(s_a, a) \Rightarrow d(s_a) \geq d(t(s_a, a))$
H2. $\exists F \in \mathcal{F} : \forall a \in F : G(s) \wedge \neg Q(s) \Rightarrow e(s_a, a) \wedge d(s_a) > d(t(s_a, a))$
H3. *$d(s) \geq d(s')$ and $d(s) > d(s')$ are local–global relations*

then for all complete executions either E1 or E2 holds.

Proof. Follows from the following equalities,

$$\forall a, s : \Big(e(s_a, a) = E(s, a)\Big) \wedge \Big((d(s_a) \geq d(t(s_a, a)) \Rightarrow (d(s) \geq d(T(s, a))\Big)$$
$$\wedge \Big((d(s_a) > d(t(s_a, a)) \Rightarrow (d(s) > d(T(s, a))\Big),$$

which follow from the definition of e, and since $d(s) \geq d(s')$ and $d(s) > d(s')$ are local–global relations □

The following corollaries give sufficient conditions for proving progress. We denote by *Range*(d) the range of d.

Corollary 3.12 *If Range*(d) *is a well-founded set and Conditions H1–3 hold, then eventually* Q.

Proof. Since d is well-founded, $d(s)$ does not decrease infinitely often. □

Corollary 3.13 *If* $d(s) =$ fold $\circ$ s *where* $\circ$ *is associative, commutative, and strictly monotonic, then Conditions H1 and H3 hold.*

Proof. Proof follows from Theorem 3.5. □

3.4. Discussion on verification methodology

This section has introduced local–global relations and applied them to concurrent systems. When applied to such systems, local–global relations are relations on states that can be used to express safety and progress properties (Theorems 3.7 and 3.11, respectively). Local–global relations are at the root of a hierarchy of system properties. The abstract mathematical operator $\circ$, and its composition function fold, are used to specialize local–global relations. Property correctness is maintained because $\circ$ fits the local–global definition (Corollaries 3.8 and 3.9). As an example, consider a collection of analog sensors that use continuous dynamics to converge to the average of their initial values. A safety property of the system could be that, for the execution of any action, the sum of sensor values does not change. In this case, the addition operator is an instance of $\circ$, and equality is the relation between the summation on states. From Theorem 3.3, we know that such a relation is local–global, and therefore, the safety property holds.

Our methodology allows for systems with both discrete and continuous state spaces. This section has presented our methodology for discrete time systems, however it is possible to extend local–global relations to continuous time systems as well. In the next section, we present two examples of discrete time systems. Section 5 presents a continuous state space system operating over continuous time.

4. Application of local–global relations in verification

This section considers two examples motivated by distributed consensus protocols. In these examples, the system consists of N agents, each storing a real number. The first example fits the local–global layer in that we express properties of the system as local–global relations. This example focuses on convergence and describes a continuous state space system operating over discrete time. The second example fits the operator layer in that we express properties of the system using fold and $\circ$; this example focuses on termination. By representing their action sets as predicates, these two examples describe classes of protocols. At the end of each subsection we present specific instances from the literature [OSFM07, BHOT05, JLM03].

4.1. Distributed average without failures

The system $\mathscr{A}$ consists of N agents, each storing a real number. The goal of the system is to converge to the average of its initial values. This means that the global state should eventually get arbitrarily close to the state $\hat{s}$ where

$$\forall j \in [N] : \hat{s}[j] = \frac{1}{N} \sum_{k \in [N]} s_0[k]$$

where s_0 is an arbitrary initial state of the system.

Next, we prove that the system converges to $\hat{s}$ for a very general class of protocols. This class includes all group-based algorithms with the following property: the execution of an action maintains the average of the group, and decreases the group's mean square error (i.e., the variance of the group), by at least a fixed percentage. In this class of algorithms each group takes actions to solve its own local problem independent of agents outside the group, but local–global ensures that though each group follows a myopic strategy, the global problem is solved.

We model the automaton $\mathscr{A}$ as follows

States. The state of each agent i consists of the real-valued variable $s[i]$. Initially, $s[i]$ stores an arbitrary real value.

Actions. The set A of all feasible actions of the system is defined as

$$A = \{\, a \mid \forall s :\ average(s_a) = average(t(s_a, a)) \ \wedge\ MSE(s_a) \geq MSE(t(s_a, a)) \,\}$$

where

$$average(s_a) = \frac{1}{N_a} \sum_{j \in agents(a)} s[j]$$

and

$$MSE(s_a) = \frac{1}{N_a} \sum_{j \in agents(a)} (s[j] - average(s_a))^2$$

and N_a is size of the agent set: $N_a = \mid agents(a) \mid$.

The set A includes all group actions which keep the average of the group constant and do not increase the mean square error of the group. The enabled condition of action $a \in A$ for the state s is that the mean square error of $s[agents(a)]$ is positive; that is $e(s_a, a) = (MSE(s_a) > 0)$. For example, for $j, k \in [N]$ and $0 \leq r \leq 1$, the $interact(j, k, r)$ action (defined in the Example 3.1 and shown in Fig. 4b) belongs to the set A. This is because it maintains a constant average,

$$\begin{aligned} average(t(s_{interact(j,k,r)}, interact(j, k, r))) &= \frac{s[i] + r(s[j] - s[i]) + s[j] - r(s[j] - s[i])}{2} \\ &= average(s_{interact(j,k,r)}) \end{aligned}$$

and does not increase the mean square error,

$$\begin{aligned} &MSE(t(s_{interact(j,k,r)}, interact(j, k, r))) \\ &= \frac{(s[i] + r(s[j] - s[i]) - average(s))^2 + (s[j] - r(s[j] - s[i]) - average(s))^2}{2} \\ &= MSE(s_{interact(j,k,r)}) - (s[j] - s[i])^2 \cdot r \cdot (1 - r) \\ &\leq MSE(s_{interact(j,k,r)}) \end{aligned}$$

since, by assumption on r, $r \cdot (1 - r) \geq 0$.

Fairness. We assume that the system cannot be permanently partitioned into non communicating subsets. This ensures that for any nonempty sets J and $\bar{J}$, eventually an action is executed in which agents from J and $\bar{J}$ participate. After the execution of this action, the MSE of the group decreases by a factor lower bounded by some constant $\Delta > 0$. Formally, for every non empty $J \subset [N]$, we define

$$F_J = \{interact(j, k, r) \mid\ j \in J,\ k \in \bar{J},\ r \in [L, 1 - L],\ L \in (0, 0.5]\}$$

where L as the unique solution of the equation $x \cdot (1 - x) = \Delta$ in the interval $(0, 0.5]$. The fairness condition $\mathcal{F}$ is

$$\mathcal{F} = \{\, F_J \mid J \subset [N] \ \wedge\ J \neq \emptyset\}.$$

Informally, under this fairness criteria the system cannot be permanently partitioned into non-communicating subsets. Furthermore, when agents from complementary sets interacts the MSE of the group decreases by a factor lower bounded by $L \cdot (1 - L)$.

The state $\hat{s}$ stores the average of the initial values of the agents, and $MSE(\hat{s}) = 0$. To show correctness, we are obligated to prove that the automaton $\mathscr{A}$ converges to $\hat{s}$. The first step is to identify a local–global relation between states of the system. Then we use this relation to prove safety and progress.

Definition 12 For all $x, x' \in \mathbb{R}^N$

$$x\ R\ x' \ \equiv\ \Big(average(x) = average(x') \bigwedge\ MSE(x) > MSE(x')\Big)$$

where $average(x)$ is the average of the value in x, and $MSE(x)$ is the mean square error of x.

Lemma 4.1 *R is a local–global relation.*

Proof. By definition, LG1 of Definition 8 holds for R; it is left to show that LG2 (of the same definition) holds. Consider arbitrary $x, x' \in \mathbb{R}^N$ and J not empty subset of $[N]$. Since + is associative and commutative, LG2 holds for *average*

$$(average(x[J]) = average(x'[J])) \Rightarrow (average(x) = average(x'))$$

Next, we prove Condition LG2 for *MSE*. We want to show that

$$(MSE(x[J]) > MSE(x'[J])) \Rightarrow (MSE(x) > MSE(x'))$$

In order to do so, we show the following sequence of equalities

$$\begin{aligned}
&| J | \cdot (MSE(x[J]) - MSE(x'[J])) \\
&= \{ \text{ by definition } \} \\
&\sum_{j \in J} \Big(x[j]^2 - x'[j]^2 - 2x[j] \cdot average(x[J]) + 2x'[j] \cdot average(x'[J]) + average(x[J])^2 - average(x'[J])^2 \Big) \\
&= \{ \text{ since } average(x[J]) = average(x'[J]) \} \\
&\sum_{j \in J} \big(x[j]^2 - x'[j]^2 \big) \\
&= \{ \text{ since } average(x) = average(x') \} \} \\
&\sum_{j \in [N]} \big(x[j]^2 - x'[j]^2 - 2x[j] \cdot average(x) + 2x'[j] \cdot average(x') + average(x)^2 - average(x')^2 \big) \\
&= \{ \text{ by algebraic manipulation } \} \\
&N \cdot \big(MSE(x) - MSE(x') \big)
\end{aligned}$$

□

Given that R is local–global, we can apply Theorem 3.7 to show that the average of the system is maintained and mean square error is reduced.

Lemma 4.2

$$\big(\forall t : average(s^{(t)}) = average(s^{(0)}) \big) \bigwedge \big(\forall y, z : y < z \Rightarrow MSE(s^{(y)}) > MSE(s^{(z)}) \big)$$

Proof. By definition, the set of actions

$$A = \{ a \mid a \Rightarrow R \}$$

The lemma directly follows from Lemma 4.1 and Theorem 3.7. □

This result states that if the mean square error is positive then it will decrease eventually. It remains to be shown that the mean square error will converge to 0. To do so, we provide a positive fraction, α ($0 \leq \alpha < 1$), such that the mean square error decreases by at least α eventually.

Lemma 4.3 *There exists α ($0 \leq \alpha < 1$) such that*

$$\forall v : \Box \big(MSE = v \Rightarrow \Diamond \Box \; MSE \leq v \cdot \alpha \big)$$

with

$$\alpha = 1 - \frac{2L \cdot (1 - L)}{N \cdot (N - 1)^2}$$

Proof. At least one element of the array is greater than or equal to the average of the array

$$\exists j : (s[j] - average)^2 \geq MSE \tag{2}$$

with $MSE = v$. Without loss of generality, we restrict attention to the case where $s[j] - average < 0$ (proofs of the other case are symmetric).

Under this assumption, Eq. (2) becomes

$$-s[j] + average \geq RMSE \tag{3}$$

where $RMSE$ denotes the root mean square error $RMSE = \sqrt{MSE}$

Let x be the array s sorted in increasing order. Hence,

$$x[0] \leq s[j] \tag{4}$$

Since at least one element of x is greater than or equal to the average

$$x[N-1] \geq average \tag{5}$$

From Eqs. (3), (4) and (5)

$$x[N-1] - x[0] \geq RMSE \tag{6}$$

For $0 < k < N$, define $\Delta[k]$ as the difference between $x[k]$ and $x[k-1]$

$$\forall 0 < k < N : \Delta[k] = x[k] - x[k-1]$$

For all k, $\Delta[k] \geq 0$, and

$$x[N-1] - x[0] = \sum_{k=1}^{N-1} \Delta[k] \tag{7}$$

From Eqs. (6) and (7)

$$\sum_{k=1}^{N-1} \Delta[k] \geq RMSE$$

Hence, because for all k, $\Delta[k] \geq 0$,

$$\exists i : \Delta[i] \geq \frac{RMSE}{N-1} \tag{8}$$

Let F be the partition of the set of agents into two sets: set J of agents whose value is greater than or equal to $x[i]$ and its complement $\bar{J}$

$$J = \{j \mid s[j] \geq x[i]\}$$

Then from Eq. (8)

$$\forall u \in J, v \in \bar{J} : s[u] - s[v] \geq \frac{RMSE}{N-1} \tag{9}$$

From the fairness requirement, an *interact* action between some agent u in J and agent v in $\bar{J}$ occurs infinitely often. For completeness, we present the algebra for concluding that the mean square error after the interaction of u and v is at most α times the mean square error before the interaction. When the action $interact(u, v, r)$ is executed (with $r \in [L, 1-L]$), we have

$$s'[u] = s[u] - r \cdot (s[u] - s[v])$$
$$s'[v] = s[v] + r \cdot (s[u] - s[v])$$

Hence,

$$s'[u]^2 + s'[v]^2 = s[u]^2 + s[v]^2 - 2r \cdot (1-r) \cdot (s[u] - s[v])^2$$

Applying Eq. (9)

$$s'[u]^2 + s'[v]^2 \leq s[u]^2 + s[v]^2 - 2r \cdot (1-r) \cdot \frac{MSE}{(N-1)^2}$$

Therefore, the MSE of the new state s', which we denote by MSE', is such that

$$\begin{aligned} MSE' &\leq MSE - 2r \cdot (1-r) \cdot \frac{MSE}{N \cdot (N-1)^2} \\ &= MSE \cdot \left(1 - \frac{2r \cdot (1-r)}{N \cdot (N-1)^2}\right) \end{aligned}$$

Since $L \leq r \leq 1 - L$, and $r \cdot (1 - r) \geq L \cdot (1 - L)$

$$\begin{aligned} MSE' &\leq MSE \cdot \left(1 - \frac{2L \cdot (1 - L)}{N \cdot (N - 1)^2}\right) \\ &\leq v \cdot \alpha \\ &= MSE \cdot \alpha \end{aligned}$$

where

$$\alpha = \left(1 - \frac{2L \cdot (1 - L)}{N \cdot (N - 1)^2}\right)$$

with $\alpha \in [0, 1)$. □

We now have the tools to show system convergence.

Theorem 4.4 *$\mathscr{A}$ converges to $\hat{s}$.*

Proof. Follows from Lemma 4.3 and Corollary 2.5 on convergence since the sequence $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^n$ is decreasing and converging to 0. □

Moving to the average protocol. In this section we have proved that group-based algorithms with certain properties converge to the average of the initial system values. Any action of the system maintains the average of the group, and does not increase the group error. As an instance of this class, we consider the protocol in [JLM03]. In this group-based protocol, the state space, and fairness criteria, are inherited from the automaton $\mathscr{A}$, while the set of actions in the system is define as follows,

$$A' = \{\ a \mid \forall j \in \mathit{agents}(a) : s[j] := \mathit{average}(s_a)\ \}.$$

All agents participating in the action, set their value to the average of the group. It is straightforward to show that this protocol is an instance of the class. The actions in A' are a subset of the parent layer, as $A' \subset A$.

4.2. Distributed consensus without failures

The system consists of N agents where the state of each agent is a real number. The goal of the system is to eventually reach and remain in a state $\hat{s}$ in which

$$\forall j \in [N] : \hat{s}[j] = \text{fold}\ \circ\ s_0,$$

where s_0 is an arbitrary initial state of the system. This is a generic task; for example, if $\circ$ is instantiated with the min operator, then the system is required to eventually reach and remain in a state where all agent values are the minimum of the initial state. We provide a protocol that applies the $\circ$ operator to pairs of agents. In Theorem 4.10, we prove that this protocol solves the task in that the system terminates in $\hat{s}$. Our results apply provided that the operator is associative, commutative, and idempotent.

Components of the concurrent system $\mathscr{A}$ are as follows

States. The state of agent j consists of the real-value variable $s[j]$. Initially, $s[j] = s_0[j]$.

Actions. The set A is

$$A = \{(j, k) \mid s[j] := s[j] \circ s[k],\ j, k \in [N]\}$$

The action $s[j] := s[j] \circ s[k]$ is represented by the ordered pair (j, k). The pair is ordered because in action (j, k) agent j reads the value $s[k]$ of agent k and sets the state of agent j to $s[j] \circ s[k]$ while leaving the state of agent k unchanged.

Fairness. The fairness criterion is that for every nonempty subsets $J, \bar{J}$ of agents, infinitely often some agent j in J updates its own state with the state of some agent k in $\bar{J}$.

$$F_J = \{(j, k) \mid j \in J \wedge k \in \bar{J}\}$$

The fairness condition becomes

$$\mathcal{F} = \{F_J \mid J \subseteq [N]\}$$

Given this system, we would like to show that $\mathscr{A}$ terminates in $\hat{s}$. Following the strategy from Sect. 4.1, we first identify a local–global relation between states and use it to show safety and progress. In this example, safety and progress imply termination and the local–global relation is a relation on fold.

Definition 13 For all $y, y' \in \mathbb{R}^N$, $y\ R\ y' \equiv \Big((\text{fold}\ \circ\ y) = (\text{fold}\ \circ\ y')\Big)$

Lemma 4.5 *R is a local–global relation.*

Proof. Follows from Theorem 3.3 and assumptions on $\circ$. □

Lemma 4.6 $\forall t : (\text{fold}\ \circ\ s^{(t)}) = (\text{fold}\ \circ\ s^{(0)})$

Proof. Follows from Theorem 3.7. □

This result states that the executions of the actions in A maintain $(\text{fold}\ \circ\ s)$ constant and equal to $(\text{fold}\ \circ\ s^{(0)})$. For example, if $\circ$ is the min operator, this lemma implies that throughout the execution of $\mathscr{A}$ the min of the system does not change.

We prove progress of the system by applying Theorem 3.11. Recall the theorem requires an invariant of the system G, a distance function d on the state space of $\mathscr{A}$, and a predicate Q on the global state of the system. For this reason, we introduce an auxiliary variable x that is an array of length N. The variable x is not included in the automaton because it plays no role in the algorithm. Each entry of the array stores a set of indices ($\subseteq [N]$), where the entry $x[j]$ keeps track of the indices of the agents used by agent j to update its own state. Initially each agent know its own value, hence for all j the variable $x[j]$ is initialized as $x[j] = \{j\}$. When an action (j, k) is executed, the set $x[j]$ is updated as follows

$$x[j] := x[j] \bigcup x[k] \tag{10}$$

After executing the action (j, k), the set $x[j]$ contains the index k, along with all indices used by k to compute its own state. Since indices are never removed from x, it is always the case that

Lemma 4.7 $\Box(\forall j : j \in x[j])$.

Proof. Follows by construction. □

Using the auxiliary variable x, we define the following two predicates

Definition 14 Predicates G and Q on the state space of the system, are defined as follows

$$G(s) \equiv \forall j : s[j] = \text{fold}\ \circ\ s^{(0)}[x[j]]$$
$$Q(s) \equiv \forall j : (x[j] = [N])$$

The predicate G holds in state s if the state of each agent can be obtained as the application of fold to the restriction of $s^{(0)}$ on the indices stored in x. The predicate Q holds in s if in state s the auxiliary variable x contains all agent indices.

The variable x plays an important role in the definition of these predicates and in the proof of correctness of the system. In Lemma 4.8, we prove that G is an invariant of the system; while, in Lemma 4.9, we define a distance function d using x and prove using Theorem 3.11 that eventually Q. Combining these two lemmas and Lemma 4.6, the proof of the main theorem follows. We start by proving the following.

Lemma 4.8 *The predicate G is an invariant of the system.*

Proof. From the initialization of x, G is true initially.
Consider an arbitrary $j \in [N]$. By definition, we know that $s'[j] = s[j] \circ s[k]$ for all action (j, k) where s' denotes the post-state of action (j, k). Since the invariant holds for $s[j]$, $s[k]$, we have that

$$\begin{aligned} s'[j] &= (\text{fold}\ \circ\ s^{(0)}[x[j]]) \circ (\text{fold}\ \circ\ s^{(0)}[x[k]]) \\ &= \text{fold}\ \circ\ \Big(s^{(0)}[x[j]] \bigcup s^{(0)}[x[k]]\Big) \end{aligned}$$

Since $\circ$ is idempotent

$$\forall j : s'[j] = \text{fold}_{\circ}\, s^{(0)}[x'[j]].$$

where x', that denotes the vector x in the state s', has been computed using Eq. (10). □

Eventually all agents update their own states using the values of all agents.

Lemma 4.9 *The predicate Q eventually-always holds.*

Proof. For each $k \in [N]$, and state s, we define J_k as the set of agents j where $x[j]$ does not include k

$$J_k(s) = \{j \mid k \notin x[j]\}$$

Then, by construction, it follows that for all actions of the system, where execution of the action takes the system from state s to s', $J_k(s') \subseteq J_k(s)$.

For agent k, we define the predicate Q_k on the state space of the system, as

$$Q_k(s) \equiv \forall j : (k \in x[j]) \tag{11}$$

which is true if k is a member of $x[j]$ for all j.

From the definition of Q_k, $\neg Q_k \equiv (J_k \neq \emptyset)$. Consider the case where $\neg Q_k$ holds. For this case, partition the set of agents into J_k, and its complement. From Lemma 4.7, it follows that $\bar{J}_k \neq \emptyset$; and by definition of $\neg Q_k$, $J_k \neq \emptyset$. Let F_k be the set of actions

$$F_k = \{(i, j) \mid i \in J_k \wedge j \in \bar{J}_k\}$$

Execution of any action in F_k strictly decreases the cardinality of J_k.

From Theorem 3.11 with $d_k(s)$ defined to be the size of $J_k(s)$, $\Diamond\Box\ Q_k$. Since $\Diamond\Box$ is conjunctive

$$(\forall k : \Diamond\Box\ Q_k) = \Diamond\Box\ (\forall k : Q_k), \text{ and}$$
$$(\forall k : Q_k) \Rightarrow (\forall j : x[j] = [N]).$$

The progress property follows. □

We are now able to show system termination.

Theorem 4.10 *If $\circ$ is associative, commutative and idempotent, $\mathscr{A}$ terminates in $\hat{s}$.*

Proof. The theorem follows from the above safety and progress properties—Lemmas 4.6, 4.8 and 4.9, respectively. □

Consensus using non-linear operators. As concrete examples, we consider the protocols in [OSFM07, BHOT05], where the $\circ$ operator is instantiated with non-linear operators such as max and min. We present the protocol for min. This concrete instantiation has the same state space and fairness criteria as the generic protocol presented above. The set of actions is refined as follows,

$$A' = \{\ (j, k) \mid s[j] := \min(s[j], s[k]),\ \ j, k \in [N]\ \}.$$

In this case, agent j takes on the minimum value of itself and the value of agent k. This protocol is a valid instantiation of the generic protocol because the min operator satisfies the assumptions on $\circ$: associativity, commutativity, and idempotence. In this example the state space of the system is discrete in that the set of values each agent can contain is finite, dictated by the initial values in the system.

5. Faulty message passing systems with continuous state spaces

In this section, we discuss local–global relations, stability, convergence and termination for systems where agents interact by sending messages. In this class, agents interact by exchanging messages, which can be lost, delayed or arrive out-of-order. A key assumption of these systems is that messages are either lost or delivered in finite but unknown time.

We present results for proving convergence of these systems and present examples. One of the examples considers the message passing version of the Gauss algorithm, an iterative scheme used to solve systems of linear equations. We show that this algorithm is correct in that it recovers the solution of the system. We are interested in this specific problem because many pattern formation algorithms are instances of the general problem; for example, agents forming a straight line [CMP08].

5.1. Automata formulation

We give a formal description of a message passing system with bounded delay. A message passing system consists of N agents that use a communication medium to interact. We first describe the agents and then the communication layer.

Agent automata. Each agent j can be formalized as an automaton. Its state $s[j]$ contains (1) a clock variable now_j, which stores agent j local time and is initially set to 0, and (2) a vector $C[j, \cdot]$, which stores in $C[j, k]$ the last message received from k for each $k \neq j$, each entry of the vector is initially set to $\perp$. Each agent can store other problem specific state variables. Agents have input, output and internal actions. They must implement a *send* and *receive* action. When agent j executes a *send* action, it sends the pair (j, v), i.e. its sends its identifier and the current value of some variable v of interest (stored in $s[j]$). The agent executes the *send* action infinitely often; however, the number of messages sent within a finite time interval is finite. The input action *receive* delivers messages to the agent. It is controlled by the communication channel described below.

Communication channel. The communication layer is a broadcast channel $\mathcal{M}$ allowing for lost, delayed or out-of-order messages. We make the following assumptions that formalizes our fairness requirement on agent communication:

- messages remain on the channel for a finite, but unknown, amount of time, denoted by T; and
- for all pair of agents (i, j), i receives messages from j infinitely often.

Under these assumptions, messages can either get lost or received in bounded, but unknown, time. It is not possible that all messages between two agents are lost; rather, it is always the case that eventually some message from i is delivered to j.

The state of $\mathcal{M}$ consists of a two-dimensional array buf of size $N \times N$. Each entry (j, k) contains the set of time stepped messages in transit from j to k. Initially, each entry is empty. We assume that $buf[j, j]$ is always empty. The communication medium has a real time variable called *now*, initially set to 0. The actions include

- $send_j(m)$. This is a input action where agent j broadcasts message m. This action adds the time message (m, t) to all the out channels of j with $t = now + T$. The value t is called the deadline of m. Message m must be delivered by real-time t.
- $drop_{j,k}(m)$. It is an internal action and models the loss of message m in transit from j to k.
- $receive_j(m)$. This is an output action where agent j receives message m. The receive action is enabled only when the deadline of message m is not violated. This means that the current real-time is not larger than the message deadline. The message m is deleted from the channel.

A detailed description of this automaton is presented in [CMP08]. Readers interested in its implementation are referred to our PVS code [GMPJ09]. Further, the complete partially synchronous system is modeled as an automaton $\mathscr{B}$ that is obtained by composing N agent automata and the broadcast channel automaton.

5.2. Proofs of stability and convergence using local–global relations

This section discusses the concepts of stability, convergence, termination and local–global relations for partially synchronous systems. These properties can only be shown for un-timed systems, because the values of real-time related variables diverge along any admissible execution of the automaton. Given a timed system, the corresponding un-timed one is obtained by removing the real-time variables from the states of the system. Hence, the un-timed automata for a message passing system has the same variables of the timed one, except the time variables.

The definitions of stability, convergence and termination for un-timed message-passing systems are analogous to the ones given in Sect. 2. In [CMP08] (Theorem 2), the authors present sufficient conditions for proving convergence of $\mathscr{B}$ (the message passing system automaton) in terms of the convergence proof of the corresponding shared state system $\mathscr{A}$. A synchronous system is a concurrent system, where the execution of any action a can change only the state of one agent in $agents(a)$ (the state of all remaining agents is unchanged). These results rely on establishing the existence of a (Lyapunov) function that is non-increasing along all executions of the system (see [Tsi87]). This Lyapunov function is usually described in terms of a collection of level sets of the

system state space. The sufficient conditions presented in [CMP08] have two assumptions. First, level sets of the Lyapunov function are *rectangular*. The second assumption pertains to the communication medium: for each communication channel, the receiver eventually receives at least one message from the sender. In [CMP08], the authors do not allow for out-of-order messages.

The definition of local–global relation can be extended to message passing system. We represent the state of the system as an array of length $2 \cdot N^2$; the first N^2 entries contain the values of the matrix C, while the remaining N^2 entries correspond to the buf variable and contain the set of messages in transit between any pair of agents. Hence, the definition of local–global relations for message passing systems is analogous to Definition 8.

5.3. Distributed consensus with messages

In this subsection, we extend the example in Sect. 4.2 to message passing systems. Each agent j stores a real value in $s[j]$ and sends the pair $(j, s[j])$ infinitely often. When agent j receives a message m from $m.sender$ (which stores the identifier of the sender of m), it sets its value to

$$s[j] := s[j] \circ m.value$$

where $m.value$ is the content of the message. Theorem 4.10 proved that the system reaches consensus under perfect communication. The theorem also holds under partial synchronous communication, however. If we denote by $\mathscr{B}$ the partially synchronous (un-timed) automaton, we have the following;

Theorem 5.1 *If $\circ$ is associative, commutative and idempotent, $\mathscr{B}$ terminates in $\hat{s}$.*

Proof. The proof of Theorem 4.10 applies to message passing systems as well since fold is assumed to be idempotent. □

5.4. Solving systems of linear equations

In this section, we consider an iterative scheme for solving systems of linear equations. In partially synchronous systems, each agent is responsible for solving one equation of the system (i.e. finding the value of one variable in the system). Agents update their variables using the received messages; these messages contain values of the other variables which are potentially old and computed at different times.

We show convergence for this iterative scheme. This is an example where the state space is continuous and the system operates over continuous time. We are interested in this algorithm, because many iterative schemes for pattern formations of robots (for example [OSFM07, CMP08]) are instances of this algorithm. In the end of this section, we discuss a specific pattern formation algorithm where agents want to form a straight line. The results presented in this Section extend the work of [GM80] in the linear case, the work of [CMP08] allowing for out-of-order messages and the work of [CM69] relaxing some assumptions on the equations of the system.

The system we consider is as follows.

Linear systems of equations. The system consists of N agents that communicate via wireless messages. Each agent $j \in \{0, \ldots, N-1\}$, has a real-valued state variable denoted by $x[j]$. Let A be an invertible $N \times N$ matrix of reals and b be a vector of real numbers with N elements indexed by $\{0, \ldots, N-1\}$. Let $\hat{x}$ be the solution to the equations $A \cdot x = b$, that is, $\hat{x} = A^{-1}b$. The goal of the algorithm is for the states of the individual agents to converge to $\hat{x}$ by exchanging messages.

Restrictions on A. We assume that A is invertible, weakly diagonally dominant, $N \times N$ matrix with diagonal elements equal to unity; that is, $\forall k : \sum_{j \neq k} \mid A[k, j] \mid \leq 1$. We make the additional assumption that:

$$\exists i : \sum_{j \neq i} \mid A[i, j] \mid < 1. \tag{12}$$

The *incidence graph* (V, E) of matrix A is defined as follows: V is the set $[N]$, and a directed edge (j, k) is in E, if $A[j, k] \neq 0$. For any i and j in $[N]$, j is said to be a *neighbor* of i if (i, j) is an edge in E. We denote the set of neighbors of i by N_i. A vertex $i \in [N]$ is said to be a *root* if it satisfies Eq. (12). For the remainder of this section, we restrict our attention to the matrices where there is a directed path from all vertices to a root. Our

assumption on the matrix A relaxes the assumption in [CM69] where the authors consider strictly diagonally dominant matrices.

Given the incidence graph, we define a *rooted forest* as a collection of disjoint trees rooted at root vertices. Given a rooted forest $\mathbb{F}$ and a vertex $j \in \mathbb{F}$, we denote by *ancestors*$[j]$ the set of ancestors of j in the forest. This set includes j itself, and for any vertex k is in *ancestors*$[j]$, if k has a parent k', then k' also in *ancestors*$[j]$.

Agent states. Associated with each agent $j \in [N]$ are the following state variables:

- $x[j]$ is a real-valued variable and is called the *location*. Initially, $x[j]$ is arbitrary.
- $z[j]$ is also a real-valued variable and is called the *target*. Initially, $z[j]$ is equal to the initial value of $x[j]$.
- $C[j, \cdot]$ is an array of length N with elements of type $\mathbb{R} \cup \{\perp\}$. For all $k \neq j$, $C[j, k]$ contains the last message received by j from k; if no such messages exists then $C[j, k] = \perp$. $C[j, j] = x[j]$.

Agent actions. The state of an agent changes through the performance of three types of actions. In what follows, we describe the state transitions brought about by these action.

- A $send_j(m)$ action models the sending of message m by agent j. A message sent by j is a pair $(j, x[j])$. We denote by $m.sender$ the first component, j in this case, and by $m.value$ the second component $x[j]$, of message m. Agent j sends messages to the channel infinitely often; however, the number of messages sent within a finite time interval is finite.
- A $receive_j(m)$ action models the receipt of message m by agent j. When $receive_j(m)$ occurs, it sets $C[j, m.sender]$ to be $m.value$. If agent j has received a message from all of its neighbors, that is, for all $k \in N_j : C[j, k] \neq \perp$, then the target $z[j]$ is set as

$$z[j] := b[j] - \sum_{k \neq j} A[j, k] \cdot C[j, k].$$

  Notice that the right hand side is the solution of the j^{th} equation in the system of linear equations, with $C[j, k]$ in place of $x[k]$.
- A $move_j(dt)$ models the movement of agent j towards $z[j]$ over the finite time interval dt. This action updates the values of $x[j]$ and of the global clock *now* as follows

$$x[j] := f_j(z[j], dt)$$
$$now := now + dt$$

  where the function f_j represents the dynamics of agent j. We only allow dynamics where

$$x'[j] \in [x[j], z[j]]$$

  with x' being the value of $x[j]$ in the post-state of the action. In particular, we allow agents to be stationary in the interval dt. As described in the next paragraph, agents will eventually move because of fairness constraints. It should be noted that the action does have a pre-condition: it can be executed only if the value of *now* in the post-state does not violate messages deadlines. In other words, the deadlines of all messages in $\mathcal{M}$ are greater than $now + dt$.

Fairness. Given agent j, we denote by $x^{(t)}[j]$ and $z^{(t)}[j]$ the values of $x[j]$, $z[j]$ at time t. These variables are not well-defined at points in time where multiple discrete actions happen at the same real time. In this case, we assume their values to be the ones stored in the post-state of the last action executed at time t.

Denote by L the position of agent j at real time t. In the case when for all $t' \geq t$, $z^{(t')}[j] > L$, we make the following fairness assumption: agent j will either move to reach its destination, or it will move to its right by at least some constant amount Δ. Formally,

$$\forall j \in [N], t \in \mathbb{R}_{\geq 0}, (L = x^{(t)}[j]) \in \mathbb{R} :$$
$$\left(L < z^{(t)}[j]\right) \wedge \left(\forall t' \geq t : L < z^{(t')}[j]\right) \Rightarrow \left(\exists t' \geq t : x^{(t')}[j] = z^{(t')}[j] \vee x^{(t')}[j] \geq L + \Delta\right)$$

The condition where the agent destinations are to the left of the agents location is symmetric.

Our goal is to show that the corresponding untimed automaton $\mathscr{A}$ converges to $\hat{S}$ which is the set of all final state of the system. This set consists of all states where

$$\forall j : x[j] = \hat{x}[j] \ \wedge \ z[j] = \hat{x}[j] \wedge \ \forall k : C[k, j] = \hat{x}[j] \wedge \forall m \in \mathcal{M} : \ (m.sender = j \Rightarrow m.value = \hat{x}[j])$$

i.e., for each agent j, all occurrences of j stores the value $\hat{x}[j]$.

This formulation is very general. This is an iterative scheme, where agent j solves the jth equation of the system using values received from its neighborhood. In our model, agent j has two variables to represent the solution to its equation, *location* and *target*. Using two variables allows us to model system dynamics, which are common in real-world applications such as robotics. For example, if *location* represents the current position of agent j and *target* represents its destination position, then the *move* models the movement of agent j as a function of time. If only one variable is used to represent the solution, this action would be instantaneous.

We want to prove that the automaton $\mathscr{B}$ converges to $\hat{S}$. First we introduce definitions to capture the evolution of the errors of the system. These definitions are used to create a local–global relation.

Definition 15 We define the following error variables and functions in the system

- *localError* is an array of length N. For each agent, it stores the difference between agent desired location and its current location,

  $$\forall j : \mathit{localError}[j] = \mid \hat{x}[j] - x[j] \mid$$

- *destinationError* is an array of length N containing the error of the target location z,

  $$\forall j : \mathit{destinationError}[j] = \mid \hat{x}[j] - z[j] \mid$$

- *receivedError*$[j, \cdot]$ is an array of length N storing the errors of the values of $C[j, \cdot]$. For all k,

  $$C[j, k] = \perp \Longrightarrow \mathit{receivedError}[j, k] = 0$$
  $$C[j, k] \neq \perp \Longrightarrow \mathit{receivedError}[j, k] = \mid \hat{x}[k] - C[j, k] \mid$$

- the function $\mathit{msgError} : \mathcal{M} \to \mathbb{R}$ computes the error of the messages in $\mathcal{M}$. If m is a message in transit,

  $$\mathit{msgError}(m) = \mid \hat{x}[m.sender] - m.value \mid$$

- *error* is a array of length N. For each agent, it contains the maximum error of the agent in the system. For all j

  $$\begin{aligned} \mathit{error}[j] = \max \Big(& \mathit{localError}[j], \\ & \mathit{destinationError}[j], \\ & \max_{k \neq j} \mathit{receivedError}[k, j], \\ & \max_{m \in \mathcal{M} \wedge m.sender = j} \mathit{msgError}(m) \Big) \end{aligned}$$

- E is the maximum error over all agents

  $$E = \max_j \mathit{error}[j]$$

The following inequality relates agent destination and received errors.

Lemma 5.2 *For all* $j \in [N]$,

$$(\forall k \in N_j \ : \ C[j, k] \neq \perp) \Rightarrow \left(\mathit{destinationError}[j] \leq \sum_{k \neq j} \mid A[j, k] \mid \cdot \mathit{receivedError}[j, k] \right)$$

Proof. Assume that $\forall k \in N_j$, agent j has received at least one message from k. Under this assumption, agent j sets its destination variable $z[j]$ to

$$z[j] = b[j] - \sum_{k \neq j} A[j, k] \cdot C[j, k]$$

By definition,

$$\hat{x}[j] = b[j] - \sum_{k \neq j} A[j,k] \cdot \hat{x}[k]$$

The difference of the two above expression is given by

$$(\hat{x}[j] - z[j]) = -\sum_{k \neq j} A[j,k] \cdot (\hat{x}[k] - C[j,k])$$

Hence,

$$\mathit{destinationError}[j] \leq \sum_{k \neq j} \mid A[j,k] \mid \cdot \mathit{receivedError}[j,k]$$

□

We next define a local–global relation on the state of the system

Definition 16 For all $s, s' \in \mathbb{R}^{2 \cdot N^2}$, we define R as follows

$$R(s, s') \equiv \big(E(s) \geq E(s')\big)$$

where $E(s)$ is the maximum error of the system at state s.

We want to show that R is local–global.

Lemma 5.3 *R is a local–global relation.*

Proof. It follows from the definition of E. □

Using this local–global relation, we show that the error of the system does not increase.

Lemma 5.4 $\forall u, v \in \mathbb{R} : 0 \leq u \leq v : E(s^{(u)}) \geq E(s^{(v)})$

In this lemma $s^{(u)}$ denotes the state of the system at time u.

Proof. We first prove that the set of action A of the automation $\mathscr{B}$ implies the relation R, i.e. $A \Longrightarrow R$. This is equivalent to showing that the maximum error E does not increase when any action is executed.

The proof is straightforward when the system executes a *send* action. This is because no new values are added to the system. The proof of the case when a *receive* action is executed follows from Lemma 5.2. It holds when the system executes a *move* action. In this case, by assumptions on the dynamics, the error of the location in the post-state is upper bounded by the maximum of the *destinationError* and *localError* of the pre-state.

Using the generalization to continuous time system of Theorem 3.7, the proof follows. □

Next we find some factor α, $0 \leq \alpha < 1$, for which the error of the system, E, decreases (eventually) by α while remaining positive. We fix an arbitrary rooted forest $\mathbb{F}$ of the incidence graph of the matrix A, and give the following recursive definition:

Definition 17 For each agent j, we define $p[j]$ as follows

$$p[j] = \begin{cases} \sum_{k \neq j} \mid A[j,k] \mid & \text{if } j \text{ is a root in } \mathbb{F}, \\ \mid A[j, \mathit{parent}(j)] \mid \cdot\, p[\mathit{parent}(j)] + \sum_{k \notin \{j, \mathit{parent}(j)\}} \mid A[j,k] \mid & \text{otherwise.} \end{cases}$$

where $\mathit{parent}(j)$ denotes the parent of j in the forest $\mathbb{F}$.

The p values are constants in $[0, 1)$:

Lemma 5.5 $\forall j : 0 \leq p[j] < 1$

Proof. Proof follows by induction on the trees within the forest using the restriction on A in Eq. (12). □

Using the p values, we define the following family of predicates.

Definition 18 For any real number W, and $j \in [N] \cup \perp$,

$$Z_j \equiv \begin{cases} E \leq W & \text{if } j = \perp, \\ \Big(\forall k \in \mathit{ancestors}[j] : \mathit{error}[k] \leq W \cdot p[k]\Big) \wedge E \leq W & \text{otherwise.} \end{cases}$$

For all non-root vertices j, the predicate $Z_{parent(j)}$ is well defined. When i is a root of $\mathbb{F}$, we assume $Z_{parent(i)} \equiv Z_{\perp}$.

For each agent k the predicate Z_j gives an upper bound on its error in the system. This upper bound is $p[k] \cdot W$ if the agent is an ancestor of j and it is W otherwise. Note that $p[k] \cdot W < W$, as shown in Lemma 5.5.

We now show stability and progress results for this family of predicates. We first prove that if the predicate Z_j (with $j \in [N] \cup \perp$) holds at any point in an execution, then this predicate continues to hold forever thereafter.

Lemma 5.6 *For any real number W, for any $j \in [N] \cup \perp$, $\Box\,(Z_j \Rightarrow \Diamond\Box\, Z_j)$*

Proof. The proof is straightforward when executing a send or a move action. When executing a receive action, we consider two cases.

Case 1: We assume that $j \in [N]$, $k \in$ *ancestors*$[j]$ and k executes a receive action. The interesting case is when k updates its destination $z[k]$. We want to show that Z_j holds for k. From Lemma 5.2,

$$\textit{destinationError}[k] \leq \sum_{l \neq k} \mid A[k, l] \mid \cdot \textit{receivedError}[k, l]$$

From the above equation, it holds that, if k is a root of $\mathbb{F}$,

$$\textit{destinationError}[k] \leq W \cdot p[k].$$

This follows from the definition of $p[k]$ and since $\forall l : \textit{receivedError}[k, l] \leq W$ (from Z_j). When k is not a root, rewriting the above equation, we get

$$\begin{aligned}\textit{destinationError}[k] \leq{}& \mid A[k, \textit{parent}(k)] \mid \cdot \textit{receivedError}[k, \textit{parent}(k)] \\ &+ \sum_{l \neq \{k, \textit{parent}(k)\}} \mid A[k, l] \mid \cdot \textit{receivedError}[k, l]\end{aligned}$$

Using Z_j and the recursive definition of $p[k]$, we get

$$\begin{aligned}\textit{destinationError}[k] &\leq \mid A[k, \textit{parent}(k)] \mid \cdot\ W \cdot p[\textit{parent}(k)] + \sum_{l \neq \{k, \textit{parent}(k)\}} \mid A[k, l] \mid \cdot\ W \\ &\leq W \cdot p[k]\end{aligned}$$

Case 2: We assume that $j = [N] \cup \perp$ and agent k executes the receive action. We also assume that if $j \in [N]$, then $k \notin$ *ancestors*$[j]$. We want to show that Z_j holds for k when k updates its destination $z[k]$.

Using Lemma 5.2, the assumption on Z_j and weakly diagonally dominant assumption on A, we get

$$\begin{aligned}\textit{destinationError}[k] &\leq \sum_{l \neq k} \mid A[k, l] \mid \cdot \textit{receivedError}[k, l] \\ &\leq \sum_{l \neq k} \mid A[k, l] \mid \cdot W \\ &\leq W\end{aligned}$$

□

We next show the progress result. If the predicate $Z_{parent(j)}$ holds at any point in an execution, then the predicate Z_j eventually holds.

Lemma 5.7 *For any real number W, for any $j \in [N]$, $\Box\,(Z_{parent(j)} \Longrightarrow \Diamond Z_j)$*

Proof. The proof is by induction on each tree of $\mathbb{F}$.

Base Case. Let j be a root in $\mathbb{F}$. Predicate $Z_{parent(j)} \equiv Z_{\perp}$ holds. From Lemma 5.6, we know that once this condition holds, then it continues to hold forever thereafter. In order to prove that Z_j holds at some point later in the execution, we only need to show that $\textit{error}[j] \leq W \cdot p[j]$.

From the fairness criterion on agent communication, j receives messages infinitely often from any k with $A[j, k] \neq 0$. Therefore,

$$\forall k \ :\ \textit{receivedError}[j, k] \leq W$$

Hence, using Lemma 5.2, and the definition of $p[i]$, eventually

$$\mathit{destinationError}[j] \leq W \cdot p[j]$$

From the fairness condition on agent movement, eventually

$$\mathit{localError}[j] \leq W \cdot p[j]$$

By assumption on $\mathcal{M}$, all earlier messages will no longer be in transit after some bounded time T. Hence,

$$\forall m \in \mathcal{M} \wedge (m.\mathit{sender} = j) : \mathit{msgError}(m) \leq W \cdot p[j]$$

Hence, for any k where $A[k, j] \neq 0$, eventually

$$\mathit{receivedError}[k, j] \leq W \cdot p[j]$$

From the all the above equations, eventually

$$\mathit{error}[j] \leq W \cdot p[j]$$

Induction step. By assumption, j is not a root. The proof is very similar to the base case, and we only show the reduction in error of the $\mathit{destinationError}[j]$.
Using Lemma 5.2,

$$\begin{aligned}\mathit{destinationError}[j] \leq {} & \mid A[j, \mathit{parent}(j)] \mid \cdot \mathit{receivedError}[j, \mathit{parent}(j)] \\ & + \sum_{k \neq \{j, \mathit{parent}(j)\}} \mid A[j, k] \mid \cdot \mathit{receivedError}[j, k]\end{aligned}$$

Using $Z_{\mathit{parent}(j)}$, and the recursive definition of $p[j]$, eventually

$$\begin{aligned}\mathit{destinationError}[j] &\leq \mid A[j, \mathit{parent}(j)] \mid \cdot W \cdot p[\mathit{parent}(j)] + \sum_{k \neq \{j, \mathit{parent}(j)\}} \mid A[j, k] \mid \cdot W \\ &\leq W \cdot p[j]\end{aligned}$$

□

Using the local–global relation, we have shown that the maximum error of the system does not increase for any infinite fair trajectory of the system. We still have to show that it will converge to 0. Next we provide some factor α, $0 \leq \alpha < 1$, for which the error of the system, E, decreases (eventually) by α while remaining positive.

Lemma 5.8 *There exists an α where $0 \leq \alpha < 1$, for all $W > 0$,*

$$\Box(E > 0 \ \wedge \ E \leq W \Rightarrow \Diamond\Box\, E \leq W \cdot \alpha)$$

Proof. Let α be defined as follows,

$$\alpha = \max_j p[j]$$

Then from Lemma 5.5, $\alpha < 1$.

Consider the family $\{Z_i\}_{i \in [N] \cup \perp}$. By assumption, $E \leq W$, meaning that $Z_\perp$ holds. It holds forever thereafter, by Lemma 5.6. $Z_\perp$. Using Lemma 5.7, Z_i eventually holds with i being any root of $\mathbb{F}$. From the fairness criterion on agent communication and agent movement, iterating Lemma 5.6 and Lemma 5.7 for all levels of $\mathbb{F}$, eventually results in

$$\forall j : \mathit{error}[j] \leq W \cdot p[j]$$

Hence, eventually

$$\begin{aligned}E &= \max_j \mathit{error}[j] \\ &\leq W \cdot \max_j p[j] \\ &= W \cdot \alpha\end{aligned}$$

Using Lemma 5.6, $E \leq W \cdot \alpha$ holds forever thereafter. □

Theorem 5.9 *$\mathscr{B}$ converges to $\hat{S}$.*

Proof. Follows from Lemma 5.8 and Corollary 2.5 on convergence. □

Mobile agent pattern formations. As an instance of the iterative method provided above, we present a pattern formation algorithm. Consider a system of N agents communicating using a wireless medium. The goal is to form the following pattern. Assuming 0 and $N-1$ stationary, we want to design algorithms for other agents so that they eventually converge to a straight line joining the locations of agents 0 and $N-1$, where agents are equispaced. In [CMP08], the authors discuss convergence results for the following algorithm. Each agent $j \neq \{0, N-1\}$ upon receive a message m_l from $j-1$ and m_r from $j+1$, sets its location to the average of the two received locations,

$$s[j] := \frac{m_l.value + m_r.value}{2}$$

In [CMP08], the authors show convergence of the algorithm assuming instantaneous movement (from their current position to their new position) and not allowing for out-of-order messages.

The algorithm can be translated into a system of linear equations $A \cdot x = b$ where

$$\begin{aligned}
\forall 0 \leq j \leq N-1 &: A[j,j] = 1 \\
\forall j > 0 &: A[0,j] = 0 \\
\forall j < N-1 &: A[N-1,j] = 0 \\
\forall 0 < j < N-1 &: (A[j,j-1] = A[j,j+1] = -0.5) \wedge (\forall i \notin [j-1, j+1] : A[j,i] = 0) \\
\forall j \notin \{0, N-1\} &: b[j] = 0 \\
b[0], b[N-1] &: \text{ arbitrary values}
\end{aligned}$$

This matrix satisfies the condition given by Eq. (12). In the rows 0, $N-1$, the sum of the non-diagonal entries is 0, while the diagonal entries have value 1.

The solution to this system of equations has all the $x[j]$ values equispaced between $b[0]$ and $b[N-1]$. The algorithm converges to a straight line, from Theorem 5.9, also in the case of non-instantaneous movement. We can design algorithms for other formations in a similar way.

6. Verifying distributed systems using PVS

The theorems presented in this paper have also been verified mechanically using PVS. This allowed us to not only gain confidence in our methodology, but to build a reusable library of theorems for proving correctness of concurrent systems. Verifying theorems mechanically takes more work than proofs checked by hand, as nothing can be assumed and small steps must be verified—for example, that min is commutative. One way to alleviate this burden is to develop libraries of theorems. Most prover libraries deal with mathematics in general, defining properties of sets, functions, and numbers. Our contribution is a library that deals with concurrent composition, and distributed systems.

The organization of our library closely follows the organization of this paper, and PVS proves to be an adequate tool in this regard. Inheritance in PVS is achieved through assumptions and parametrized theories [MB97, dJvdPH00]. Parametrization within PVS is a means of making abstract types within a package concrete. Assumptions are theorems that are assumed to be true within a package.[1] These theorems must be discharged once the package is imported by another package. For example, to prove properties of fold we assume that the operator is commutative, associative, and monotonic—PVS assumptions are a way to force those that inherit from fold to actually discharge these assumptions. Figure 5 provides an overview of our verification methodology in PVS. Throughout the section, we represent agents with an uninterpreted type A and the type of their state with an uninterpreted type T. States of the distributed system are functions from A to T.

Local–global relations are at the root of our theoretic hierarchy (lines 1–9). The package is parametrized with the types A and T, a function f from states and sets of agents to T, and a transitive relation $>$ over the agent-value type. This package assumes f to be local–global as denoted by the assumption `f_local_global`, which is LG2 of Definition 8. The fold package (lines 11–26) provides a declaration of fold as outlined in Definition 9. This theory is parametrized with A, T, a binary operator $\circ$, and a transitive relation $>$. The binary operator is a function

[1] We use the term "package" to refer to a PVS theory file.

```
1  local_global[A: TYPE, T: TYPE, f: FUNCTION[[[A -> T], finite_set[A]] -> T], >: (transitive?[T])]:
2  THEORY BEGIN
3   ASSUMING
4    f_local_global: ASSUMPTION
5     FORALL (pre, post: [A -> T], K: finite_set[A], k: A | NOT member(k, K)):
6       f(pre, K) > f(post, K) AND pre(k) = post(k) IMPLIES f(pre, add(k, K)) > f(post, add(k, K))
7    ENDASSUMING
8    % ... code cut for brevity
9  END local_global
10
11 fold[A: TYPE, T: TYPE, o: FUNCTION[T, T -> T], >: (transitive?[T])]:
12 THEORY BEGIN
13   ASSUMING
14    commutative: ASSUMPTION FORALL (u, v: T): u o v = v o u
15    associative: ASSUMPTION FORALL (u, v, w: T): u o (v o w) = (u o v) o w
16    monotonic:   ASSUMPTION FORALL (u, v, w: T): u > v IMPLIES u o w > v o w
17   ENDASSUMING
18
19   fold(S: [A -> T], K: finite_set[A]): RECURSIVE T =
20     IF singleton?(K) THEN S(the(K))
21     ELSE fold(S, rest(K)) o S(choose(K))
22     ENDIF
23   MEASURE card(K)
24
25   IMPORTING local_global[A, T, fold, >]
26 END fold
27
28 ifold[A: TYPE, T: TYPE, o: FUNCTION[T, T -> T], >: (transitive?[T])]:
29 THEORY BEGIN
30   ASSUMING
31    idempotent:  ASSUMPTION FORALL (u: T): u o u = u
32
33    commutative: ASSUMPTION FORALL (u, v: T): u o v = v o u
34    associative: ASSUMPTION FORALL (u, v, w: T): u o (v o w) = (u o v) o w
35    monotonic:   ASSUMPTION FORALL (u, v, w: T): u > v IMPLIES u o w > v o w
36   ENDASSUMING
37   % ...
38   IMPORTING fold[A, T, o, >]
39 END ifold
40
41 min[A: TYPE]:
42 THEORY BEGIN
43  IMPORTING ifold[A, real, min, =]  % 'min' is defined in the PVS prelude
44 END min
```

Fig. 5. A summary of our methodology in PVS for distributed consensus (Sect. 4.2). The model includes the following theories: local global, fold, ifold, and min

from $T \times T$ to T. The assumptions on lines 13 through 17 correspond to Theorem 3.5. By importing the local global package (line 25), the fold package is required to show `f_local_global`, formally proved in Theorem 3.5. In PVS, like in our hand proof, this is done using the commutativity, associativity, and monotonicity operator assumptions, along with some algebraic manipulation. The ifold package (lines 28–39) is a specialization of fold. It accepts the same parameters, but adds a restriction that $\circ$ be idempotent. This assumption allows us to prove the correctness of distributed consensus protocol presented in Sect. 4.2. Lines 33–35 are repeated from fold (lines 13–17) since they cannot be discharged at this level. In our library, the ifold package is instantiated with several examples, including min, max, gcd, lcm, and convex hull. The min example is presented on line 43; it is parametrized with an agent type. When importing ifold, the type T is set to real numbers, $\circ$ is the min operator (a standard PVS function), and the conservation relation $>$ is equality. When we import ifold, we have to discharge the commutative, associative, idempotent, and monotonic assumptions with respect to min.

Implementing the entire package required 57 lemmas and approximately 800 proof steps.[2] Almost 60% of our work went into developing a library of relatively simple lemmas that we used repeatedly, such as algebraic properties about fold. Approximately 15% of work in PVS was devoted to proving that the concrete operators,

[2] We use "lemmas" and "steps" to describe PVS metrics. A lemma count gives some idea of the complexity of a proof. Steps are the number of commands required to discharge a lemma. These are highly subjective metrics; they are intended to provide informal measures of work.

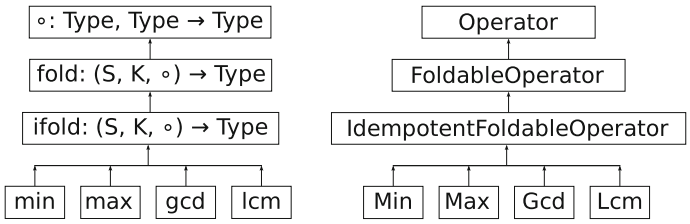


Fig. 6. A comparison of the PVS library structure (*left*) and Java object refinement (*right*). The correspondence between the development design is one-to-one

such as gcd, satisfied the assumptions made of the abstract operator ∘. Future work will be reduced as these libraries develop.

The example presented in Sect. 4.1, distributed average, has also been implemented in PVS. We also have libraries representing automata and concurrent systems as defined in Sects. 2 and 3.1. We proved sufficient conditions for system safety and progress, outlined in Theorems 3.7 and 3.11, respectively. All of the proofs used for dealing with convergence and stability of continuous systems (Theorems 2.2 and 2.3) were proved in PVS. Proofs of convergence of mobile agents to the straight-line formation, Sect. 5.4, were carried out in PVS as well. However, mechanically proving the general problem of solving systems of linear equations using mobile wireless agents is ongoing.

Although we were confident in our hand-proofs, checking our work in a theorem prover was a valuable exercise. First, it forced us into a very particular kind of thought process. On occasion, we had to rethink our proof based on an inability to show it in PVS. Often, the result of such rethought was a simplification in our approach. Complexity in our model comes from simple building blocks, which ultimately made the implementation, and subsequent debugging during implementation, much easier. The second advantage of working with a theorem prover was that it provided a tangible representation of our verification methodology. Rather than having several ideas that were loosely related, when implemented in PVS we had to think about the problem in a more structured fashion—exactly how fold fit into the local–global idea had to be well defined. Such structure ultimately improved our methodology.

7. Java implementation

Based on our PVS model, we have transformed the examples from Sect. 4 into executable programs written in Java. Our goal was to create correct software based on the layering method used in verification. Unlike more automated efforts [BPH07, DF06, Sac08, Ken08], our transformations were done manually.

In this section we look specifically at our Java implementation of the consensus example from Sect. 4.2. It consists of three base classes: one representing operators, another for agents, and a third for communication. The operator class is abstract, refined to obtain specific functions of interest, such as max and gcd. This is analogous the operators abstract representation in PVS, and its instantiation by concrete operators. As shown in Fig. 6, the correspondence between the PVS library structure and Java object refinement is one-to-one. The agent class represents a single agent in our system; each is a separate thread, with multiple agents communicating via message passing.

The communication layer is necessary to compose groups of agents. In our case, it also provided agents with an atomic environment in which to share and update their state. This layer in the implementation is not formally verified in our model; we refer the reader to the wide body literature which addresses this concern (see, for example, [Kha04, ADKM92, Rei96, HJR04, JS04]). We assume that the environment—for example the communication medium—satisfies certain properties and prove that the overall system consisting of agents and the environment satisfy other properties. In the Java implementation we implemented both the agents and we simulated the environment. In our simulation of the communication medium, a group of agents upholds the fairness guarantee our model requires (recall Definition 2). Groups are created using Java's random number generator (JRNG): given the entire set of agents within the system, random agents to are assigned to a single group (agents without a group assignment are thought of as being in a group consisting of only themselves). Thus, we assume that over the course of the system lifetime, JRNG will choose each agent at least once. We did not prove that JRNG satisfies Definition 2.

Our verification methodology produces several modular components whose correctness is well understood—both in their assumptions and in their execution guarantees. Knowing these components helps to direct the focus

```
1  min(m, n): {p: real | p <= m AND p <= n} =
2    IF m > n THEN n
3    ELSE m
4    ENDIF
```

```
1  /*@
2    @ assert(\result <= x && \result <= y &&
3    @        (\result == x || \result == y));
4    @*/
5  double min(double m, double n) {
6    return Math.min(m, n);
7  }
```

Fig. 7. Implementation of min in PVS (*left*) and Java (*right*). The Java implementation is annotated with JML

of implementation verification. In this example, there are two aspects of the implemented system we must verify: that the concrete operators in Java are equivalent to their PVS representations, and that fold is conserved in each state transition (Lemma 4.6). Our Java implementation uses the Java Modeling Language (JML) [BCC+05] to enforce system verifications. In general, JML is used to specify the behavior of Java objects, and check during run-time that those specifications are upheld. The tool allows programmers to prepend object methods with pre- and post-conditions, and annotate arbitrary lines of code with system invariants. An advantage of JML is its support for abstract programming concepts that are closer to mathematics than standard Java, such as set theory and quantification. An example of how JML is used to verify the Java min operator is outlined in Fig. 7. The figure on the left is the PVS representation of min; on the right is its representation in Java (lines 5–7), annotated with JML (lines 1–4). Based on the min implementation in PVS, a valid functional transform is one that places a restriction on its return type (left line 1), and whose value is equal one of its input parameters. Both are captured with the JML post-condition checks on (right) lines 2 and 3, respectively. These assertions alleviate the need to be concerned with how min is implemented in Java; thus, in the transformation from PVS to correct software, the crucial step is the JML.

8. Conclusions

In this paper we have proposed a methodology for verifying concurrent systems whose correctness can be expressed through local–global relations. These properties ensure that when groups of agents take steps that change their local states, global properties of the system are preserved. We have provided sufficient conditions for proving convergence and termination for these systems. Our proof obligation is to show (1) *safety*: that local–global relation is preserved, and (2) *progress*: there is a set of fair actions whose execution reduces a distance (Lyapunov) function by at least a constant amount. We have shown how our methodology is applied to three examples: distributed consensus and average without failure, and multi-agent cooperation in solving systems of linear equations when agent communication is faulty. The theorems and algorithms presented in this paper have been mechanically checked using PVS, producing a reusable library of theorems [GMPJ09]. Finally, the PVS algorithms have been translated into Java.

Our methodology can be applied to systems with both discrete and continuous state spaces, as shown by our examples. The theory has also been applied to systems that operate in continuous state spaces and over continuous time intervals. Our examples have focused on distributed control problems where the application of our methodology allowed us to give a better formalization, and further generalization, of known results in the field. For example, in the average consensus problem we proved correctness by identifying a local–global property, namely that average, and a decreasing mean square error, were locally maintained. In the systems of linear equation example, our methodology allowed us to weaken the strictly diagonally dominant assumption on the matrix, and to deal with continuous state changes of agents.

An area of further work is to develop algorithms and theory for proving distributed control problems that operate in adversarial environments. Another area is to extend our PVS library to richer classes of distributed problems, or even extend the library to another theorem prover entirely. Finally, we would like to automate the translations of PVS to Java, and PVS lemmas and assumptions to JML.

Acknowledgments

The authors would like to thank the referees for their very helpful and constructive comments. This work was supported in part by the Multidisciplinary Research Initiative (MURI) from the Air Force Office of Scientific Research.

References

[Abr96] Abrial JR (1996) The B-book: assigning programs to meanings. Cambridge University Press, New York

[ADKM92] Amir Y, Dolev D, Kramer S, Malki D (1992) Membership algorithms for multicast communication groups. In: Proceedings of the 6th international workshop on distributed algorithms (WDAG '92). Lecture notes in computer science, vol 647. Springer, Berlin, pp 292–312

[AHS98] Archer M, Heitmeyer C, Sims S (1998) TAME: a PVS interface to simplify proofs for automata models. In: Proceedings of the 1st international workshop on user interfaces for theorem provers (UITP '98), July 1998

[Arc00] Archer M (2000) TAME: using PVS strategies for special-purpose theorem proving. Ann Math Artif Intell 29(4):139–181

[AW04] Attiya H, Welch J (2004) Distributed computing: fundamentals, simulations and advanced topics. Wiley, New York

[BCC+05] Burdy L, Cheon Y, Cok DR, Ernst MD, Kiniry JR, Leavens GT, Leino KRM, Poll E (2005) An overview of JML tools and applications. Int J Softw Tools Technol Transf 7(3):212–232

[BHOT05] Blondel VD, Hendrickx JM, Olshevsky A, Tsitsiklis JN (2005) Convergence in multiagent coordination consensus and flocking. In: Proceedings of the joint 44th IEEE conference on decision and control and european control conference (CDC-ECC '05). IEEE Computer Society, Washington, DC, pp 2996–3000

[BKN07] Bulwahn L, Krauss A, Nipkow T (2007) Finding lexicographic orders for termination proofs in isabelle/hol. In: Proceedings of the 20th international conference on theorem proving in higher order logics (TPHOLs '07). Lecture notes in computer science, vol 4732. Springer, Berlin, pp 38–53

[BPH07] Blech JO, Poetzsch-Heffter A (2007) A certifying code generation phase. Electron Notes Theor Comput Sci 190(4):65–82

[BS96] Back RJR, Sere K (1996) Superposition refinement of reactive systems. Formal Aspects Comput 8(3):324–346

[CM69] Chazan D, Miranker W (1969) Chaotic relaxation. Linear Algebra Appl 2(2):199–222

[CM88] Chandy KM, Misra J (1988) Parallel program design: a foundation. Addison-Wesley Longman Publishing Co., Inc., Boston

[CMP08] Chandy KM, Mitra S, Pilotto C (2008) Convergence verification: from shared memory to partially synchronous systems. In: Proceedings of 5th international conference on formal modeling and analysis of timed systems (FORMATS '08). Lecture notes in computer science, vol 5215. Springer, Berlin, pp 217–231

[CS77] Chatterjee S, Seneta E (1977) Towards consensus: some convergence theorems on repeated averaging. J Appl Probab 14(1): 89–97

[DF06] Denney E, Fischer B (2006) Extending source code generators for evidence-based software certification. In: Proceedings of the 2nd international symposium on leveraging applications of formal methods, verification and validation (ISoLA '06). IEEE Computer Society, Washington, DC, pp 138–145

[Dij68] Dijkstra EW (1968) A constructive approach to the problem of program correctness. BIT 8:174–186

[Dij76] Dijkstra EW (1976) Executional abstraction, chapter 0. Prentice-Hall, New Jersey

[dJvdPH00] de Jong E, van de Pol J, Hooman J (2000) Refinement in requirements specification and analysis: a case study. In: Proceedings of the 7th IEEE international conference and workshop on the engineering of computer based systems (ECBS '00). IEEE Computer Society, Washington, DC, pp 290–298

[Flo67] Floyd R (1967) Assigning meanings to programs. In: Symposium on applied mathematics. mathematical aspects of computer science. American Mathematical Society, Providence, pp 19–32

[GM80] Gabay D, Moulin H (1980) On the uniqueness and stability of nash equilibria in non-cooperative games. In: Applied stochastic control of econometrics and management science. North-Holland, Amsterdam, pp 271–293

[GMPJ09] Go B, Mitra S, Pilotto C, White J (2009) Infospheres project. http://www.infospheres.caltech.edu/facj

[Got00] Gottliebsen H (2000) Transcendental functions and continuity checking in pvs. In: Proceedings of the 13th international conference on theorem proving in higher order logics (TPHOLs '00), Lecture notes in computer science, vol 1869. Springer, Berlin, pp 197–214

[Har98] Harrison J (1998) Theorem proving with the real numbers. Springer, Berlin

[HJR04] Huang Q, Julien C, Roman GC (2004) Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. IEEE Trans Mobile Comput 3(2):192–205

[Hor03] Horowitz B (2003) Giotto: a time-triggered language for embedded programming. PhD thesis, University of California, Berkeley

[Jac00] Jackson PB (2000) Total-correctness refinement for sequential reactive systems. In: Proceedings of the 13th international conference on theorem proving in higher order logics (TPHOLs '00). Lecture notes in computer science, vol 1869. Springer, Berlin, pp 320–337

[JLM03] Jadbabaie A, Lin J, Morse AS (2003) Coordination of groups of mobile autonomous agents using nearest neighbor rules. IEEE Trans Autom Control 48(6):988–1001

[JS04] Jain A, Shyamasundar RK (2004) Failure detection and membership management in grid environments. In: Proceedings of the 5th IEEE/ACM international workshop on grid computing (GRID '04). IEEE Computer Society, Washington, DC, pp 44–52

[Ken08] Kennedy KE (2008) Caps: concurrent automatic programming system. PhD thesis, Clemson University, Clemson, SC, USA

[Kha04] Khazan RI (2004) Group membership: a novel approach and the first single-round algorithm. In: Proceedings of the 23rd annual ACM symposium on principles of distributed computing (PODC '04). ACM, New York, pp 347–356

[KLSV06] Kaynar DK, Lynch NA, Segala R, Vaandrager F (2006) The theory of timed I/O automata (Synthesis lectures in computer science). Morgan & Claypool Publishers, Oxford

[Les] Lester D, NASA langley PVS library for topological spaces. http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/topology-details.html

[Lib03] Liberzon D (2003) Switching in systems and control. Systems and control: foundations and applications. Birkhauser, Boston

[LKLM05] Lim H, Kaynar D, Lynch NA, Mitra S (2005) Translating timed I/O automata specifications for theorem proving in PVS. In: Proceedings of the 3rd international conference on formal modelling and analysis of timed systems (FORMATS '05). Lecture notes in computer science, vol 3829. Springer, Berlin,

[LS91] Lam S, Shankar AU (1991) A composition theorem for layered systems. In: Proceedings of the IFIP WG6.1 international symposium on protocol specification, testing and verification XI. North-Holland, Amsterdam, pp 93–108
[LT87] Lynch NA, Tuttle MR (1987) Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the 6th annual ACM symposium on principles of distributed computing (PODC '87). ACM, New York, pp 137–151
[LT89] Lynch NA, Tuttle MR (1989) An introduction to input/output automata. CWI-Quarterly 2(3):219–246
[Lue79] Luenberger DG (1979) Introduction to dynamic systems: theory, models, and applications. Wiley, New York
[Lya66] Lyapunov AM (1966) Stability of motion. Academic Press, New York
[Lyn96] Lynch NA (1996) Distributed algorithms. Morgan Kaufmann, San Francisco
[MA05] Mitra S, Archer M (2005) PVS strategies for proving abstraction properties of automata. Electron Notes Theor Comput Sci 125(2):45–65
[MB97] Maharaj S, Bicarregui J (1997) On the verification of vdm specification and refinement with pvs. In: Proceedings of the 12th international conference on automated software engineering (ASE '97). IEEE Computer Society, Washington, DC, p 280
[MC08] Mitra S, Chandy KM (2008) A formalized theory for verifying stability and convergence of automata in PVS. In: Proceedings of the 21st international conference on theorem proving in higher order logics (TPHOLs '08). Lecture notes in computer science, vol 5170. Springer, Berlin, pp 230–245
[Mit07] Mitra S (2007) A Verification Framework for Hybrid Systems. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007
[Mor87] Morris JM (1987) A theoretical basis for stepwise refinement and the programming calculus. Sci Comput Program 9(3):287–306
[ORS92] Owre S, Rushby JM, Shankar N (1992) PVS: a prototype verification system. In: Kapur D (ed) Proceedings of 11th international conference on automated deduction (CADE '92). Lecture notes in artificial intelligence, vol 607. Springer, Saratoga, pp 748–752
[OS07] Olfati-Saber R (2007) Distributed kalman filtering for sensor networks. In: Proceedings of the 46th IEEE conference on decision (CDC '07). IEEE Computer Society, Washington, DC, pp 5492–5498
[OSFM07] Olfati-Saber R, Fax JA, Murray RM (2007) Consensus and cooperation in networked multi-agent systems. Proc IEEE 95(1):215–233
[Pnu77] Pnueli A (1977) The temporal logic of programs. In: 18th Annual Symposium on foundations of computer science, pp 46–57
[Rei96] Reiter MK (1996) A secure group membership protocol. IEEE Trans Softw Eng 22(1):31–42
[RP96] Rohwedder E, Pfenning F (1996) Mode and termination checking for higher-order logic programs. In: Proceedings of the 6th European symposium on programming languages and systems (ESOP '96). Lecture notes in computer science, vol 1058. Springer, Berlin, pp 296–310
[Sac08] Sacha K (2008) Model-based implementation of real-time systems. In: Proceedings of the 27th international conference on computer safety, reliability, and security (SAFECOMP '08). Lecture notes in computer science, vol 5219. Springer, Berlin, pp 332–345
[Spi07] Spichkova M (2007) Specification and seamless verification of embedded real-time systems: FOCUS on Isabelle. PhD thesis, Technische Universität München
[Spi08] Spichkova M (2008) Refinement-based verification of interactive real-time systems. Electron Notes Theor Comput Sci 214:131–157
[Tau05] Tauber JA (2005) Verifiable compilation of I/O automata without global synchronization. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA
[Tsi87] Tsitsiklis JN (1987) On the stability of asynchronous iterative processes. Theory Comput Syst 20(1):137–153
[UL07] Umeno S, Lynch NA (2007) Safety verification of an aircraft landing protocol: a refinement approach. In: Proceedings of the 10th international conference on hybrid systems: computation and control (HSCC '07). Lecture notes in computer science, vol 4416. Springer, Berlin, pp 557–572
[Wir71] Wirth N (1971) Program development by stepwise refinement. Commun ACM 14(4):221–227

Received 28 February 2009
Revised 1 October 2009
Accepted 8 March 2010 by T. Margaria, D. Kröning, and J. Woodcock
Published online 9 April 2010