

# Guaranteeing Convergence of Distributed Systems: From Specification to Implementation via Refinement

K. Mani Chandy<sup>1</sup>, Brian Go<sup>1</sup>, Sayan Mitra<sup>2</sup>, Concetta Pilotto<sup>1</sup> and Jerome White<sup>1</sup>

<sup>1</sup>California Institute of Technology

{mani,bgo,pilotto,jerome}@cs.caltech.edu

<sup>2</sup>University of Illinois at Urbana Champaign

mitras@crhc.uiuc.edu

**Abstract.** This paper describes a methodology for developing and verifying a class of distributed systems using stepwise refinement and a theorem prover. The process of refinement leads to algorithms in which individual steps can be implemented atomically in a straightforward way. These algorithms are then transformed from the notation of the theorem prover to a target programming language, such as Java and Erlang. The methodology allows the system state space to be continuous. The key temporal properties that are preserved by the above refinement and transformation processes are *convergence* and *termination*. The proof techniques used are extensions of control theoretic results to temporal logic of continuous time and state spaces. We present a library of theorems and proofs to reduce the work required to develop and verify programs in this class. The applicability of the method is demonstrated by modeling and performing step-wise refinement of a collection of standard algorithms for sensor networks and multi-vehicle systems.

**Keywords:** concurrency · continuous systems · convergence · stability · refinement · automatic theorem prover

## 1. Introduction

Verification and validation of distributed systems, in which multiple agents cooperate to control physical systems, is becoming increasingly important. For example, research in distributed control is focusing on electrical power grids, air traffic, and multi-vehicle systems. Our challenge is to verify correctness of these types of systems, that have both discrete and continuous components and operate over discrete and continuous time.

Verifying correctness of distributed systems at a level of abstraction that includes implementation details is difficult because of large state-spaces and nondeterminism. The problem becomes much harder if the system model includes

infinite or uncountable data-types, such as, stacks, queues, and real-valued variables (e.g., positions). These types are often required for describing the behavior of sensor networks and autonomous vehicle systems. In order to circumvent this problem of verification, the *model-based* approach has been proposed in the literature [Tau05, Hor03]. The idea is to first develop algorithms at a relatively abstract mathematical level, then translate or refine the models to real implementations in a provably correct manner; formally, this is known as *stepwise refinement*. Early work on refinement includes [Wir71, Mor87], more generally examples of refinements are, in the context of distributed systems and I/O automata [LT87, Spi08, Spi07], and in reactive and parallel systems [Abr96, BS96, CM88].

This paper proposes a methodology for model-based development of *concurrent systems*, a special class of distributed systems. Within this class, we are specifically interested in the temporal properties *convergence* and *termination*. These properties are relevant in distributed control algorithms for tracking [OS07] and pattern formation [OSFM07], along with flocking [BHOT05]. Informally, the system converges to a desired state if the state gets arbitrarily close to a desired value as time tends to infinity [Lue79]. Termination is a related property of distributed systems which requires that a desired state is actually reached in a finite number of steps. Most traditional distributed algorithms, such as those for achieving consensus or for electing a leader [Lyn96, AW04], require termination.

In our framework, we develop algorithms at an abstract mathematical level. First, we prove convergence and termination using standard techniques, such as providing the existence of Lyapunov-like functions [CS77, BHOT05, Lib03]. Our mathematical models are then refined step-by-step, to obtain implementations that can be deployed on a target distributed computing platform, for example, a message passing network.

We encode theorems, specifications, and algorithm abstractions in the theorem prover PVS [ORS92], developing refinement libraries for proving convergence and termination of concurrent systems. Our meta-theories are applicable to a general class of problems and are extensions of theorems from control theory to temporal logic [MC08]. Our theory builds on the PVS formalization of input/output automata [LT89, AHS98] and its subsequent 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]. Refinement within a theorem prover has been studied for specific case studies [Jac00, dJvdPH00, MB97].

We present several case studies based on this methodology. In our examples, algorithms are specified using abstract operators. We prove that these algorithms are correct provided the operators have certain properties such as associativity and commutativity. Given a specification with an instance of the abstract operator, we only need verifying that the instance satisfies the property of the abstract operator. Each step of an algorithm encoded in PVS is transformed into a step of a program encoded in Java. Specifically, stepwise refinement is carried out until each step of the algorithm in PVS is straightforward to implement in Java as an atomic step. 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 also discuss *message passing* systems with bounded delay. As we show, these are a subclass of concurrent systems, therefore, our results on convergence and termination apply directly. We prove that message passing algorithms for solving systems of linear equations, such as Gauss and Gauss-Seidel, converge to the correct solution. We show that distributed algorithms for mobile agents may be casted as distributed algorithms for solving systems of linear equations; for example, agent pattern formation algorithms where agents want to form a straight line. Our results extend the work of [GM80, CM69, CMP08].

The paper is organized as follows. Section 2 presents basic definitions for automata and proves sufficient conditions for termination and convergence. Section 3 outlines our method of refinement. Section 4 presents two applications from control theory and proves their termination and convergence. 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 gives concrete Java implementation examples of our theoretical results. Section 8 concludes the paper.

## 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  $\mathbb{N} = \{0, 1, \dots\}$ , and the set of reals by  $\mathbb{R}$ . For a set  $A$ ,  $A^\omega$  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  $\{0, 1, \dots, N-1\}$ ;  $n, m \in \mathbb{N}$  denote arbitrary natural numbers.

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

**Definition 1.** An automaton  $\mathcal{A}$  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  $E : [S, A \rightarrow \mathbb{B}]$ , and
- (e) a transition function  $T : [S, A \rightarrow S]$ .

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 and indeed actions can label functions for discrete transitions as well as continuous evolution. We refer the reader to [LKLM05] for models of timed and hybrid systems in this formalism.

Hereafter,  $\mathcal{A}$  denotes an automaton with parameters  $\langle S, A, S_0, E, T \rangle$ ;  $s, s^* \in S$ ,  $s_0 \in S_0$  denote arbitrary (initial) states of the automaton;  $S^*$  a nonempty subset of  $S$ ;  $a \in A$  an arbitrary action of the automaton.

The semantics of an automaton  $\mathcal{A}$  are defined in terms of its executions. An *execution fragment* of  $\mathcal{A}$  is a possibly infinite alternating sequence of states and actions  $s_0, a_0, s_1, a_1, s_2, \dots$ , 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  $a_0, a_1, \dots, a_n$  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

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

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

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

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

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 have to 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  $a_0, a_1, \dots \in A^\omega$  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  $\alpha = s_0, a_0, s_1, a_1, \dots$  is said to be  $\mathcal{F}$ -fair if the corresponding sequence of actions  $a_0, a_1, \dots$  is  $\mathcal{F}$ -fair.

In other words, 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. For example, the fairness criterion could be that the system is not permanently partitioned into non-communicating (nonempty) subsets; though the system may be partitioned into non-communicating subsets

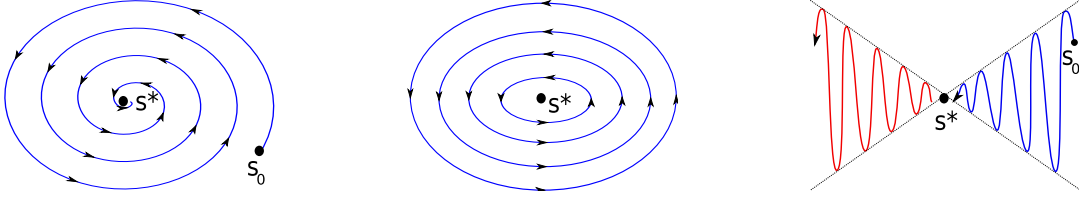


Fig. 1. Stable and convergent (*left*), stable and nonconvergent (*middle*) and convergent and unstable (*right*) executions. In the last case, the execution from  $s_0$  converges, but executions starting from the left neighborhood of  $s^*$  diverge.

temporarily. In this case for each partition  $\{J, \bar{J}\}$  we define  $\mathcal{F}$  to include a set  $F$  of actions where each action in  $F$  operates on at least on agent in  $J$ , and one agent in  $\bar{J}$ .

## 2.2. Stability, Convergence, and Termination

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

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

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

where  $d : [s, S \rightarrow \mathbb{R}_{\geq 0}]$  is a *distance function* for  $s^*$  satisfying the following:

$$\forall s \neq s^*, d(s^*, s) > d(s^*, s^*).$$

The  $\epsilon$ -balls around a given state  $s$  define a topological structure around  $s$ . We refer to [MC08] for a discussion on stability and convergence with respect to arbitrary topologies.

### 2.2.1. Stability

Informally, an equilibrium state  $s^*$  is stable for  $\mathcal{A}$  if every execution fragment that starts close to  $s^*$  remains close to  $s^*$ , where closeness is defined in terms of the  $\epsilon$ -balls around  $s^*$ . Examples of stable executions are depicted in Figure 1.

**Definition 4.**  $s^*$  is *stable* for  $\mathcal{A}$  if

$$\forall \epsilon > 0, \exists \delta > 0, \text{ReachFrom}(B_\delta(s^*)) \subseteq B_\epsilon(s^*).$$

Note that stability is independent of the starting states of the automaton. The definitions for the  $\epsilon$ -balls around  $S^*$  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  $\text{Range}(f)$ , 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.

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

- B1.  $\forall \epsilon \geq 0, \exists p \in P$ , such that  $L_p \subseteq B_\epsilon(S^*)$ .
- B2.  $\forall p \in P, \exists \epsilon \geq 0$ , such that  $B_\epsilon(S^*) \subseteq L_p$ .
- B3.  $\forall s, a \ E(s, a) \implies f(T(s, a)) \leq f(s)$ .

Then  $S^*$  is stable for  $\mathcal{A}$ .

B1 requires that every  $\epsilon$ -ball around  $S^*$  contains a  $p$ -sublevel set  $L_p$ . B2 is symmetric—it requires that every sublevel set contains an  $\epsilon$ -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.

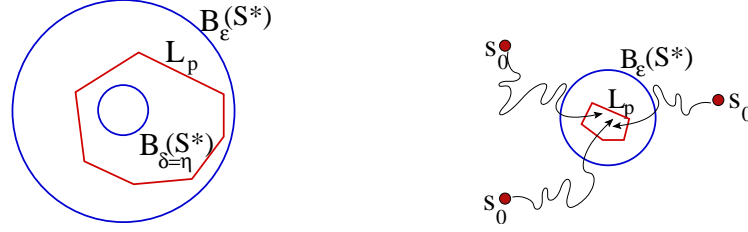


Fig. 2. A graphical representation of the proof of Theorem 2.2 (left) and of Theorem 2.3 (right).

*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  $B_\delta(S^*)$  remains within  $B_\epsilon(S^*)$ .

From B1,

$$\exists p \in P : L_p \subseteq B_\epsilon(S^*),$$

and from B2,

$$\exists \eta \geq 0 : B_\eta(S^*) \subseteq L_p \subseteq B_\epsilon(S^*).$$

We refer to Figure 2 for a pictorial representation. Set  $\delta = \eta$ . From B3,

$$\text{ReachFrom}(L_p) = L_p.$$

Since  $B_\delta(S^*) \subseteq L_p$ , we get that

$$\text{ReachFrom}(B_\delta(S^*)) \subseteq L_p \subseteq \text{ReachFrom}(B_\epsilon(S^*)).$$

□

### 2.2.2. Convergence

An automaton  $\mathcal{A}$  converges to  $s^*$ , if for every fair infinite execution of  $\mathcal{A}$ , the automaton gets closer to  $s^*$ . See Figure 1 for examples of convergent and divergent executions.

**Definition 5.**  $\mathcal{A}$  converges to  $s^*$ , if  $\forall \epsilon \diamond \square s \in B_\epsilon(s^*)$ .

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

We define  $\mathcal{P}$  as the set of values in  $P$  that are reached eventually:

$$\mathcal{P} := \{p \in P \mid \diamond (s \in L_p)\}.$$

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

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

- C1.  $\forall p, q \in P, p < q \implies L_p \subsetneq L_q$ .
- C2.  $\forall \epsilon > 0, \exists p \in P$ , such that  $L_p \subseteq B_\epsilon(S^*)$ .
- C3.  $\forall s, a, (\text{reachable}(s) \wedge E(s, a)) \implies f(T(s, a)) \leq f(s)$ .
- C4.  $\forall p \in P, L_p \neq S^*$  implies  $\exists F \in \mathcal{F}$ , such that  $\forall a \in F, s \in L_p, \text{reachable}(s) \implies (E(s, a) \wedge f(T(s, a)) < f(s))$ .
- C5.  $\forall P' \subseteq \mathcal{P}$ , if  $P'$  is lower bounded then it has a smallest element.

Then  $\mathcal{A}$  converges to  $S^*$

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  $\epsilon$ -ball around  $S^*$ . 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  $S^*$ , 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  $\implies \text{Range}(f) = \mathcal{P}$ .

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

*Proof of Theorem 2.3* Let us fix  $\epsilon \geq 0$ ,  $f$  satisfying the conditions in the hypothesis. By C2,

$$\exists p \in \text{Range}(f) : L_p \subseteq B_\epsilon(S^*).$$

By Lemma 2.4,

$$\diamond (s \in L_p),$$

see Figure 2. By C3,

$$\text{ReachFrom}(L_p) = L_p.$$

Hence,

$$\diamond \square s \in L_p,$$

which implies

$$\diamond \square s \in B_\epsilon(S^*).$$

□

**Special case.** In certain applications the function  $d$  which defines the topological structure around  $s^*$  can itself be used as the Lyapunov-like function for proving convergence. The obvious advantage of doing so is that C2 follows automatically. We provide a restricted version of Theorem 2.3 which can be applied in such cases.

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

$$\text{D1. } \forall i, j \in \mathbb{N}, i > j \implies L_{p_i} \subsetneq L_{p_j}.$$

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

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

Then  $\mathcal{A}$  converges to  $S^*$ .

### 2.2.3. Termination

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

**Definition 6.**  $\mathcal{A}$  *terminates* in  $s^*$ , if  $\diamond \square s = s^*$ .

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

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

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

$$\text{E1. } \forall i, j \in [n], i > j \implies L_{p_i} \subsetneq L_{p_j}.$$

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

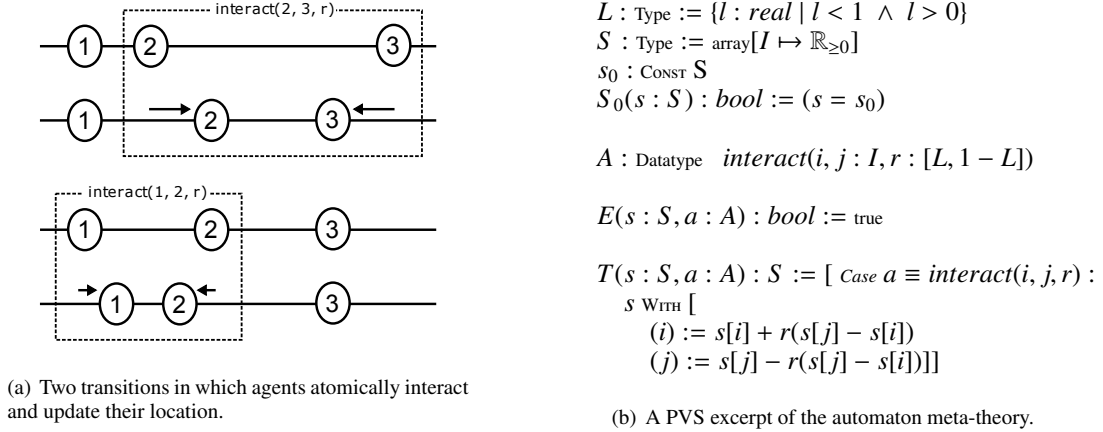


Fig. 3. Representations of the line convergence algorithm.

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

Then  $\mathcal{A}$  terminates in  $S^*$ .

This is analogous to corollary 2.5 where the sequence of evaluations of  $f$  is finite. We can also derive a theorem analogous to Theorem 2.3.

### 3. Refinement Techniques for Concurrent Systems

In this section, we introduce our refinement methodology as applied to concurrent systems. We discuss theoretical results of our methodology, and provide examples that reinforce its relevance.

#### 3.1. Automaton Formulation

A concurrent system is an automaton (Definition 1) with additional structure. The system has  $N$  agents, where  $N$  is a positive (constant) integer. The agents are indexed  $j$  for  $0 \leq j < N$ . In line with Section 2, we define the automaton as follows:

- (a) A state  $s \in S$  is an array with  $N$  elements, where  $s[j]$  is the state of the  $j$ -th agent.
- (b) Associated with each action  $a \in A$  is a nonempty set  $\text{agents}(a)$  of agents whose states could be modified by the occurrence of action  $a$ . Action  $a$  only reads the states of agents that are in  $\text{agents}(a)$ . We use the notation  $s_a$  for  $s[\text{agents}(a)]$ .
- (c) Start states are problem dependent.
- (d) The enabled predicate,  $E$ , depends only on the states of agents that participate in the action. We define an enabling condition  $e$  for an action  $a$  as

$$\forall s, a : e(s_a, a) = E(s, a)$$

- (e) The transition function  $T$  depends only on the states of agents that participate in the action. We define the transition function  $t$  of an action  $a$  as

$$\forall s, a : t(s_a, a) = s'_a \quad \text{where } s' = T(s, a)$$

where the pre-states,  $s$ , and post-states,  $s'$ , are restricted to the agents that participate in the action.

**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 each other to reduce the distance between them by at least  $2L$ , where  $L$  is a constant real in the range  $(0, 1)$ . An iteration of this algorithm can be seen in Figure 3(a).

Table 3(b) 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 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  $S$ . Other elements of  $S$  are considered start states if they are equal to  $s_0$ , noted with the predicate  $S_0$ . The real value  $r$  regulates the distance a given agent travels. Note that  $r$  is a member of an uncountable set  $[L, 1 - L]$ .

In this example, two agents can always interact, because the enabling predicate,  $E(s, a)$ , is always true. Finally, 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  $s[i] + r(s[i] - s[j])$  and  $s[j] - r(s[i] - s[j])$ , respectively. ■

### 3.2. Local-Global Relations

A key idea of our refinement technique is the concept of local-global relations. These relations aid in the proof of stability, convergence, and termination. Local-global properties are properties whereby state transitions of agents that participate in an action, then the property also holds for state transitions of the entire system, including agents that did not participate in the action.

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 an ordered nonempty set of indices of array  $x$  or  $x'$ , ordered in increasing order. Let  $\bar{J}$  be the complement of  $J$ . For an array  $x$ , let  $x[J]$  be the array obtained by projecting  $x$  on to  $J$ : then  $x[J]$  has the same number of elements as  $J$ , and the  $k$ -th element of  $x[J]$  is  $x[J[k]]$ .

**Example 3.2** If  $M = 5$  and  $J = [0, 2, 3]$  then  $\bar{J} = [1, 4]$

$$x[J] = [x[0], x[2], x[3]]$$

$$x[\bar{J}] = [x[1], x[4]]$$

■

A formal definition is as follows

**Definition 7.** 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.  $\forall x, x', J : (x[J] R x'[J]) \wedge (x[\bar{J}] = x'[\bar{J}]) \implies (x R x')$

**Example 3.3** 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.

2. In the case where the elements of  $x$  are in a partial ordering  $\leq$ :

$$x R x' \equiv \min_j x'[j] \leq \min_j x[j]$$

Consider a relation  $R$  defined as

$$x R x' \equiv \min_j x'[j] < \min_j x[j]$$

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

These operators are instances of a larger class of binary operators which obey our aforementioned local-global relation. In order to talk about correctness of algorithms involving such operators, we define a generic operator  $\circ$ , along with a composition function *fold*. Together, they allow us to reason about correctness without talking about specific functionality.

**Definition 8.** Let  $\circ$  be an associative, commutative, dyadic operator on equal-sized arrays of some type. For an array  $x$  with  $m > 0$  elements define  $fold \circ x$  as:

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

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

**Theorem 3.4.** The relation,  $(fold \circ x) = (fold \circ x')$ , is a local-global relation if  $\circ$  is associative and commutative.

*Proof.* Follows from algebraic manipulation.  $\square$

**Example 3.5** Applications of Theorem 3.4:

- Where the elements of  $x$  are reals, since  $+$  is associative and commutative

$$\sum_j x'[j] = \sum_j x[j]$$

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

$$\min_j x'[j] = \min_j x[j]$$

- Where the elements of  $x$  are points on a Cartesian plane and  $C(x)$  is the convex hull of  $x$ , and  $x \circ x'$  returns the extreme points of the convex hull of the union of  $x$  and  $x'$

$$C(x') = C(x)$$

The proof that  $\circ$  is commutative follows directly from the commutativity of set union. The proof that it is associative is straightforward from the definition of convex hulls.

- Each element of  $x$  and  $x'$  is treated as a bag (multiset) containing a single item

$$x' \text{ is a permutation of } x$$

The operator  $\circ$  takes the union of bags. This operator is associative and commutative. If the bags of elements of  $x$  and  $x'$  are equal, then  $x'$  is a permutation of  $x$ . ■

**Definition 9.** Let  $u, u'$  and  $v$  be members of a partial ordering  $\leq$ . The operator  $\circ$  is *monotonic* exactly when:

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

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

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

**Theorem 3.6.** Consider arrays  $x$  and  $x'$  whose elements are members of the partial ordering  $\leq$ , then

1.  $(fold \circ x') \leq (fold \circ x)$  is a local-global relation if  $\circ$  is associative, commutative and monotonic.
2.  $(fold \circ x') < (fold \circ x)$  is a local-global relation if  $\circ$  is associative, commutative and strictly monotonic.

*Proof.* Follows from algebraic manipulation.  $\square$

**Example 3.7** Example applications of Theorem 3.6:

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

$$\sum_j x'[j] < \sum_j x[j]$$

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

$$\bigcup_j x[j] \subseteq \bigcup_j x'[j]$$

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

$$\bigcap_j x[j] \subseteq \bigcap_j x'[j]$$

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:

$$(\forall j : x[j]) \implies (\forall j : x'[j])$$

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

$$(\exists j : x[j]) \implies (\exists j : x'[j])$$

■

### 3.3. Correctness of Concurrent Systems

With the proper abstractions in hand, we can reason about the safety and progress properties that entail algorithmic correctness. In this case safety maintains a weak relation for all actions, while progress ensures that for at least one action, a strict relation holds.

#### 3.3.1. Sufficient Conditions for Proving Safety

Let  $\mathcal{A}$  be a concurrent system. Given a relation  $R$ , and an action  $a \in A$ , the predicate  $a \implies R$  holds if it is always the case that agents within pre and post states effected by the execution of  $a$  satisfy  $R$ . Formally

$$a \implies R \equiv (\forall s, s' : e(s_a, a) \wedge (s'_a = t(s_a, a)) \implies s_a R s'_a)$$

The predicate can be extended to a set of actions,  $A$

$$A \implies R \equiv (\forall a \in A : a \implies R)$$

As shown in the following theorem, the pre and post states of an arbitrary but finite sequence of actions are related by  $R$  if  $R$  is local-global.

**Theorem 3.8.** Given an automaton  $\mathcal{A}$ , and a local-global relation  $R$ , if  $A \implies R$ , then for all executions of  $\mathcal{A}$

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

For the special case of the *fold* operator we present the following corollaries

**Corollary 3.9 (Conservation Law).** If  $\circ$  is an associative and commutative operator, and

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

then

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

*Proof.* Follows from Theorems 3.4 and 3.8.  $\square$

**Corollary 3.10.** If  $\circ$  is an associative, commutative, monotonic operator, and

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

then

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

*Proof.* Follows from Theorems 3.6 and 3.8.  $\square$

Corollary 3.9 is called a conservation law because it gives conditions for conserving *fold* with respect to the start 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 3.10 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. Let  $G$  be an invariant of the system. Let  $Q$  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 3.11.** If the following hold

$$D1. \forall a, s : G(s) \wedge E(s, a) \implies d(s) \geq d(T(s, a))$$

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

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:

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

or

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

$$\Diamond \Box Q$$

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 3.11 is extended to deal with local-global relations:

**Theorem 3.12.** If the following hold

$$H1. \forall a, s : G(s) \wedge e(s_a, a) \implies d(s_a) \geq d(t(s_a, a))$$

$$H2. \exists F \in \mathcal{F} : \forall a \in F : G(s) \wedge \neg Q(s) \implies e(s_a, a) \wedge d(s_a) > d(t(s_a, a))$$

$$H3. d(s) \geq d(s') \text{ and } d(s) > d(s') \text{ are local-global relations}$$

then for all complete executions either E1 or E2 holds.

*Proof.* Follows from the following equalities which follow from the definition of  $e$ , and since  $d(s) \geq d(s')$  and  $d(s) > d(s')$  are local-global relations

$$\begin{aligned} \forall a, s : & (e(s_a, a) = E(s, a)) \wedge \\ & ((d(s_a) \geq d(t(s_a, a)) \implies (d(s) \geq d(T(s, a))) \wedge \\ & ((d(s_a) > d(t(s_a, a)) \implies (d(s) > d(T(s, a))) \end{aligned}$$

$\square$

We use a similar theorem for proving convergence and termination properties (see Theorem 2.3 and Theorem 2.6, respectively). The following corollaries give sufficient conditions for proving progress. We denote by  $Range(d)$  the range of  $d$ .

**Corollary 3.13.** 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.  $\square$

**Corollary 3.14.** If  $\text{Range}(d)$  is a well-founded set, Conditions H1-2 hold, and  $d(s) = \text{fold} \circ s$ , where  $\circ$  is associative, commutative, and strictly monotonic, then eventually  $Q$ .

*Proof.* By assumption on  $\text{fold}$ , Theorem 3.6 holds; and thus, Condition H3 holds. Using Corollary 3.13, eventually-always  $Q$ .  $\square$

## 4. Application of Refinement Techniques

Next we show the application of our theory to proof refinement by looking at two distributed consensus algorithms. The first example focuses on termination, while the second on convergence. They are motivated by problems from control systems [OSFM07, BHOT05].

### 4.1. Distributed Consensus without Failures

We consider consensus problems where the system is required to eventually reach and remain in a state  $s^*$  in which

$$\forall j : s^*[j] = \text{fold} \circ s_0$$

$s_0$  being the initial state of the system. We restrict attention to the case where  $\circ$  is associative, commutative and idempotent. For example, if  $\circ$  is  $\min$  then the system is required to eventually reach and remain in a state where all agent values are the minimum of the initial state. Given a concurrent system,  $\mathcal{A}$ , consisting of  $[N]$  agents ( $N > 1$ ), we would like to prove termination to the desired state. The components of this system are as follows

**States.** The state of agent  $j$  consists of the real-value variable  $s[j]$ .

**Actions.** The set  $A$  is

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

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

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

**Theorem 4.1.** If  $\circ$  is associative, commutative and idempotent,  $\mathcal{A}$  terminates in  $s^*$ .

In order to prove 4.1, we introduce an auxiliary, or “thought,” variable  $x$  that is an array of length  $N$ . Each entry of the array stores a set of indices. This variable will help with the proof of Theorem 4.1, though it plays no other role in the algorithm; it is initialized as follows,

$$\forall j : x[j] = \{j\} \tag{2}$$

We include operations on  $x$  to an action  $(j, k)$  as follows.

$$s[j] := s[j] \circ s[k] \parallel \tag{3}$$

$$x[j] := x[j] \cup x[k] \tag{4}$$

By construction of the action,

$$\square(\forall j : j \in x[j]) \tag{5}$$

Components of correctness follow.

**Lemma 4.2.**  $\square (\forall j : s[j] = fold \circ s^{(0)}[x[j]])$

*Proof.* From Eq. (2), we get that the invariant is true initially. From Eq. (3) we know that  $s'[j] = s[j] \circ s[k]$  for all action  $(j, k)$ . Since the invariant holds for  $s[j], s[k]$ ,

$$\begin{aligned} s'[j] &= (fold \circ s^{(0)}[x[j]]) \circ (fold \circ s^{(0)}[x[k]]) \\ &= fold \circ \left( s^{(0)}[x[j]] \cup s^{(0)}[x[k]] \right) \end{aligned}$$

Since  $\circ$  is idempotent

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

□

**Lemma 4.3.**  $\diamond \square \forall j : (x[j] = [N])$

*Proof.* 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, from the given actions 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  $Q_k$ , as

$$Q_k \equiv \forall j : (k \in x[j]) \tag{6}$$

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

From the definition of  $Q_k$

$$\neg Q_k \equiv (J_k \neq \emptyset). \tag{7}$$

Consider the case where  $\neg Q_k$  holds. For this case, partition the set of agents into  $J_k$ , and its complement. From Equation (5) and (7), both sets are nonempty. Let  $F$  be the set of actions

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

Execution of any action in  $F$  strictly increases  $J$ . From Theorem 3.12 with  $d(s)$  defined to be the size of  $J_k(s)$

$$\diamond \square Q_k$$

Since  $\diamond \square$  is conjunctive

$$(\forall k : \diamond \square Q_k) = \diamond \square (\forall k : Q_k)$$

and

$$(\forall k : Q_k) \implies (\forall j : x[j] = [N])$$

the progress property follows. □

We now have the tools to prove Theorem 4.1

*Proof of Theorem 4.1* The theorem follows from the above safety and progress properties; Lemmas 4.2 and 4.3, respectively. □

## 4.2. Distributed Average without Failures

We discuss a decentralized algorithm for determining the average sensor readings (temperature or pressure readings, for example), described in the control systems literature [OSFM07]: Consider a concurrent system  $\mathcal{A}$  with  $N$  agents ( $N > 1$ ). Each agent  $i \in [N]$  has a variable  $s[i]$ , which stores the reading of sensor  $i$ . We assume that sensor readings are real values. The goal of the system is to compute the average of these measures in a decentralized way.

**States.** The state of each agent  $i$  consists of the variable  $s[i]$ . Initially,  $s[i]$  stores the reading of sensor  $i$ .

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

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

where denoting by  $N_a = |agents(a)|$ ,

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

and

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

This set 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; more formally

$$e(s_a, a) = (MSE(s_a) > 0)$$

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 Figure 3(b), 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  $r \in [0, 1]$ .

**Fairness.** We assume that the system cannot be permanently partitioned into non communicating subsets. This means that eventually an action consisting of agents belonging to complementary sets is executed. 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

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

We assume that the improvement in MSE is lower bounded by some constant  $\Delta$ . We define  $L$  as the unique solution of the equation  $x \cdot (1 - x) = \Delta$  in the interval  $(0, 0.5)$ . Hence, all actions in  $F_J$  have  $r \in [L, 1 - L]$ .

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  $L \cdot (1 - L)$ .

Let  $s^*$  be a vector in  $\mathbb{R}^N$  with  $\forall i s^*[i] = average(s^{(0)})$ . The state  $s^*$  stores the average of the initial values of the agents and  $MSE(s^*) = 0$ . Our proof obligation is to show that the the automaton  $\mathcal{A}$  converges to  $s^*$ .

**Theorem 4.4.**  $\mathcal{A}$  converges to  $s^*$ .

In order to prove the theorem, we define the following relation.

**Definition 10.** For all  $x, x' \in \mathbb{R}^N$

$$R(x, x') \equiv (average(x) = average(x')) \wedge (MSE(x) > MSE(x'))$$

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

We next prove that  $R$  is a local-global relation

**Lemma 4.5.**  $R$  is a local-global relation.

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

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

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

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

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

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

□

Now that we know  $R$  is local-global, we can apply Theorem 3.8 to show that the average of the system is maintained and mean square error is reduced.

**Lemma 4.6.**

$$\left( \forall t : average(s^{(t)}) = average(s^{(0)}) \right) \wedge \left( \forall y, z : y < z \implies MSE(s^{(y)}) > MSE(s^{(z)}) \right)$$

*Proof.* By definition, the set of actions

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

The lemma directly follows from Lemma 4.5 and Theorem 3.8. □

This result states that if the mean square error is positive then it will decrease eventually. We still have to show that that the mean square error will converge to 0. We provide a positive fraction,  $\alpha$  ( $0 \leq \alpha < 1$ ), such that the mean square error decreases by at least  $\alpha$  eventually.

**Lemma 4.7.** There exists  $\alpha$  ( $0 \leq \alpha < 1$ ) such that

$$\forall v : \square(MSE = v \implies \diamond \square MSE \leq v \cdot \alpha)$$

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

$$\exists j : (s[j] - average)^2 \geq MSE \tag{8}$$

with  $MSE = v$ . Without loss of generality, we restrict attention to the case where  $s[j] - average < 0$ .<sup>1</sup> Under this assumption, Eq. (8) becomes

$$-s[j] + average \geq RMSE \quad (9)$$

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

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

$$x[0] \leq s[j] \quad (10)$$

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

$$x[N-1] \geq average \quad (11)$$

From Eq. (9), (10) and (11)

$$x[N-1] - x[0] \geq RMSE \quad (12)$$

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

$$x[N-1] - x[0] = \sum_{k=1}^{N-1} \Delta[k] \quad (13)$$

From Eq. (12) and (13)

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

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

$$\exists i : \Delta[i] \geq \frac{RMSE}{N-1} \quad (14)$$

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. (14)

$$\forall u \in J, v \in \bar{J} : s[u] - s[v] \geq \frac{RMSE}{N-1} \quad (15)$$

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  $\alpha$  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. (15)

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

---

<sup>1</sup> Proofs of the other case are symmetric.

Therefore

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

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

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

where

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

with  $\alpha \in [0, 1)$ .  $\square$

*Proof of Theorem 4.4* It follows from Lemma 4.7 and Corollary 2.5 on convergence since the sequence  $\alpha, \alpha^2, \alpha^3, \dots, \alpha^n$  is decreasing and converging to 0.  $\square$

## 5. Message Passing Systems with Bounded Delay

This section focuses on a subclass of concurrent systems, namely message passing systems, where agents interact by exchanging messages. Messages can be lost or arrive out of order. A key assumption is that messages either get lost or are delivered in finite but unknown time. This class of distributed systems is also known in literature as partially synchronous systems.

We discuss some results on convergence and termination for this class of distributed systems. We also show that algorithms for solving systems of linear equations, such as Gauss, Gauss-Seidel, converges to the correct solution assuming message-passing communication with bounded, but unknown, delay. Finally, we refine these algorithms and derive verified control theory algorithms for robot pattern formations.

### 5.1. Automata formulation

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 Automaton.** 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 it 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 implementation of the input action *receive* is problem specific.

**Communication Channel.** The communication layer is a faulty broadcast channel  $\mathcal{M}$ ; we allow for lost, delayed or out-of-order messages. We make the following assumptions:

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

The state of  $\mathcal{M}$  consists of a bi-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 an 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 partially synchronous automaton  $\mathcal{B}$  is the composition of the  $N$  agent automaton and the broadcast channel automaton.

## 5.2. Stability, Converge and Termination

The definitions of stability, convergence and termination for message-passing systems are analogous to those given in Section 2. These properties can only be shown for untimed systems, because the values of real-time related variables diverge along any admissible execution of the automaton. Given a timed system, the corresponding untimed one is obtained by removing the real-time variables from the states of the system. This section looks at results on convergence and termination for partially synchronous systems.

**Sufficient conditions for coverage.** In [CMP08] (Theorem 2), sufficient conditions for proving convergence of  $\mathcal{B}$  in terms of the convergence proof of the corresponding synchronous system  $\mathcal{A}$  are presented. 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). Our result relies on a well-known result in control theory to prove convergence of synchronous systems (see [Tsi87]), and it is based on establishing the existence of a (Lyapunov) function that is non-increasing along all executions of the system. This function is usually described in term of a collection of level sets of the system state space. Our result needs two extra assumptions. First we restrict the structure of the level sets to *rectangular* level sets. The second assumption pertains to the communication medium: for each communication channel, the receiver eventually receives at least one message from the sender. Put another way, it is not possible that all sent messages get lost. In [CMP08] we do not allow for out-of-order messages.

**Distributed consensus application.** In Theorem 4.1, it was shown that the system reaches consensus under perfect communication. The same theorem works also in presence of partial synchronous communication. In this case, each agent  $j$  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. If we denote by  $\mathcal{B}$  the partially synchronous automaton (with untimed states), the following theorem holds

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

*Proof.* The proof of Theorem 4.1 applies to message passing systems as well since *fold* is assumed to be idempotent.  $\square$

In the next subsection, we extend the result in [CMP08], for a specific class of algorithms.

## 5.3. Solving Systems of Linear Equations

Iterative schemes for solving systems of linear equations, such as Gauss and Gauss-Seidel methods, can be easily adapted to message passing systems. 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 general linear iterative schemes under message-passing systems with bounded, but

unknown delay. Many iterative schemes for pattern formations of robots (for example [OSFM07]) belong to this class. Our results allow us to develop verified algorithms for mobile agents by stepwise refinement. In the end of the subsection we apply our results and derive convergence of a special agent pattern formation algorithm where agents want to form a straight line. Our problem formulation is very general: each agent computes the new value of its variable and *moves* towards it. While moving, it sends messages containing its current location. Our result extends the work of [GM80] in the linear case and the work of [CM69, CMP08] allowing for out-of-order messages.

**Linear Systems.** The system consists of  $N$  agents that communicate via wireless messages. Each agent  $j \in \{0, \dots, N-1\}$ , 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  $\{0, \dots, N-1\}$ . Let  $x^{(*)}$  be the solution to the equations  $A \cdot x = b$ , that is,  $x^* = A^{-1}b$ . The goal of the algorithm is for the states of the individual agents to converge to  $x^*$  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,  $\forall k : \sum_{j \neq k} |A[k, j]| \leq 1$ . We make the additional assumption that:

$$\exists i : \sum_{j \neq i} |A[i, j]| < 1. \quad (16)$$

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. (16). 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. 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. Formally, for all time  $t$ , the number of messages sent by  $j$  in the interval  $(0, t)$  is finite and there exists a time,  $t' > t$ , in which  $j$  sends a message to the channel at time  $t$ .
- 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

$$\begin{aligned} x[j] &:= f_j(z[j], dt) \\ now &:= now + dt \end{aligned}$$

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 its right by at least some constant amount  $\Delta$ . Formally,

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

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  $\mathcal{A}$  converges to  $S^*$  which is the set of all final state of the system. This set consists of all states where

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

**Theorem 5.2.**  $\mathcal{A}$  converges to  $S^*$ .

We first prove some lemmas dealing with safety and progress; then we show that the theorem holds.

**Definition 11.** 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,

$$\forall j : localError[j] = |x^{(*)}[j] - x[j]|$$

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

$$\forall j : destinationError[j] = |x^{(*)}[j] - z[j]|$$

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

$$C[j, k] = \perp \implies receivedError[j, k] = 0$$

$$C[j, k] \neq \perp \implies receivedError[j, k] = |x^{(*)}[k] - C[j, k]|$$

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

$$msgError(m) = |x^{(*)}[m.sender] - m.value|$$

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

$$error[j] = \max \left( localError[j], \right. \\ destinationError[j], \\ \max_{k \neq j} receivedError[k, j], \\ \left. \max_{m \in \mathcal{M} \wedge m.sender = j} msgError(m) \right)$$

- $E$  is the maximum error over all agents

$$E = \max_j error[j]$$

The following inequality relates agent destination and received errors.

**Lemma 5.3.** For all  $j \in [N]$ ,

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

*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,

$$x^{(*)}[j] = b[j] - \sum_{k \neq j} A[j, k] \cdot x^{(*)}[k]$$

The difference of the two above expression is given by

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

Hence,

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

□

Next we find some factor  $\alpha$ ,  $0 \leq \alpha < 1$ , for which the error of the system,  $E$ , decreases (eventually) by  $\alpha$  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 12.** For each agent  $j$ , we define  $p[j]$  as follows

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

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

The  $p$  values satisfy the following lemma:

**Lemma 5.4.**

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

*Proof.* Proof follows by induction on the trees within the forest using the restriction on  $A$  in Eq (16). □

We define the following family of predicates.

**Definition 13.** For any real number  $W$ , and  $j \in [N] \cup \perp$ ,

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

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

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 5.5.** For any real number  $W$ , for any  $j \in [N] \cup \perp$ ,

$$\square (Z_j \implies \circ Z_j)$$

*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  $j \in [N]$ ,  $k \in \text{ancestors}[j]$  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 5.3,

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

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

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

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

$$\text{destinationError}[k] \leq |A[k, \text{parent}(k)]| \cdot \text{receivedError}[k, \text{parent}(k)] + \sum_{l \neq \{k, \text{parent}(k)\}} |A[k, l]| \cdot \text{receivedError}[k, l]$$

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

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

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  $k \notin \text{ancestors}[j]$ . We want to show that  $Z_j$  holds for  $k$  when  $k$  updates its destination  $z[k]$ .

Using Lemma 5.3, the assumption on  $Z_j$  and weakly diagonally dominant assumption on  $A$ , we get

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

□

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

**Lemma 5.6.** For any real number  $W$ , for any  $j \in [N]$ ,

$$\square (Z_{\text{parent}(j)} \implies \diamond Z_j)$$

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

**Base Case.** Let  $j$  be a root in  $\mathbb{F}$ . Predicate  $Z_{\text{parent}(j)} \equiv Z_{\perp}$  holds. From Lemma 5.5, 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  $\text{error}[j] \leq W \cdot p[j]$ .

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

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

Hence, using Lemma 5.3, and the definition of  $p[i]$ , eventually

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

From the fairness condition on agent movement, eventually

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

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

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

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

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

From the all the above equations, eventually

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

**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  $\text{destinationError}[j]$ .

Using Lemma 5.3,

$$\text{destinationError}[j] \leq |A[j, \text{parent}(j)]| \cdot \text{receivedError}[j, \text{parent}(j)] + \sum_{k \neq \{j, \text{parent}(j)\}} |A[j, k]| \cdot \text{receivedError}[j, k]$$

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

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

□

We now find an  $\alpha$  which eventually strictly decreases the error of the system.

**Lemma 5.7.** There exists an  $\alpha$  where  $0 \leq \alpha < 1$ , for all  $W > 0$ ,

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

*Proof.* Let  $\alpha$  be defined as follows,

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

Then from Lemma 5.4,  $\alpha < 1$ .

Consider the family  $\{Z_i\}_{i \in [N] \cup \perp}$ . By assumption,  $E \leq W$ . This means that  $Z_{\perp}$  holds. It holds forever thereafter, by Lemma 5.5. Using Lemma 5.6,  $Z_i$  eventually holds with  $i$  being any root of  $\mathbb{F}$ . Iterating Lemma 5.5 and Lemma 5.6 for all levels of  $\mathbb{F}$ , we have that eventually

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

Hence, eventually

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

Using Lemma 5.5,  $E \leq W \cdot \alpha$  holds forever thereafter. □

*Proof of Theorem 5.2* Follows from Lemma 5.7 and Corollary 2.5 on convergence. □

**Mobile Agent Pattern Formations.** 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], we 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,

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

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

```

fold[D: TYPE, R: TYPE, o: FUNCTION[R, R -> R]]: THEORY
BEGIN
  ASSUMING
    commutative: ASSUMPTION FORALL (x, y: R): x o y = y o x
    associative: ASSUMPTION FORALL (x, y, z: R): (x o y) o z = x o (y o z)
    idempotent: ASSUMPTION FORALL (x: R): x o x = x
  ENDASSUMING

  fold_element: LEMMA
    FORALL (s: myset, K: indices, k: D):
      member(k, K) IMPLIES fold(s, K) = fold(s, K) o s(k)

% ...
END fold

```

Fig. 4. An example *fold* property in PVS. The *fold* definition has been removed for brevity.

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

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

For clarity, we present the matrix  $A$

$$\begin{pmatrix}
 1 & 0 & 0 & 0 & 0 & \dots & 0 \\
 -0.5 & 1 & -0.5 & 0 & 0 & \dots & 0 \\
 0 & -0.5 & 1 & -0.5 & 0 & \dots & 0 \\
 & & & \ddots & & & \\
 0 & \dots & 0 & -0.5 & 1 & -0.5 & 0 \\
 0 & \dots & 0 & 0 & -0.5 & 1 & -0.5 \\
 0 & \dots & 0 & 0 & 0 & 0 & 1
 \end{pmatrix}$$

The solution to these equations has all the  $x[j]$  values equispaced between  $b[0]$  and  $b[N - 1]$ . The algorithm converges to a straight line, from Theorem 5.2. We can design algorithms for other formations in a similar way.

## 6. PVS Discussion

Verifying theorems in an automatic theorem prover, such as PVS, takes more work than proofs checked by a human being because small steps must be verified mechanically. The benefit, however, of using the theorem prover is that all the small steps are, indeed, verified mechanically. One way to help reduce the time required to verify concurrent programs is to develop libraries of theorems that deal with concurrent composition and then reuse theorems and abstract programs from the library. Towards this end we proposed the concept of local-global relations and proved, in PVS, that different state-transition relations have this property. Through parameterized theories and parameterized assumptions, we were able to reuse theorems for different problems by setting parameters appropriately [MB97, dJvdPH00].

To give an idea of the use of PVS in this way, we present the consensus example outlined in Section 4.1. We started with definitions and proofs with a generic binary operator  $\circ$  and a generic function  $g$ ; the domain of  $g$  is states and agents, while its range is some value type. In this abstract theory, we defined the necessary conditions for  $g$  to be local-global. Next, we refined this theory by adding the commutative, associative, and idempotent condition to  $\circ$  and instantiating  $g$  with *fold*. The next refinement was to define transition predicates that we used in abstract programs for reaching consensus. We proved correctness of consensus programs by reusing theorems about safety and programs. The final step was to instantiate the transition theory with concrete operators such as min, max, gcd, and lcm and prove in PVS that these operators are commutative, associative, and idempotent.

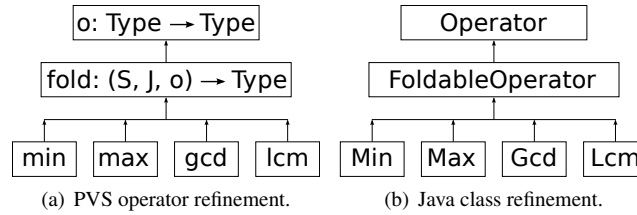


Fig. 5. A comparison of PVS and Java consensus operator refinement. There is a one-to-one correspondence between the abstract refinement in PVS and the more concrete version found in Java.

Implementing the consensus example in PVS required 57 lemmas and about 800 proof steps.<sup>2</sup> Almost 60% of our work went into developing a library of relatively simple lemmas that we used repeatedly. Approximately 15% of work in PVS was devoted to proving that the concrete operators, such as gcd, satisfied the assumptions made of the abstract operator  $o$ . We were able to use existing libraries of PVS theorems, and future work will be reduced as these libraries get richer.

All 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 were carried out in PVS as well. However, the proof of the general problem of solving systems of linear equations using mobile agents, communicating by wireless, has not as yet been proved in PVS.

## 7. Java Implementation

We transformed algorithms from our theorem prover, PVS, into executable programs written in Java and Erlang. For the case of mobile agents, we transformed programs to simulations in Player/Stage. In both cases, an action specified in PVS was transformed into an action implemented in a conventional programming language. In this section we look specifically at our Java implementation.

### 7.1. Distributed Consensus

To provide an example of our theory-to-implementation methodology, we revisit the consensus problem (Section 4.1). Our Java implementation 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. As seen in Figure 5, the refinement correspondence between PVS and Java is one-to-one. The agent class represents a single agent in our system. Each agent 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]). It is also up to the group manager to uphold 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 be 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. Formally proving JRNG satisfies Definition 2 is beyond the scope of our work; we found it to be sufficient in practice.

Verification of our implementation is important to us. There are several examples of automatic transformations from specifications to executable programs in the literature [BPH07, DF06, Sac08, Ken08], however they are inadequate for our particular task; thus we rely on manual methods. Specifically, there are two aspects of our implementation that are verified: that the operators perform as they are expected, and that system state, specifically agent interactions, obey our theory. The former requires verification of very small objects—an advantage of our algorithm refinement

<sup>2</sup> 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.

technique. The implemented objects are small enough that visual code checks suffice. For example, a check that our `min` object in Java is equivalent to the mathematic `min`.

System safety (Corollary 3.9) is more involved. We are required to check that each new state preserves our *fold* guarantee. To this end, we augment our Java code with JML [BCC<sup>+</sup>05] specifications to analyze states within the system. Between agent interactions, a snapshot of the system is taken. JML performs checks of these snapshots as defined in Corollary 3.9. JML has the notion of universal quantification and set theory, making this a very natural exercise for the tool. Moreover, because of this characteristic the translation from our PVS to JML is straightforward—although we perform the actual code translations by hand (from PVS to JML), it is trivial to verify that they are equivalent.

## 8. Conclusions

This paper has focused on the class of constrained optimization algorithms satisfying local-global structures. This property ensures that when groups of agents take steps that change their local states, global properties of system behavior are preserved. Sufficient conditions for proving convergence and termination for this class of algorithms has been provided. Our proof obligation is to show (1) *safety*: that hill-climbing (more accurately valley-descending) steps are local-global, and (2) *progress*: there is a set of actions that is executed infinitely often, where execution of any action in the set reduces a nonnegative Lyapunov function by at least a constant amount. We have provided applications of our theory and proved correctness of distributed consensus algorithms without failure. The applicability of this technique to message passing systems with bounded delay has been investigated, and verified algorithms for mobile agent formation derived. Moreover, these theorems and algorithms were mechanically verified using PVS. In doing so, a reusable library of theorems, specifically for distributed consensus and message passing formation algorithms, was created. Finally, the PVS algorithms have been translated into verified executable Java code.

The class of algorithms investigated in this paper was very general and could be applied to a wider range of applications. Many control theory algorithms may be viewed as constrained optimization algorithms. In this case, each group of agents performs actions that reduce a Lyapunov function subject to the constraint that certain properties are maintained after performing the actions. For example, in the case of consensus to the average, it was shown that each group of agents reduces the mean square error for their respective groups; while, at the same time, the average for the group remains unchanged. This class includes algorithms for message passing systems with bounded delay: systems that operate in continuous state spaces and over continuous time intervals, whereby agents exchange messages. Regardless of message loss, duplication, reordering or delay, algorithm correctness can still be derived. Using our methodology, we have verified a large class of mobile agent pattern formation protocols.

We plan to extend our refinement methodology to a wider class of distributed systems. In particular, we are interested in investigating the applicability of our theory to systems with constraints on the communication media. We also plan to extend our PVS library and explore the applicability of our methodology to other concrete applications. Preliminary work in this direction, including initial results, show that local-global properties fit nicely to other distributed problems, such as graph and sorting algorithms.

## References

- [Abr96] J.-R. Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.
- [ADKM92] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership algorithms for multicast communication groups. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG '92)*, volume 647 of *Lecture Notes in Computer Science*, pages 292–312, Berlin, Heidelberg, November 1992. Springer-Verlag.
- [AHS98] M. Archer, C. Heitmeyer, and S. Sims. 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] M. Archer. TAME: Using PVS strategies for special-purpose theorem proving. *Annals of Mathematics and Artificial Intelligence*, 29(4):139–181, 2000.
- [AW04] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, 2004.
- [BCC<sup>+</sup>05] L. Burdy, Y. Cheon, D.R. Cok, M.D. Ernst, J.R. Kiniry, G.T. Leavens, K.R.M. Leino, and E. Poll. An overview of JML tools and applications. *International Journal on Software Tools Technology Transfer*, 7(3):212–232, 2005.
- [BHOT05] V.D. Blondel, J.M. Hendrickx, A. Olshevsky, and J.N. Tsitsiklis. 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)*, pages 2996–3000, Washington, DC, USA, December 2005. IEEE Computer Society.
- [BKN07] L. Bulwahn, A. Krauss, and T. Nipkow. 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)*, volume 4732 of *Lecture Notes in Computer Science*, pages 38–53, Berlin, Heidelberg, September 2007. Springer-Verlag.

- [BPH07] J. O. Blech and A. Poetzsch-Heffter. A certifying code generation phase. *Electronic Notes in Theoretical Computer Science*, 190(4):65–82, 2007.
- [BS96] R.J.R. Back and K. Sere. Superposition refinement of reactive systems. *Formal Aspects of Computing*, 8(3):324–346, May 1996.
- [CM69] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra and its Applications*, 2(2):199–222, 1969.
- [CM88] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.
- [CMP08] K.M. Chandy, S. Mitra, and C. Pilotto. 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)*, volume 5215 of *Lecture Notes in Computer Science*, pages 217–231. Springer Verlag, September 2008.
- [CS77] S. Chatterjee and E. Seneta. Towards consensus: some convergence theorems on repeated averaging. *Journal of Applied Probability*, 14(1):89–97, 1977.
- [DF06] E. Denney and B. Fischer. 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)*, pages 138–145, Washington, DC, USA, November 2006. IEEE Computer Society.
- [dJvdPH00] E. de Jong, J. van de Pol, and J. Hooman. 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)*, pages 290–298, Washington, DC, USA, April 2000. IEEE Computer Society.
- [Flo67] R. Floyd. Assigning meanings to programs. In *Symposium on Applied Mathematics. Mathematical Aspects of Computer Science*, pages 19–32. American Mathematical Society, 1967.
- [GM80] D. Gabay and H. Moulin. On the uniqueness and stability of nash equilibria in non-cooperative games. In *Applied Stochastic Control of Econometrics and Management Science*, pages 271–293, Amsterdam, 1980. North-Holland.
- [GMPJ09] B. Go, S. Mitra, C. Pilotto, and J. White. Infospheres project. <http://www.infospheres.caltech.edu/facj>, January 2009.
- [Got00] H. Gottlieb. Transcendental functions and continuity checking in pvs. In *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLS '00)*, volume 1869 of *Lecture Notes in Computer Science*, pages 197–214, Berlin, Heidelberg, August 2000. Springer-Verlag.
- [Har98] J. Harrison. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998.
- [HJR04] Q. Huang, C. Julien, and G.-C. Roman. Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(2):192–205, 2004.
- [Hor03] Benjamin Horowitz. *Giotto: a time-triggered language for embedded programming*. PhD thesis, University of California, Berkeley, 2003. Chair-T.A. Henzinger.
- [Jac00] P.B. Jackson. Total-correctness refinement for sequential reactive systems. In *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics (TPHOLS '00)*, volume 1869 of *Lecture Notes in Computer Science*, pages 320–337, Berlin, Heidelberg, August 2000. Springer-Verlag.
- [JS04] A. Jain and R.K. Shyamasundar. Failure detection and membership management in grid environments. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID '04)*, pages 44–52, Washington, DC, USA, November 2004. IEEE Computer Society.
- [Ken08] K. E. Kennedy. *Caps: concurrent automatic programming system*. PhD thesis, Clemson University, Clemson, SC, USA, 2008. Adviser-Dennis Stevenson.
- [Kha04] R.I. Khazan. 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)*, pages 347–356, New York, NY, USA, July 2004. ACM.
- [KLSV06] D.K. Kaynar, N.A. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata (Synthesis Lectures in Computer Science)*. Morgan & Claypool Publishers, 2006.
- [Les] David Lester. NASA langley PVS library for topological spaces. <http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/topology-details.html>.
- [Lib03] D. Liberzon. *Switching in Systems and Control*. Systems and Control: Foundations and Applications. Birkhauser, Boston, June 2003.
- [LKLM05] H. Lim, D. Kaynar, N.A. Lynch, and S. Mitra. 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)*, volume 3829 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, September 2005. Springer-Verlag.
- [LT87] N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 137–151, New York, NY, USA, August 1987. ACM.
- [LT89] N.A. Lynch and M.R. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, September 1989.
- [Lue79] D.G. Luenberger. *Introduction to Dynamic Systems: Theory, Models, and Applications*. John Wiley and Sons, Inc., New York, 1979.
- [Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.
- [MA05] S. Mitra and M. Archer. PVS strategies for proving abstraction properties of automata. *Electronic Notes in Theoretical Computer Science*, 125(2):45–65, 2005.
- [MB97] S. Maharaj and J. Bicarregui. On the verification of vdm specification and refinement with pvs. In *Proceedings of the 12th International Conference on Automated Software Engineering (ASE '97)*, page 280, Washington, DC, USA, November 1997. IEEE Computer Society.
- [MC08] S. Mitra and K.M. Chandy. 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)*, volume 5170 of *Lecture Notes in Computer Science*, pages 230–245, Berlin, Heidelberg, August 2008. Springer-Verlag.
- [Mit07] S. Mitra. *A Verification Framework for Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.
- [Mor87] J. M. Morris. A theoretical basis for stepwise refinement and the programming calculus. *Science of Computer Programming*, 9(3):287–306, 1987.
- [ORS92] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *Proceedings of 11th*

- International Conference on Automated Deduction (CADE '92)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.
- [OS07] R. Olfati-Saber. Distributed kalman filtering for sensor networks. In *Proceedings of the 46th IEEE Conference on Decision (CDC '07)*, pages 5492–5498, Washington, DC, USA, December 2007. IEEE Computer Society.
- [OSFM07] R. Olfati-Saber, J.A. Fax, and R.M. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, January 2007.
- [Rei96] M. K. Reiter. A secure group membership protocol. *IEEE Transactions on Software Engineering*, 22(1):31–42, 1996.
- [RP96] E. Rohwedder and F. Pfenning. Mode and termination checking for higher-order logic programs. In *Proceedings of the 6th European Symposium on Programming Languages and Systems (ESOP '96)*, volume 1058 of *Lecture Notes in Computer Science*, pages 296–310, Berlin, Heidelberg, April 1996. Springer-Verlag.
- [Sac08] K. Sacha. Model-based implementation of real-time systems. In *Proceedings of the 27th International Conference on Computer Safety, Reliability, and Security (SAFECOMP '08)*, volume 5219 of *Lecture Notes in Computer Science*, pages 332–345. Springer-Verlag, September 2008.
- [Spi07] M. Spichkova. *Specification and Seamless Verification of Embedded Real-Time Systems: FOCUS on Isabelle*. PhD thesis, Technische Universität München, 2007.
- [Spi08] M. Spichkova. Refinement-based verification of interactive real-time systems. *Electronic Notes in Theoretical Computer Science*, 214:131–157, 2008.
- [Tau05] J.A. Tauber. *Verifiable Compilation of I/O Automata Without Global Synchronization*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005.
- [Tsi87] J.N. Tsitsiklis. On the stability of asynchronous iterative processes. *Theory of Computing Systems*, 20(1):137–153, December 1987.
- [UL07] S. Umeno and N.A. Lynch. 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)*, volume 4416 of *Lecture Notes in Computer Science*, pages 557–572. Springer-Verlag, April 2007.
- [Wir71] N. Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14(4):221–227, 1971.