Distrib. Comput. (1997) 10: 137–148

DISTRIBUTED COMPUTING

Keeping track of the latest gossip in a distributed system*

Madhavan Mukund[1], **Milind Sohoni**[2]

[1] SPIC Mathematical Institute, 92 G.N. Chetty Road, Madras 600 017, India (e-mail: madhavan@ssf.ernet.in)
[2] Department of Computer Science and Engineering, Indian Institute of Technology, Bombay 400 076, India (e-mail: sohoni@cse.iitb.ernet.in)

Received: December 1994 / Accepted: July 1996

Summary. We tackle a natural problem from distributed computing, involving time-stamps. Let $\mathscr{P} = \{p_1, p_2, \ldots, p_N\}$ be a set of computing agents or processes which synchronize with each other from time to time and exchange information about themselves and others. The *gossip problem* is the following: Whenever a set $P \subseteq \mathscr{P}$ meets, the processes in P must decide *amongst themselves* which of them has the latest information, direct or indirect, about each agent p in the system. We propose an algorithm to solve this problem which is finite-state and local. Formally, this means that our algorithm can be implemented as an asynchronous automaton.

Key words: Distributed algorithms – Synchronous communication – Bounded time-stamps – Asynchronous automata

Introduction

The aim of this paper is to tackle a natural problem from distributed computing, involving time-stamps. Let $\mathscr{P} = \{p_1, p_2, \ldots, p_N\}$ be a set of computing agents or processes which synchronize with each other from time to time and exchange information about themselves and others. The *gossip problem* is the following: Whenever a set $P \subseteq \mathscr{P}$ meets, the processes in P must decide *amongst themselves* which of them has the latest information, direct or indirect, about each agent p in the system.

This is easily accomplished if the agents decide to "time-stamp" every synchronization and pass these time-stamps along with each exchange of information. This does not require that all their clocks be synchronized. For example, each process can use an independent counter. When a set $P \subseteq \mathscr{P}$ meets, the processes in P jointly agree on a new value for their counters which exceeds the maximum of the counter values currently held by them. Thus, for any process p, the time-stamps assigned to synchronization events involving p form a strictly increasing sequence (albeit with gaps between successive time-stamps). So, the problem of deciding who has the latest information about p reduces to that of checking for the largest time-stamp.

This scheme has the following drawback: as the computation progresses these counter values increase without bound and most of the agents' time would be taken up in passing on large numbers, as opposed to actual gossip.

We propose an algorithm using counters which take on values from a bounded, finite set. We assign an independent counter to each subset of processes which can potentially synchronize. These counters are updated when the corresponding sets of processes meet. The update is performed jointly by the processes which meet.

Since our set of counter values is bounded, time-stamps have to be reused and, in general, different synchronizations involving a particular set of processes will acquire the same time-stamp during a computation. Despite this, our algorithm guarantees that whenever a set $P \subseteq \mathscr{P}$ meets, the processes in P can decide correctly which of them has the best information about any other agent p in the system. Thus, in essence, the processes in $\mathscr{P}$ may be finite state machines and yet manage to keep track of the latest information about other agents. Further, the algorithm itself does not induce any additional communications.

We formalize the gossip problem and our solution to it in terms of asynchronous automata. These machines were first introduced by Zielonka and are a natural generalization of finite-state automata for modelling concurrent systems [30]. An asynchronous automaton consists of a set of finite-state agents which synchronize to process their input. Each letter a in the input alphabet Σ is assigned a subset $\theta(a)$ of processes which jointly update their state when reading a. The processes outside $\theta(a)$ remain unchanged during this move – in fact, they are oblivious to the occurrence of a.

(Calling these automata *asynchronous* is misleading. The automata communicate *synchronously*. Zielonka used the term "asynchronous" to emphasize that different components of the network can proceed independently while

*A preliminary version of this paper appeared in [22]. The first author was partially supported by IFCPAR Project 502-1. This work was done while the second author was with the School of Mathematics, SPIC Science Foundation, now known as the SPIC Mathematical Institute

Correspondence to: M. Mukund

processing the input. We continue to use this rather inappropriate terminology for historical reasons.)

A preview of the algorithm

Our algorithm proceeds by having each process maintain different levels of information about the rest of the system. At the *primary* level, a process p records the latest information that it has heard from every other process. Thus, the primary information of process p regarding process q corresponds to the most recent event performed by q which p is aware of, either directly or indirectly.

The *secondary* information of a process consists of its knowledge about the primary information of other processes. Thus, for processes p, q and r, p's secondary information would refer to events of the form "the latest that p knows about what q knows about r". Similarly, the *tertiary information* of a process consists of its knowledge about the secondary information of every other process.

Events are identified by labels assigned to them when they occur – the label of each event is assigned jointly by all the processes that take part in the event. (An event involving only one process corresponds to an internal event.) These labels are best thought of as time-stamps. The goal of the algorithm is to correctly compare and update the primary information of all processes which participate in each synchronization, with the constraint that only a bounded number of labels are used to time-stamp events, regardless of the length of the overall computation.

Let p and q be processes that synchronize. We prove that p's primary information about r is more recent (and therefore better) than q's primary information about r if and only if the event corresponding to r recorded in q's primary information is also present in p's secondary information. In other words, the primary information of p and q can be compared by just checking for *equality* of labels within their primary and secondary information – we do not have to maintain any order between the labels. This feature is crucial for designing an algorithm which uses only a bounded number of time-stamps: reusing a label will, in general, destroy any a priori order on the set of labels.

It then follows that we can reuse time-stamps provided we maintain the following invariant across the system: at any stage of the computation, if the same label is present in the primary or secondary information of two different processes, then the labels actually point to the same event. In other words, two *different* instances of the same time-stamp should never simultaneously be present in the primary and secondary information of the system. This invariant is difficult to maintain because the processes which take part in an event and assign a time-stamp to it will not, in general, have access to the primary and secondary information of all processes across the system. So, they have no way of knowing precisely which labels are "in use" across the system when the event occurs.

It turns out that tertiary information can be used to resolve the problem of deciding when a time-stamp can be reused. We prove that a time-stamp previously assigned by a process p is currently "in use" – i.e., it currently belongs to the primary or secondary information of some other process q – only if it also belongs to the tertiary information of p. From this, it follows that labels which are not in the tertiary information of a process can be reused. Since the number of events in the tertiary information of each process is bounded, processes in the system can always work with a bounded set of labels large enough to permit each event to be assigned a fresh time-stamp which does not clash with the set of time-stamps currently "in use" across the system.

As we remarked earlier, our algorithm can be described as an asynchronous automaton, where each process is locally a finite-state machine. The automaton that we construct to solve the gossip problem can be effectively presented in space polynomial in the number of processes in the system. This means that the automaton can be embedded in other distributed algorithms without sacrificing efficiency.

In this paper, we solve the gossip problem for systems with multi-party synchronization. Surprisingly, the problem appears *no easier* when restricted to systems with pariwise synchronization. Though systems with pairwise synchronization are perhaps more representative of "real world" systems, we have chosen to describe our solution in the general setting because it has important applications in theoretical studies of distributed systems based on the asynchronous automaton model [13, 14, 23].

The technique we describe for analyzing synchronizing systems is also applicable to more general classes of distributed systems. For instance, we can adapt our algorithm for maintaining primary, secondary and tertiary information to solve the gossip problem for "well behaved" classes of message-passing systems [21].

The paper is organized as follows. In the next section, we introduce asynchronous automata and formalize the gossip problem in terms of these automata. To do this, we define a natural partial order on events in the system. In Sect. 2 we introduce ideals and frontiers, both of which play a crucial role in the rest of the paper. Sections 3 and 4 describe how to maintain, compare and update in a local manner the latest information about other processes. The next section puts all these ideas together and formally describes the "gossip automaton" which solves the problem we set out to tackle.

Section 6 briefly examines extensions of the basic gossip automaton and possibilities for optimizing the construction. We also look at applications of the gossip automaton in logic and the theory of asynchronous automata [13, 14, 23, 27].

In the concluding Discussion, we place our results in perspective. We discuss similarities and differences with other work on "gossiping" and bounded time-stamps [1, 2, 3, 5, 8, 9, 12].

1 Preliminaries

Let $\mathscr{P}$ be a finite set of processes which synchronize periodically and let the set of possible synchronizations permitted in the system be denoted $\mathscr{C}$, where $\mathscr{C} \subseteq (2^{\mathscr{P}} - \{\emptyset\})$. So, each element $c \in \mathscr{C}$ is a non-empty subset of $\mathscr{P}$. When c occurs, the processes in c share all information about their local states and update their states in synchrony.

We model a computation of the system as a sequence of communications – that is, a word $u \in \mathscr{C}^*$. Let u be of length m. It is convenient to think of u as a function $u: [1..m] \to \mathscr{C}$, where for natural numbers i and j, $[i..j]$ abbreviates the set $\{i, i+1, \ldots, j\}$ if $i \leqq j$ and $[i..j] = \emptyset$ otherwise. By this convention, the empty word ε is denoted by the unique function $\emptyset \to \mathscr{C}$.

Events. With $u: [1..m] \to \mathscr{C}$, we associate a set of *events* $\mathscr{E}_u$. Each event e is of the form $(i, u(i))$, where $i \in [1..m]$. In addition, it is convenient to include an *initial event* denoted 0. Thus, $\mathscr{E}_u = \{0\} \cup \{(i, u(i)) \mid i \in [1..m]\}$.

The initial event marks an implicit synchronization of all the processes before the start of the actual computation. So, if u is the empty word ε, $\mathscr{E}_u = \{0\}$.

Usually, we will write $\mathscr{E}$ for $\mathscr{E}_u$. For $p \in \mathscr{P}$ and $e \in \mathscr{E}$, we write $p \in e$ to denote that $p \in u(i)$ when $e = (i, u(i))$. Thus, $p \in e$ implies that process p participated in the synchronization e. For the initial event 0, we define $p \in 0$ to hold for all $p \in \mathscr{P}$. If $p \in e$, then we say that e is a *p-event*.

Ordering relations on $\mathscr{E}$. The word u imposes a total order on events in $\mathscr{E}$: define $e < f$ if $e \neq f$ and either $e = 0$ or $e = (i, u(i))$, $f = (j, u(j))$, and $i < j$. We write $e \leqq f$ if $e = f$ or $e < f$.

However, the temporal order $<$ does not accurately reflect the cause and effect relationship between events in $\mathscr{E}$. Clearly, synchronizations between disjoint sets of processes can be performed independently. In particular, if two such synchronizations occur consecutively in u, they could also be transposed without affecting the outcome of the computation. To record information about causality and independence, we define a partial order $\sqsubseteq^*$ on $\mathscr{E}$.

To begin with, we observe that each process p orders the events in which it participates: define $\lhd_p$ to be the relation

$$e \lhd_p f \triangleq e < f, p \in e \cap f \text{ and for all } e < g < f,\ p \notin g.$$

The set of all p-events in $\mathscr{E}$ is totally ordered by $\lhd_p^*$, the reflexive, transitive closure of $\lhd_p$.

Define $e \sqsubset f$ if for some p, $e \lhd_p f$ and $e \sqsubseteq f$ if $e = f$ or $e \sqsubset f$. Let $\sqsubseteq^*$ denote the transitive closure of $\sqsubseteq$. If $e \sqsubseteq^* f$ then we say that e is *below* f.

It is not difficult to see that the *causality relation* $\sqsubseteq^*$ accurately models the cause and effect relationship between events in $\mathscr{E}$. In particular, all rearrangements of the letters in u which arise out of permuting adjacent independent synchronizations will give rise to isomorphic structures $(\mathscr{E}, \sqsubseteq^*)$.

Example. Let $\mathscr{P} = \{p, q, r, s\}$ and $\mathscr{C} = \{a, b, c\}$ where $a = \{p, q\}$, $b = \{r, s\}$ and $c = \{q, r, s\}$. Figure 1 shows the events $\mathscr{E}$ corresponding to the word *bacabba*. The dashed box corresponds to the "mythical" event 0, which we insert at the beginning for convenience.

In the figure, the arrows between the events denote the relations $\lhd_p$, $\lhd_q$, $\lhd_r$ and $\lhd_s$. From these, we can compute $\sqsubset$ and $\sqsubseteq^*$. Thus, for example, we have $e_1 \sqsubseteq^* e_4$ since $e_1 \lhd_r e_3 \lhd_q e_4$.

Note that 0 is below every event. Also, for each $p \in \mathscr{P}$, the set of all p-events in $\mathscr{E}$ is totally ordered by $\sqsubseteq^*$ since $\lhd_p^*$ is contained in $\sqsubseteq^*$.

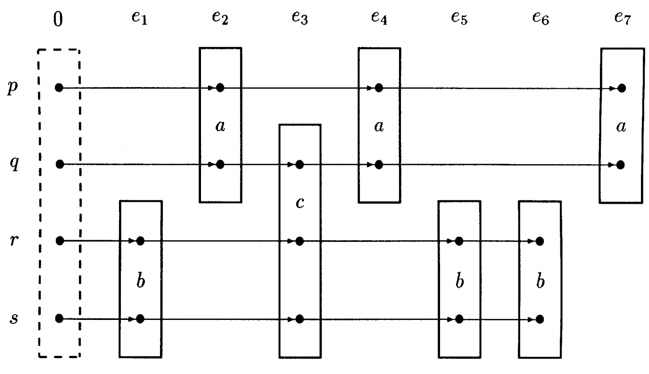

Fig. 1. A typical computation of a system of synchronizing processes

The set of events below e is denoted $e\downarrow$. These represent the only synchronizations in $\mathscr{E}$ which are "known" to the processes in e when e occurs.

Latest information. Let $\mathscr{E}$ be the set of events of the communication sequence $u: [1..m] \to \mathscr{C}$. The $\sqsubseteq^*$-maximum p-event in $\mathscr{E}$ is denoted $max_p(\mathscr{E})$. $max_p(\mathscr{E})$ is the last event in $\mathscr{E}$ in which p has taken part. Since $p \in 0 \in \mathscr{E}$ and all p-events are totally ordered by $\sqsubseteq^*$, $max_p(\mathscr{E})$ is well-defined.

Let $p, q \in \mathscr{P}$. The latest information p has about q in $\mathscr{E}$ corresponds to the $\sqsubseteq^*$-maximum q-event in the subset of events $max_p(\mathscr{E})\downarrow$. We denote this event by $latest_{p \to q}(\mathscr{E})$. Since all q-events are totally ordered by $\sqsubseteq^*$ and $q \in 0 \sqsubseteq^* max_p(\mathscr{E})$, $latest_{p \to q}(\mathscr{E})$ is well-defined.

Example. Continuing with our example, in Fig. 1, $max_p(\mathscr{E}) = e_7$ whereas $max_s(\mathscr{E}) = e_6$. $latest_{p \to q}(\mathscr{E}) = e_7$, but $latest_{p \to s}(\mathscr{E}) = e_3$. On the other hand, $latest_{s \to p}(\mathscr{E}) = e_2$.

For any processes $p, p', q \in \mathscr{P}$, the events $latest_{p \to q}(\mathscr{E})$ and $latest_{p' \to q}(\mathscr{E})$ are both q-events and are thus always comparable with respect to $\sqsubseteq^*$. Our goal is to design a scheme whereby each process p maintains a bounded amount of information locally, so that whenever a set $c \subseteq \mathscr{P}$ synchronizes, the processes in c can decide amongst themselves which of them has heard most recently from every process in the system. More formally, for every $q \in \mathscr{P}$, all the processes in c should be able to jointly compute which of the events $\{latest_{p \to q}(\mathscr{E})\}_{p \in c}$ is maximum with respect to $\sqsubseteq^*$.

We make precise the notions of *bounded* and *local* information using asynchronous automata.

Asynchronous automata

Distributed alphabet. Let $\mathscr{P}$ be a finite set of processes as before. A *distributed alphabet* is a pair (Σ, θ) where Σ is a finite set of *actions* and $\theta: \Sigma \to (2^{\mathscr{P}} - \{\emptyset\})$ assigns a non-empty set of processes to each $a \in \Sigma$.

State spaces. With each process p, we associate a finite set of states denoted S_p. Each state in S_p is called a *local state*. For $P \subseteq \mathscr{P}$, we use S_P to denote the product $\prod_{p \in P} S_p$. An element $\vec{s}$ of S_P is called a *P-state*. A $\mathscr{P}$-state is also called a *global state*. Given $\vec{s} \in S_P$, and $P' \subseteq P$, we use $\vec{s}_{P'}$ to denote the projection of $\vec{s}$ onto $S_{P'}$.

Asynchronous automaton. An *asynchronous automaton* $\mathscr{A}$ over (Σ, θ) is of the form $(\{S_p\}_{p \in \mathscr{P}}, \{\rightarrow_a\}_{a \in \Sigma}, \mathscr{S}_0, \mathscr{S}_F)$, where $\rightarrow_a \subseteq S_{\theta(a)} \times S_{\theta(a)}$ is the *local transition relation* for a, and $\mathscr{S}_0, \mathscr{S}_F \subseteq S_{\mathscr{P}}$ are sets of *initial* and *final* global states. Intuitively, each local transition relation $\rightarrow_a$ specifies how the processes $\theta(a)$ that meet on a may decide on a joint move. Processes outside $\theta(a)$ do not change their state when a occurs. Thus we define the *global transition relation* $\Rightarrow \subseteq S_{\mathscr{P}} \times \Sigma \times S_{\mathscr{P}}$ by $\vec{s} \overset{a}{\Rightarrow} \vec{s}'$ if $\vec{s}_{\theta(a)} \rightarrow_a \vec{s}'_{\theta(a)}$ and $\vec{s}_{\mathscr{P}-\theta(a)} = \vec{s}'_{\mathscr{P}-\theta(a)}$.

$\mathscr{A}$ is called *deterministic* if the global transition relation $\Rightarrow$ of $\mathscr{A}$ is a function from $S_{\mathscr{P}} \times \Sigma$ to $S_{\mathscr{P}}$ and the set of initial states $\mathscr{S}_0$ is a singleton. Notice that $\Rightarrow$ is a function iff each local transition relation $\rightarrow_a$, $a \in \Sigma$, is a function from $S_{\theta(a)}$ to $S_{\theta(a)}$. We shall only deal with deterministic asynchronous automata in this paper.

Runs. Given a word $u:[1..m] \rightarrow \Sigma$, a *run* of $\mathscr{A}$ on u is a function $\rho:[0..m] \rightarrow S_{\mathscr{P}}$ such that $\rho(0) \in \mathscr{S}_0$ and for $i \in [1..m]$, $\rho(i-1) \overset{u(i)}{\Rightarrow} \rho(i)$. If $\mathscr{A}$ is deterministic, each word u gives rise to a unique run which we denote ρ_u.

The word u is *accepted* by $\mathscr{A}$ if there is a run ρ of $\mathscr{A}$ on u such that $\rho(m) \in \mathscr{S}_F$. $L(\mathscr{A})$, the language recognized by $\mathscr{A}$, is the set of words accepted by $\mathscr{A}$. In this paper, we will not look at asynchronous automata as language recognizers. Instead, we shall treat them as devices for locally computing families of functions.

Locally computable functions. Let *Val* be a set (of *values*). A Σ-indexed family of functions is a set $\mathscr{F}_\Sigma = \{f_a : \Sigma^* \rightarrow Val\}_{a \in \Sigma}$. So, $\mathscr{F}_\Sigma$ contains a function f_a for each letter $a \in \Sigma$.

$\mathscr{F}_\Sigma$ is *locally computable* if we can find a deterministic asynchronous automaton $\mathscr{A} = (\{S_p\}_{p \in \mathscr{P}}, \{\rightarrow_a\}_{a \in \Sigma}, \mathscr{S}_0, \mathscr{S}_F)$ and a family of *local functions* $\mathscr{G}_\Sigma = \{g_a : S_{\theta(a)} \rightarrow Val\}_{a \in \Sigma}$, such that for each word $u:[1..m] \rightarrow \Sigma$, $f_a(u) = g_a(\vec{s}_{\theta(a)})$, where $\vec{s} = \rho_u(m)$ and ρ_u is the unique run of $\mathscr{A}$ over u.

In other words, the processes in $\theta(a)$ can locally compute the value $f_a(u)$ for any $u \in \Sigma^*$ by applying the function g_a to the (unique) $\theta(a)$-state reached by the automaton after reading u.

Our problem involving the latest information of processes in $\mathscr{P}$ can now be formalized in terms of asynchronous automata.

Given $\mathscr{C} \subseteq (2^{\mathscr{P}} - \{\emptyset\})$, let $\Sigma = \{\hat{c}\}_{c \in \mathscr{C}}$. The distribution function θ is defined in the obvious way – for each $\hat{c} \in \Sigma$, $\theta(\hat{c}) = c$. For convenience, henceforth we shall drop the distinction between a subset $c \in \mathscr{C}$ and the corresponding letter $\hat{c} \in \Sigma$ and refer to both as just c. Thus, we will use S_c to denote the set of $\theta(\hat{c})$-states and $\rightarrow_c$ to denote the local transition function for $\hat{c}$.

Let $u:[1..m] \rightarrow \Sigma$ be a communication sequence and $c \subseteq \mathscr{P}$. For each $q \in \mathscr{P}$, we denote by $best_c(u,q)$ the set of processes in c which have the most recent information about q at the end of u – i.e.,

$$best_c(u,q) = \{p \in c \mid \forall p' \in c.\ latest_{p' \rightarrow q}(\mathscr{E}_u) \sqsubseteq^* latest_{p \rightarrow q}(\mathscr{E}_u)\}.$$

Let $Val = (2^{\mathscr{P}} - \{\emptyset\})^{\mathscr{P}}$. So, each member of *Val* is a function from $\mathscr{P}$ to non-empty subsets of $\mathscr{P}$. Our goal is to show that the family of functions $\{latest\text{-}gossip_c : \Sigma^* \rightarrow Val\}_{c \in \Sigma}$ is locally computable, where:

$\forall u \in \Sigma^*. \forall c \in \Sigma.$ $latest\text{-}gossip_c(u)$ is the function

$$\{p \mapsto best_c(u,p)\}_{p \in \mathscr{P}}.$$

2 Ideals and frontiers

For the moment, let us fix a communication sequence $u:[1..m] \rightarrow \Sigma$ and the corresponding set of events $\mathscr{E}$.

The main source of difficulty in solving the gossip problem is the fact that the processes in $\mathscr{P}$ need to compute global information about the communication sequence u while each process only has access to a local, "partial" view of u. Although partial views of u correspond to subsets of $\mathscr{E}$, not every subset of $\mathscr{E}$ arises from such a partial view. Those subsets of $\mathscr{E}$ which do correspond to partial views of u are called ideals.

Ideals. A set of events $I \subseteq \mathscr{E}$ is called an *order ideal* if I is closed with respect to $\sqsubseteq^*$ – i.e., $e \in I$ and $f \sqsubseteq^* e$ implies $f \in I$ as well. We shall always refer to order ideals as just *ideals*.[1]

The requirement that an ideal be closed with respect to $\sqsubseteq^*$ guarantees that the observation it represents is "consistent" – whenever an event e has been observed, so have all the events in the computation which necessarily precede e.

The minimum possible partial view of a word u is the ideal $\{0\}$. This is because of our interpretation of 0 as an event which takes place *before* the actual computation begins. Since 0 lies below every event in $\mathscr{E}$, $0 \in I$ for every non-empty ideal I. We shall assume that every ideal we consider is non-empty.

Clearly the entire set $\mathscr{E}$ is an ideal, as is $e\downarrow$ for any $e \in \mathscr{E}$. It is easy to see that if I and J are ideals, so are $I \cup J$ and $I \cap J$.

Example. Let us look once again at Fig. 1. $\{0, e_2\}$ is an ideal, but $\{0, e_2, e_3\}$ is not, since $e_1 \sqsubseteq^* e_3$ but $e_1 \notin \{0, e_2, e_3\}$. $\{0, e_1, e_2, e_3, e_5\}$ is the ideal $e_5\downarrow$, whereas $\{0, e_1, e_2, e_3, e_4, e_5\}$ is an ideal which is not of the form $e\downarrow$ for any $e \in \mathscr{E}$.

We need to generalize the notion of $max_p(\mathscr{E})$, the maximum *p-event in* $\mathscr{E}$, to all ideals $I \subseteq \mathscr{E}$.

P-views. For an ideal I, the $\sqsubseteq^*$-maximum p-event in I is denoted $max_p(I)$. The *p-view of I* is the set $I|_p = max_p(I)\downarrow$. So, $I|_p$ is the set of all events in I which p can "see". For $P \subseteq \mathscr{P}$, the P-view of I, denoted $I|_P$, is $\bigcup_{p \in P} I|_p$, which is also an ideal. In particular, we have $I|_{\mathscr{P}} = I$.

Example. In Fig. 1, let I denote the ideal $\{0, e_1, e_2, e_3, e_4, e_5, e_6\}$. $max_q(I) = e_4$ and hence $I|_q = \{0, e_1, e_2, e_3, e_4\}$. On the other hand, though $max_r(I) = e_6$, $I|_r \neq I$; $I|_r = I - \{e_4\}$. The joint view $I|_{\{q,r\}} = I = I|_{\mathscr{P}}$.

[1] In the theory of partial orders, *order ideals* and *ideals* are distinct concepts. Ideals are normally assumed to be subsets which are $\sqsubseteq^*$-closed *and* directed. We shall, however, deal only with order ideals in this paper and so our terminology should cause no confusion

For an ideal I, the views $I|_p$ and $I|_q$ seen by two processes $p,q \in \mathscr{P}$ are, in general, incomparable. The events in I where these two views begin to diverge – the *frontier* of $I|_p \cap I|_q$ – play a crucial role in our analysis.

Frontiers. Let I be an ideal and $p,q,r \in \mathscr{P}$. We say that an event e is an *r-sentry* for p with respect to q if $e \in I|_p \cap I|_q$ and $e \lhd_r f$ for some $f \in I|_q - I|_p$. Thus e is an event known to both p and q whose r-successor is known only to q. Notice that there need not always be an r-sentry for p with respect to q.

The pq-frontier at I, $frontier_{pq}(I)$ is defined as follows:

$$frontier_{pq}(I)$$

$$= \{e \in I \mid \exists r \in \mathscr{P}.\, e \text{ is an } r\text{-sentry for } p \text{ with respect to } q\}$$

Observe that this definition is asymmetric – in general, $frontier_{pq}(I) \neq frontier_{qp}(I)$.

Example. As before, in Fig. 1, let I denote the ideal $\{0, e_1, e_2, e_3, e_4, e_5, e_6\}$. $I|_q \cap I|_r = \{0, e_1, e_2, e_3\}$. $frontier_{rq}(I) = \{e_2, e_3\}$ – e_2 is a p-sentry for r with respect to q whereas e_3 is a q-sentry. On the other hand, $frontier_{qr}(I) = \{e_3\}$. e_3 is both an r-sentry as well as an s-sentry for q with respect to r.

As the example demonstrates, an event $e \in frontier_{pq}(I)$ could simultaneously be an r-sentry for p for several different processes r. However, it is not difficult to show that for any process r, there is at most one r-sentry for p with respect to q.

3 Primary and secondary information

For a word u and processes $p,q \in \mathscr{P}$, we have already defined $latest_{p\to q}(\mathscr{E})$, the latest information that p has about q after u. We now extend this definition to arbitrary ideals.

Primary information. Let I be an ideal and $p,q \in \mathscr{P}$. Then $latest_{p\to q}(I)$ denotes the $\sqsubseteq^*$-maximum q-event in $I|_p$. So, $latest_{p\to q}(I)$ is the latest q-event in I that p knows about.

The *primary information* of p after I, $primary_p(I)$, is the set $\{latest_{p\to q}(I)\}_{q\in\mathscr{P}}$. More precisely, $primary_p(I)$ is an *indexed* set of events – each event $e = latest_{p\to q}(I)$ in $primary_p(I)$ is represented as a triple (p,q,e). As usual, for $P \subseteq \mathscr{P}$, $primary_P(I) = \bigcup_{p\in P} primary_p(I)$.

As we have already remarked, for all $q \in \mathscr{P}$, the set of q-events in $I|_p$ is always nonempty, since $q \in 0 \in I|_p$. Further, since all q-events are totally ordered by $\lhd_q^*$ and hence by $\sqsubseteq^*$, the maximum q-event in $I|_p$ is well-defined. Notice that $latest_{p\to p}(I) = max_p(I)$.

To compare primary events, processes need to maintain additional information. It turns out that it is sufficient for each process to keep track of all the other processes' primary information.

Secondary information. The *secondary information* of p after I, $secondary_p(I)$, is the (indexed) set $\bigcup_{q\in\mathscr{P}} primary_q(latest_{p\to q}(I)\downarrow)$. In other words, this is the latest information that p has in I about the primary information of q, for each $q \in \mathscr{P}$. Once again, for $P \subseteq \mathscr{P}$, $secondary_P(I) = \bigcup_{p\in P} secondary_p(I)$.

Each event in $secondary_p(I)$ is of the form $latest_{q\to r}(latest_{p\to q}(I)\downarrow)$ for some $q,r \in \mathscr{P}$. This is the latest r-event which q knows about upto the event $latest_{p\to q}(I)$. We abbreviate $latest_{q\to r}(latest_{p\to q}(I)\downarrow)$ by $latest_{p\to q\to r}(I)$.

Just as we represented events in $primary_p(I)$ as triples of the form (p,q,e), where $p,q \in \mathscr{P}$ and $e \in I$, we represent each secondary event $e = latest_{p\to q\to r}(I)$ in $secondary_p(I)$ as a quadruple (p,q,r,e).

However, we will often ignore the fact that $primary_p(I)$ and $secondary_p(I)$ are *indexed* sets of events and treat them, for convenience, as just sets of events. Thus, for an event $e \in I$, we shall write $e \in primary_p(I)$ to mean that there exists a process $q \in \mathscr{P}$ such that $(p,q,e) \in primary_p(I)$ – i.e., $e = latest_{p\to q}(I)$. Similarly, $e \in secondary_p(I)$ will indicate that for some $q,r \in \mathscr{P}$, $(p,q,r,e) \in secondary_p(I)$. We extend this to other set-theoretic operations as well. So, for instance, if we say $e \in primary_p(I) \cap secondary_q(I)$, we mean that we can find $p',q',q'' \in \mathscr{P}$ such that $(p,p',e) \in primary_p(I)$ and $(q,q',q'',e) \in secondary_q(I)$.

Notice that each primary event $latest_{p\to q}(I)$ is also a secondary event $latest_{p\to p\to q}(I)$ (or, equivalently, $latest_{p\to q\to q}(I)$). So, following our convention that $primary_p(I)$ and $secondary_p(I)$ be treated as sets of events, we write $primary_p(I) \subseteq secondary_p(I)$.

Comparing primary information

Our goal is to compare and update the primary information of processes whenever they meet. For this, we need the following observation regarding the significance of events lying on frontiers.

Lemma 1. *Let I be an ideal, $p,q \in \mathscr{P}$ and $e \in frontier_{pq}(I)$ an r-sentry for p with respect to q. Then $e = latest_{p\to r}(I)$. Also, for some $r' \in \mathscr{P}$, $e = latest_{q\to r'\to r}(I)$. So, $e \in primary_p(I) \cap secondary_q(I)$.*

Proof. Since e is an r-sentry, for some $f \in I|_q - I|_p$, $e \lhd_r f$. Suppose that $latest_{p\to r}(I) = e' \neq e$. Since all r-events are totally ordered by $\lhd_r^*$, we must have $e \lhd_r^+ e'$ (where $\lhd_r^+$ is the transitive closure of the irreflexive relation $\lhd_r$). However, $e \lhd_r f$ as well, so we have $e \lhd_r f \lhd_r^* e'$. Since $e' \in I|_p$, this means that $f \in I|_p$ as well, which is a contradiction.

Next, we must show that $e = latest_{p\to r'\to r}(I)$ for some $r' \in \mathscr{P}$. We know that there is a path $e \sqsubset f_1 \sqsubset \cdots \sqsubset max_p(I)$, since $e \in I|_p$. This path starts inside $I|_p \cap I|_q$.

If this path never leaves $I|_p \cap I|_q$ then $max_p(I) \in I|_q$. Since $max_p(I)$ is the $\sqsubseteq^*$-maximum p-event in I, it must be the $\sqsubseteq^*$-maximum p-event in $I|_q$. So, $e = latest_{q\to p\to r}(I)$ and we are done.

If this path does leave $I|_p \cap I|_q$, we can find an event e' along the path such that $e \sqsubseteq^* e' \lhd_{r'} f' \sqsubseteq^* max_p(I)$, where $e' \in I|_p \cap I|_q$, $f' \in I|_p - I|_q$ and $r' \in e' \cap f'$. In other words, e' is an r'-sentry for q with respect to p. We know by our earlier argument that $e' = latest_{q\to r'}(I)$. It must be the case that $e = latest_{r'\to r}(e'\downarrow)$. For, if $latest_{r'\to r}(e'\downarrow) = e'' \neq e$, then $e \lhd_r^+ e'' \sqsubseteq^* e' \sqsubseteq^* max_p(I)$. Since $e'' \in I|_p$ and $e \lhd_r^+ e''$, $e \neq latest_{p\to r}(I)$, which is a contradiction. So, $e = latest_{r'\to r}(e'\downarrow) = latest_{q\to r'\to r}(I)$ and we are done. □

Our observation about frontier events immediately gives us a way to compare primary information using both primary and secondary information.

Lemma 2. *Let I be an ideal and $p,q,r \in \mathscr{P}$. Let $e = latest_{p\to r}(I)$ and $f = latest_{q\to r}(I)$. Then $e \sqsubseteq^* f$ iff $e \in secondary_q(I)$.*

Proof. ($\Leftarrow$) Suppose $e \in secondary_q(I)$. Then $r \in e \in I|_q$ and so $e \sqsubseteq^* f \in I|_q$ by the definition of $latest_{q\to r}(I)$.
($\Rightarrow$) If $e = f$, $e \in primary_q(I) \subseteq secondary_q(I)$, and there is nothing to prove. If $e \neq f$, then there exists an event e' such that $e \lhd_r e' \lhd_r^* f$ and so $e \in I|_p \cap I|_q$. We know that $e' \in I|_q - I|_p$, so e is an r-sentry in $frontier_{pq}(I)$. But then, by our previous lemma, $e \in primary_p(I) \cap secondary_q(I)$ and we are done. □

Suppose p and q synchronize at an action a after u. At this point they "share" their primary and secondary information. For $r \notin \{p,q\}$, if the event $latest_{p\to r}(\mathscr{E}_u)$ is also present in q's set of secondary events $secondary_q(\mathscr{E}_u)$, both p and q know that q's latest r-event $latest_{q\to r}(\mathscr{E}_u)$ is at least as recent as $latest_{p\to r}(\mathscr{E}_u)$. So, after the synchronization, $latest_{q\to r}(\mathscr{E}_{ua})$ is the same as $latest_{q\to r}(\mathscr{E}_u)$, whereas p inherits this information from q – i.e., $latest_{p\to r}(\mathscr{E}_{ua}) = latest_{q\to r}(\mathscr{E}_u)$. In this way, for each $r \in \mathscr{P}$, p and q locally update their primary information about r in $\mathscr{E}_{ua}$. Clearly $latest_{p\to q}(\mathscr{E}_{ua}) = latest_{q\to p}(\mathscr{E}_{ua}) = e_a$, where e_a is the new event – i.e., $\mathscr{E}_{ua} - \mathscr{E}_u = \{e_a\}$.

This procedure generalizes to any arbitrary set $P \subseteq \mathscr{P}$ which synchronizes after u. The processes in P share their primary and secondary information and compare this information pairwise. Using Lemma 2, for each $q \in \mathscr{P} - P$ they decide who has the "latest information" about q. Each process then comes away with the best primary information from P.

Once we have compared primary information, updating secondary information is straightforward. Clearly, if $latest_{q\to r}(I)$ is better than $latest_{p\to r}(I)$, then every secondary event $latest_{q\to r\to r'}(I)$ must also be better than the corresponding event $latest_{p\to r\to r'}(I)$. So, secondary information can be locally updated too. In other words, to consistently update primary and secondary information, it suffices to correctly compare primary information, which is achieved by Lemma 2.

After a synchronization involving $P \subseteq \mathscr{P}$, notice that all processes in P will come away with the *same* set of primary and secondary events.

From the preceding argument, it is clear that the new event belongs to the primary (and hence secondary) information of the processes which synchronize at that event. Further, the update procedure reveals that if an event disappears from the secondary information of all the processes, it will never reappear as secondary information at some later stage. This is captured formally in the following proposition.

Proposition 3. *Let $u, w \in \Sigma^*$ such that $w = ua$ for some $a \in \Sigma$. Let e_a denote the new event in w – i.e., $\mathscr{E}_w - \mathscr{E}_u = \{e_a\}$. Then*:

– $e_a \in primary_{\mathscr{P}}(\mathscr{E}_w)$.
– $primary_{\mathscr{P}}(\mathscr{E}_w) \subseteq \{e_a\} \cup primary_{\mathscr{P}}(\mathscr{E}_u)$.
– $secondary_{\mathscr{P}}(\mathscr{E}_w) \subseteq \{e_a\} \cup secondary_{\mathscr{P}}(\mathscr{E}_u)$.

4 Locally updating primary/secondary information

To make Lemma 2 effective, we must make the assertions "locally checkable" – e.g., if $e = latest_{p\to r}(I)$, processes p and q must be able to decide if $e \in secondary_q(I)$.

Recall that e is represented in $primary_p(I)$ as a triple of the form (p, r, e). So, to check if $e \in secondary_q(I)$, q has to look for a quadruple of the form $(q, r', r'', e) \in secondary_q(I)$, where $r', r'' \in \mathscr{P}$. This can be checked locally provided events in $\mathscr{E}_u$ are labelled unambiguously while u is being read.

Clearly, labelling each event e as a pair $(i, u(i))$ is impossible since, in general, there is no agent which can consistently supply all processes with the "correct" value of i. Instead, we may naïvely assume that events in $\mathscr{E}_u$ are locally assigned distinct labels – in effect, at each action a, the processes in a together assign a (sequential) time-stamp to the new occurrence of a.[2] In this manner, the processes in $\mathscr{P}$ can easily assign consistent local time-stamps for each action which will let them compute the relations $\lhd_p^*$ between events.

The problem with this approach is that we will need an unbounded set of time-stamps, since u could get arbitrarily large. Instead we would like a scheme which uses only a finite set of labels to distinguish events. This means that several different occurrences of the same action will eventually get the same label. Since the update of primary and secondary information relies on comparing labels, we must ensure that this reuse of labels does not lead to any confusion.

However, from Lemma 2, we know that to compare primary information, we only need to look at the events which are currently in the primary and secondary sets of each process. So, it is sufficient if the labels assigned to these sets are consistent across the system – i.e., if the same label appears in the current primary or secondary information of different processes, the corresponding event is actually the same.

Notice that we do not need to maintain a global temporal order on labels across the system. Lemma 2 assures us that to compare events of interest to us, it suffices to check for *equality* of labels assigned to the events.

Suppose we have such a labelling on u and we want to extend this to a consistent labelling on $w = ua$ – i.e., we need to assign a label to the new a-event. By Proposition 3, it suffices to use a label which is distinct from the labels of all the a-events currently in the secondary information of $\mathscr{E}_u$. Since the cardinality of $secondary_{\mathscr{P}}(\mathscr{E}_u)$ is bounded, such a new label must exist. The catch is to detect which labels are currently in use and which are not.

Unfortunately the processes in a cannot directly see all the a-events which belong to the secondary information of the entire system. An a-event e may be part of the secondary information of processes *outside* a – i.e., $e \in secondary_{\mathscr{P}-a}(\mathscr{E}_u) - secondary_a(\mathscr{E}_u)$.

Example. Let $\mathscr{P} = \{p,q,r,s\}$ and $\Sigma = \{a,b,c,d,e\}$ where $a = \{p,q\}$, $b = \{q,r\}$, $c = \{r,s\}$, $d = \{p,s\}$ and $e = \{q,s\}$.

[2] Recall that for each action $\hat{a} \in \Sigma$, $\theta(\hat{a}) = a$, and we use a to denote both the action and the subset of processes which synchronize at that action

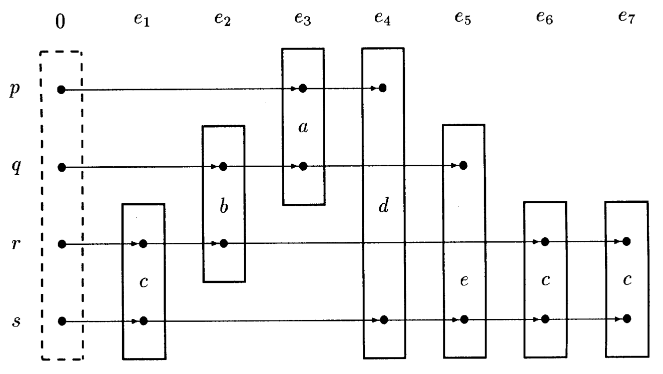

Fig. 2. At e_7, $e_1 \in secondary_p(e_7\downarrow)$, but r and s cannot "see" e_1 in $secondary_{\{r,s\}}(e_7\downarrow)$

Figure 2 shows the events $\mathscr{E}$ corresponding to the word *cbadecc*.

At the end of this word, $e_1 = latest_{p\to q\to s}(\mathscr{E})$. However, $e_1 \notin secondary_s(\mathscr{E})$;

$$\begin{aligned}secondary_s(\mathscr{E}) = \{&(s,p,p,e_4), (s,p,q,e_3), (s,p,r,e_2), (s,p,s,e_4),\\ &(s,q,p,e_4), (s,q,q,e_5), (s,q,r,e_2), (s,q,s,e_5),\\ &(s,r,p,e_4), (s,r,q,e_5), (s,r,r,e_7), (s,r,s,e_7),\\ &(s,s,p,e_4), (s,s,q,e_5), (s,s,r,e_7), (s,s,s,e_7)\}.\end{aligned}$$

Since $max_r(\mathscr{E}) = max_s(\mathscr{E})$, $secondary_r(\mathscr{E}) = secondary_s(\mathscr{E})$ when viewed as sets of events. So, $e_1 \notin secondary_r(\mathscr{E})$ either. Thus, e_1 is a c-event which belongs to $secondary_{\mathscr{P}}(\mathscr{E}) - secondary_c(\mathscr{E})$.

To enable the processes in a to know about all a-events in $secondary_{\mathscr{P}}(\mathscr{E}_u)$, we need to maintain tertiary information.

Tertiary information. The *tertiary information* of p after I, $tertiary_p(I)$, is the (indexed) set $\bigcup_{q\in\mathscr{P}} secondary_q(latest_{p\to q}(I)\downarrow)$. In other words, this is the latest information that p has in I about the secondary information of q, for all $q \in \mathscr{P}$. As before, for $P \subseteq \mathscr{P}$, $tertiary_P(I) = \bigcup_{p\in P} tertiary_p(I)$.

Each event in $tertiary_p(I)$ is of the form $latest_{q\to r\to s}(latest_{p\to q}(I)\downarrow)$ for some $q,r,s \in \mathscr{P}$. We abbreviate $latest_{q\to r\to s}(latest_{p\to q}(I)\downarrow)$ by $latest_{p\to q\to r\to s}(I)$. We represent each event $e = latest_{p\to q\to r\to s}(I)$ as a quintuple (p,q,r,s,e) in $tertiary_p(I)$. However, for convenience we will work with $tertiary_p(I)$ as though it were simply a set of events, rather than an indexed set, just as we have been doing with primary and secondary information.

Just as $primary_p(I) \subseteq secondary_p(I)$, clearly $secondary_p(I) \subseteq tertiary_p(I)$ since each secondary event $latest_{p\to q\to r}(I)$ is also a tertiary event $latest_{p\to p\to q\to r}(I)$, (or, equivalently, $latest_{p\to q\to q\to r}(I)$ and so on).

Lemma 4. *Let I be an ideal and $p \in \mathscr{P}$. If $e \in secondary_p(I)$ then for every $q \in e$, $e \in tertiary_q(I)$.*

Proof. Let $e \in secondary_p(I)$ and $q \in e$. Concretely, let $e = latest_{p\to p'\to p''}(I)$ for some $p', p'' \in \mathscr{P}$. We know that $e \in I|_p \cap I|_q$ and there is a path $e \sqsubset f_1 \sqsubset \cdots \sqsubset max_p(I)$ leading from e to $max_p(I)$ which passes through $e' = latest_{p\to p'}(I)$.

Suppose this path never leaves $I|_p \cap I|_q$. Then $max_p(I) \in I|_q$ and so $max_p(I) = latest_{q\to p}(I)$. This means that $e \in secondary_p(latest_{q\to p}(I)\downarrow) \subseteq tertiary_q(I)$ and we are done.

Otherwise, the path from e to $max_p(I)$ does leave $I|_p \cap I|_q$ at some stage.

If $e' \notin I|_p \cap I|_q$ then for some $f, f' \in \mathscr{E}$ and some $r \in \mathscr{P}$ we have $f \in I|_p \cap I|_q$, $f' \in I|_p - I|_q$ and $e \sqsubseteq^* f \lessdot_r f' \sqsubseteq^* e'$. This means that $f \in frontier_{qp}(I)$ is an r-sentry and by our earlier argument we know that $f = latest_{q\to r}(I)$. So $e = latest_{q\to r\to p''}(I) = latest_{q\to q\to r\to p''}(I) \in tertiary_q(I)$.

On the other hand, if $e' \in I|_p \cap I|_q$ we can find an r-sentry $f \in frontier_{qp}(I)$ on the path from e' to $max_p(I)$, for some $r \in \mathscr{P}$. We once again get $f = latest_{q\to r}(I)$ and so $e = latest_{q\to r\to p'\to p''}(I) \in tertiary_q(I)$. $\square$

We shall use this lemma in the following form.

Corollary 5. *Let I be an ideal, $p \in \mathscr{P}$ and e a p-event in I. If $e \notin tertiary_p(I)$ then $e \notin primary_{\mathscr{P}}(I) \cup secondary_{\mathscr{P}}(I)$.*

So, a process p can keep track of which of its labels are "in use" in the system by maintaining tertiary information. Each p-event e initially belongs to $primary_e(I)$, and hence to $secondary_e(I)$ and $tertiary_e(I)$ as well. (Recall that for an event e, we also use e to denote the subset of $\mathscr{P}$ which meets at e). As the computation progresses, e gradually "recedes" into the background and disappears from the primary and secondary sets of the system. Eventually, when e disappears from $tertiary_p(I)$, p can be sure that e no longer belongs to $primary_{\mathscr{P}}(I) \cup secondary_{\mathscr{P}}(I)$.

Since $tertiary_p(I)$ is a bounded set, p knows that only finitely many of its labels are in use at any given time. So, by using a sufficiently large finite set of labels, each new event can always be assigned an unambiguous label by the processes which take part in the event.

5 The "gossip" automaton

Using our analysis of primary, secondary and tertiary information of processes, we can now design a deterministic asynchronous automaton to keep track of the "latest gossip" – i.e., to consistently update primary information whenever a set of processes synchronizes. The processes in the automaton maintain this information in terms of time-stamps: when an event e occurs, it is assigned a time-stamp $\tau(e)$ by the processes which take part in e.

For $p \in \mathscr{P}$, a local state of p is defined by three associative arrays prim_p, sec_p and ter_p, indexed by $\mathscr{P}$, $\mathscr{P}\times\mathscr{P}$ and $\mathscr{P}\times\mathscr{P}\times\mathscr{P}$ respectively. Each entry in these arrays is a pair of the form $\langle P, \ell\rangle$, where P is a subset of $\mathscr{P}$ and ℓ is a label drawn from a finite set $\mathscr{L}$ – it is sufficient for $\mathscr{L}$ to have $N^3 + 1$ entries, where N is the number of processes in $\mathscr{P}$.

After reading a word u, each entry $\langle P, \ell\rangle$ in the arrays prim_p, sec_p and ter_p corresponds to the time-stamp $\tau(e)$ of some event $e \in \mathscr{E}_u$. We shall show that for all $q,r,s \in \mathscr{P}$, the values stored in $\mathsf{prim}_p(q)$, $\mathsf{sec}_p(q,r)$ and $\mathsf{ter}_p(q,r,s)$ are, in fact, the time-stamps of the events $latest_{p\to q}(\mathscr{E}_u)$, $latest_{p\to q\to r}(\mathscr{E}_u)$ and $latest_{p\to q\to r\to s}(\mathscr{E}_u)$, respectively.

The initial state is the global state in which, for each process p, all entries in prim_p, sec_p and ter_p are set to

$\langle \mathscr{P}, \ell_0 \rangle$, where ℓ_0 is an arbitrary but fixed label from $\mathscr{L}$. In other words, the processes jointly assign the time-stamp $\tau(0) = \langle \mathscr{P}, \ell_0 \rangle$ to the initial event 0.

The local transition functions $\rightarrow_a$ modify the local states for processes in a as follows.

(i) When an a-event e_a occurs, the processes in a first choose a time-stamp $\tau(e_a) = \langle a, \ell \rangle$ such that for some $p \in a$, $\langle a, \ell \rangle$ does not appear in ter_p.
(Recall that $\mathscr{L}$ has $N^3 + 1$ labels. Since there are at most N^3 labels in ter_p for each p, we will always be able to find an unused label in $\mathscr{L}$ to assign to e_a. In general, we have to choose $\langle a, \ell \rangle$ in a canonical way. One possibility is to linearly order the sets $\mathscr{P}$ and $\mathscr{L}$. Let p_a be the smallest process in a with respect to the ordering on $\mathscr{P}$. To fix the time-stamp $\langle a, \ell \rangle$ assigned to e_a, choose ℓ such that it is the smallest label with respect to the ordering on $\mathscr{L}$ which does not appear in ter_{p_a}.)

(ii) Let $a = \{p_1, p_2, \ldots, p_k\}$. Corresponding to each process $r \notin a$, fix a process $\pi_r \in a$ as follows.

```
$\pi_r := p_1$;
for each i in $\{2, 3, \ldots, k\}$ do
    if the value stored in $\mathsf{prim}_{\pi_r}(r)$ appears in $\mathsf{sec}_{p_i}$
    then $\pi_r := p_i$;
od
```

(iii) For each $p \in a$, update prim_p, sec_p and ter_p in phases.

- First, update $\{\mathsf{prim}_p\}_{p \in a}$ as follows.
  For each $q \in a$, $\mathsf{prim}_p(q) := \langle a, \ell \rangle$.
  For each $r \in \mathscr{P} - a$, $\mathsf{prim}_p(r) := \mathsf{prim}_{\pi_r}(r)$.
- Next, update $\{\mathsf{sec}_p\}_{p \in a}$ as follows.
  For each $q, r \in \mathscr{P}$ such that $q \in a$,
  $\mathsf{sec}_p(q, r) := \mathsf{prim}_q(r)$.
  For each $q, r \in \mathscr{P}$ such that $q \notin a$,
  $\mathsf{sec}_p(q, r) := \mathsf{sec}_{\pi_q}(q, r)$.
- Finally, update $\{\mathsf{ter}_p\}_{p \in a}$ as follows.
  For each $q, r, s \in \mathscr{P}$ such that $q \in a$,
  $\mathsf{ter}_p(q, r, s) := \mathsf{sec}_q(r, s)$.
  For each $q, r, s \in \mathscr{P}$ such that $q \notin a$,
  $\mathsf{ter}_p(q, r, s) := \mathsf{ter}_{\pi_q}(q, r, s)$.

We now verify that the arrays maintained by each process p in the gossip automaton always record the time-stamps of the primary, secondary and tertiary information of p.

Proposition 6. *Let $u \in \Sigma^*$. For all processes $p \in \mathscr{P}$, the values stored in prim_p, sec_p and ter_p after reading u satisfy the following properties.*

- *For all $q \in \mathscr{P}$, $\mathsf{prim}_p(q) = \tau(\mathit{latest}_{p \to q}(\mathscr{E}_u))$.*
- *For all $q, r \in \mathscr{P}$, $\mathsf{sec}_p(q, r) = \tau(\mathit{latest}_{p \to q \to r}(\mathscr{E}_u))$.*
- *For all $q, r, s \in \mathscr{P}$, $\mathsf{ter}_p(q, r, s) = \tau(\mathit{latest}_{p \to q \to r \to s}(\mathscr{E}_u))$.*

Moreover, for all processes $p, q \in \mathscr{P}$ and events $e \in \mathit{primary}_p(\mathscr{E}_u) \cup \mathit{secondary}_p(\mathscr{E}_u)$ and $f \in \mathit{primary}_q(\mathscr{E}_u) \cup \mathit{secondary}_q(\mathscr{E}_u)$, if $\tau(e) = \tau(f)$ then $e = f$.

Proof. The proof is by induction on n, the length of u.
($n = 0$)

The base case is when u is the empty string. We know that for each $p \in \mathscr{P}$, all entries in prim_p, sec_p and ter_p are initially set to $\langle \mathscr{P}, \ell_0 \rangle$, where $\langle \mathscr{P}, \ell_0 \rangle$ is the time-stamp assigned to the initial event 0. Clearly, these values satisfy all the conditions specified in the statement of the proposition.
($n > 0$)

Suppose $u = wa$. The time-stamp $\langle a, \ell \rangle$ assigned to the new a-event e_a in step (i) of the transition does not appear in ter_p for some $p \in a$. By the induction hypothesis, the values stored in ter_p after reading w are the time-stamps of the corresponding events in $\mathit{tertiary}_p(\mathscr{E}_w)$. Lemma 4 then guarantees that no event in $\mathit{primary}_{\mathscr{P}}(\mathscr{E}_w) \cup \mathit{secondary}_{\mathscr{P}}(\mathscr{E}_w)$ is time-stamped $\langle a, \ell \rangle$ – if there were such an event, it would be in the tertiary information $\mathit{tertiary}_q(\mathscr{E}_w)$ of *every* process $q \in a$ as a result of which, by the induction hypothesis, the label $\langle a, \ell \rangle$ would be in ter_q after reading w for *every* $q \in a$.

By the induction hypothesis, we also know that no two distinct events in $\mathit{primary}_{\mathscr{P}}(\mathscr{E}_w) \cup \mathit{secondary}_{\mathscr{P}}(\mathscr{E}_w)$ are assigned the same time-stamp. Since $\mathit{primary}_{\mathscr{P}}(\mathscr{E}_u) \subseteq \{e_a\} \cup \mathit{primary}_{\mathscr{P}}(\mathscr{E}_w)$ and $\mathit{secondary}_{\mathscr{P}}(\mathscr{E}_u) \subseteq \{e_a\} \cup \mathit{secondary}_{\mathscr{P}}(\mathscr{E}_w)$ (Proposition 3), it follows that the time-stamps assigned to $\mathit{primary}_{\mathscr{P}}(\mathscr{E}_u) \cup \mathit{secondary}_{\mathscr{P}}(\mathscr{E}_u)$ are also distinct. In other words, for all processes $p, q \in \mathscr{P}$ and events $e \in \mathit{primary}_p(\mathscr{E}_u) \cup \mathit{secondary}_p(\mathscr{E}_u)$ and $f \in \mathit{primary}_q(\mathscr{E}_u) \cup \mathit{secondary}_q(\mathscr{E}_u)$, if $\tau(e) = \tau(f)$ then $e = f$.

We now have to verify that the updated values in prim_p, sec_p and ter_p for each $p \in a$ are the time-stamps of the corresponding events in the primary, secondary and tertiary information of p after u.

By the induction hypothesis, for each $p \in a$, the values in prim_p and sec_p after reading w are the time-stamps assigned to the corresponding events in $\mathit{primary}_p(\mathscr{E}_w)$ and $\mathit{secondary}_p(\mathscr{E}_w)$ respectively. By the induction hypothesis, we also know that for all $p, q, r, s, s' \in \mathscr{P}$, if the value stored in $\mathsf{prim}_p(r)$ after reading w is the same as the value stored in $\mathsf{sec}_q(s, s')$ after reading w then, in fact, the event $\mathit{latest}_{p \to r}(\mathscr{E}_w)$ is the same as the event $\mathit{latest}_{q \to s \to s'}(\mathscr{E}_w)$.

It then follows that for $r \notin a$, the process $\pi_r \in a$ identified in step (ii) of the transition is one of the processes in a which has the best primary information about r after w – by Lemma 2, for $p, q \in a$ and $r \notin a$, $\mathit{latest}_{p \to r}(\mathscr{E}_w) \sqsubseteq^* \mathit{latest}_{q \to r}(\mathscr{E}_w)$ iff the event $\mathit{latest}_{p \to r}(\mathscr{E}_w)$ appears in $\mathit{secondary}_q(\mathscr{E}_w)$, which is equivalent to checking that the time-stamp $\tau(\mathit{latest}_{p \to r}(\mathscr{E}_w))$ stored in $\mathsf{prim}_p(r)$ appears in sec_q.

From this, it is easy to see that for all $p \in a$, the updates made to prim_p, sec_p and ter_p in step (iii) ensure that for all $q, r, s \in \mathscr{P}$, $\mathsf{prim}_p(q) = \tau(\mathit{latest}_{p \to q}(\mathscr{E}_u))$, $\mathsf{sec}_p(q, r) = \tau(\mathit{latest}_{p \to q \to r}(\mathscr{E}_u))$ and $\mathsf{ter}_p(q, r, s) = \tau(\mathit{latest}_{p \to q \to r \to s}(\mathscr{E}_u))$. $\square$

The gossip automaton does not have any final states, since we do not need to accept any language. Instead, we define for each $a \in \Sigma$ a function $g_a : S_a \to (2^{\mathscr{P}} - \{\emptyset\})^{\mathscr{P}}$ which checks the arrays prim_p and sec_p of each process p in a and computes for each $q \in \mathscr{P}$, the set of processes in a which have the most recent information about q.

(A small technical point: g_a must be defined for all states in S_a. However, not all combinations of local states may be "meaningful". We can easily assemble local states to form an a-state for which the inductive assertions do not hold, as a result of which our procedure for comparing primary information breaks down. However, since such a-states are unreachable, we can ignore this problem and simply assign a default value to g_a in these cases –

for instance, the default value could be the function $\{p \mapsto a\}_{p \in \mathscr{P}}$.)

This immediately yields the result we set out to establish.

Theorem 7. *Let (Σ, θ) be the distributed alphabet corresponding to $\mathscr{C} \subseteq (2^{\mathscr{P}} - \{\emptyset\})$. The family of functions $\{latest\text{-}gossip_c : \Sigma^* \to (2^{\mathscr{P}} - \{\emptyset\})^{\mathscr{P}}\}_{c \in \Sigma}$ is locally computable.*

The size of the gossip automaton

Lemma 8. *In the gossip automaton, the local state of each process $p \in \mathscr{P}$ can be described using $O(N^3 \log N)$ bits, where $N = |\mathscr{P}|$.*

Proof. A local state for p consists of the arrays prim_p, sec_p and ter_p. We estimate how many bits are required to store this.

Recall that for any ideal I, each event in $primary_p(I)$ is also present in $secondary_p(I)$. Similarly, each event in $secondary_p(I)$ is also present in $tertiary_p(I)$. So it suffices to store just the labels of tertiary events in the array ter_p. By fixing an ordering of $\mathscr{P} \times \mathscr{P} \times \mathscr{P}$, these events can be stored as a list with N^3 entries.

Each new event e was assigned a label of the form $\langle P, \ell \rangle$, where P was the set of processes that participated in e and $\ell \in \mathscr{L}$.

We have already seen that it suffices to have $O(N^3)$ labels in $\mathscr{L}$. As a result, each label $\ell \in \mathscr{L}$ can be written down using $O(\log N)$ bits.

To write down $P \subseteq \mathscr{P}$, we need, in general, N bits. This component of the label is required to guarantee that all secondary events in the system have distinct labels, since the set $\mathscr{L}$ is common across all processes. However, we do not really need to use all of P in the label for e to ensure this property. If we fix a linear order on $\mathscr{P}$, it suffices to label e by $\langle p_e, \ell \rangle$ where, among the processes participating in e, p_e is the smallest with respect to the ordering on $\mathscr{P}$. It is easy to verify that Proposition 6 continues to hold with respect to these modified labels.

Thus, we can modify our automaton so that the processes label each event by a pair $\langle p, \ell \rangle$, where $p \in \mathscr{P}$ and $\ell \in \mathscr{L}$. This pair can be written down using $O(\log N)$ bits. Overall there are N^3 such pairs in the array of tertiary events, so the whole state can be described using $O(N^3 \log N)$ bits. $\square$

The preceding lemma implies that the number of local states of a process could be exponential in N, the number of processes in the system. In general, this would mean that we require an exponential amount of space to specify the entire automaton (though each state can be described using a polynomial number of bits) since the transition table of the automaton would have an exponential number of entries.

However, in this case, we do not need to exhaustively enumerate the entire state space and transition table to completely describe the automaton. Since the states are structured entities and the transition function is presented as an algorithm which manipulates the data in these structured states, the gossip automaton can in fact be effectively presented in space polynomial in N. This ability to build the gossip automaton efficiently "on the fly" is crucial for embedding it in other distributed algorithms.

6 Extensions and applications

Beyond tertiary information

The gossip automaton consistently labels all primary and secondary events. In other words, at any point, distinct primary and secondary events have distinct time-stamps. By Lemma 2, this is sufficient to correctly compare and update primary information.

However, we may want to keep track of events which are older than secondary events. We can generalize the definition of secondary and tertiary information and inductively define the k-ary information of a process p with respect to an ideal I, for any natural number k.

The case $k = 1$ corresponds to primary information. For $k > 1$, the *k-ary information* of p after I, $k\text{-}ary_p(I)$, is the (indexed) set $\bigcup_{q \in \mathscr{P}} (k-1)\text{-}ary_p(latest_{p \to q}(I)\downarrow)$. In other words, this is the latest information that p has in I about the $(k-1)$-ary information of q, for all $q \in \mathscr{P}$.

It is not difficult to see that the argument in Lemma 4 can be extended to yield the following result, proved formally in [15].

Lemma 9. *Let I be an ideal and $p \in \mathscr{P}$. If $e \in k\text{-}ary_p(I)$ then for every $q \in e$, $e \in (k+1)\text{-}ary_q(I)$.*

Together with Lemma 2, this implies that the gossip automaton can be modified to maintain consistent time-stamps for all events upto depth k, for any natural number k.

Optimizing the automaton

If we are only concerned with comparing and updating primary information, the gossip automaton can be made more concise. To achieve this, each process maintains its primary information as a directed graph which reflects the underlying $\sqsubseteq^*$-order between primary events, rather than storing the information as a simple array of labels as we have described here. It turns out that processes can compare and update these *primary graphs* without recourse to secondary information. Secondary information is then required only to ensure that new events get unused labels. Tertiary information is dispensed with altogether. This leads to the following result (details can be found in [15]).

Lemma 10. *The gossip automaton can be modified so that the local state of each process $p \in \mathscr{P}$ is describable using $O(N^2 \log N)$ bits, where $N = |\mathscr{P}|$.*

It is important to note that this optimized automaton consistently labels *only* primary events. Secondary events could get inconsistent labels. This becomes relevant when we come to applications of the gossip automaton.

Further optimizations are possible if the structure of the alphabet (Σ, θ) is such that synchronizations are "well-behaved". One example is when the processes are located on the vertices of an d-dimensional hypercube (i.e., $|\mathscr{P}| = 2^d$), with synchronizations taking place along faces of the hypercube. Once again, details can be found in [15].

Applications

Building the gossip automaton is a basic step in tackling many problems in the theory of asynchronous automata.

Asynchronous automata play an important role in the theory of distributed systems because of their close connection to *trace theory*. Trace theory, initiated by Mazurkiewicz [19], is a language-theoretic approach to the study of concurrent systems. Traditionally, formal language theory models computations of sequential systems as strings over an abstract alphabet. Instead, trace theory describes computations of concurrent systems in terms of equivalence classes of strings, called *traces*. All strings in a trace are equivalent upto permutation of adjacent letters which belong to an underlying *independence relation* on the alphabet. Thus, the different elements of a trace correspond to different sequential observations of the same concurrent computation.

A deep theorem of Zielonka [30] shows that asynchronous automata are a natural distributed machine model for recognizing trace languages. Zielonka's original proof of the connection between these automata and recognizable trace languages is widely accepted to be difficult to assimilate. In [23], we show that the gossip automaton can be used to provide a more structured proof of Zielonka's theorem. Overall, the construction we present is quite similar to the one described in Zielonka's original paper. However, we feel that by separating out clearly the role played by the gossip automaton, our proof is much easier to digest. (Other new proofs of Zielonka's theorem exist [2], but these are based on asynchronous *cellular* automata [31], a slightly different – and, in our opinion, less intuitive – machine model.)

In [13], a determinization construction is presented for asynchronous automata using a generalization of the classical subset construction for finite automata. This construction allows us to keep track of the global states which are currently valid at any stage of a computation by a non-deterministic asynchronous automaton.

The key problem to be tackled in the determinization construction is to estimate the amount of information that can be safely "forgotten" by the subset automaton without sacrificing global consistency. It turns out that it is sufficient for each process to maintain "histories" upto the level of secondary events. This is accomplished by using the gossip automaton presented here to assign consistent time-stamps to all secondary events. Notice that the optimized gossip automaton of [15] cannot be used in the determinization construction since it only guarantees consistent time-stamps upto primary events.

In the past few years, there has been a lot of interest in extending asynchronous automata to process infinite inputs. Analogous to Büchi automata on infinite strings, Gastin and Petit [7] have defined Büchi asynchronous automata which operate on infinite traces. Although it has been known for a while that these automata are closed under complementation for algebraic reasons, the only direct complementation construction provided so far has been for Büchi asynchronous *cellular* automata [24]. Safra's determinization construction for Büchi automata [26] has recently been extended to Büchi asynchronous automata [14]. This provides a natural procedure for complementing these automata. Once again, the gossip automaton plays a crucial role in the construction.

Automata on infinite strings have always been closely linked to logic [28]. This connection holds for Büchi asynchronous automata as well. Recently, Thiagarajan [27] has developed an extension of propositional linear-time temporal logic which is interpreted over infinite traces rather than infinite linear sequences. This logic appears to be quite expressive, while remaining decidable (unlike several other partial-order logics studied in the literature [18]). The decision procedure for this logic is automata-theoretic, in the style of Vardi and Wolper [29], except that it makes use of Büchi asynchronous automata rather than conventional Büchi automata. The gossip automaton plays a crucial role in this construction as well. In fact, *this* was the original motivation for constructing the gossip automaton.

Asynchronous communication

In this paper, we have only dealt with synchronous communication. Synchronous communication achieves coordination among independent agents by periodically permitting subsets of agents to pool information together to make a decision.

Another standard way for agents to exchange information is through message-passing. This mode of exchanging information is usually referred to as asynchronous communication, since there may be an arbitrary delay between the time when a message is sent and the time when it is received.

Synchronous communication is easier to handle, both from a theoretical standpoint as well as from the point of view of programming. It is no coincidence that languages like CCS [20] and CSP [11] which have been developed for specifying communicating systems assume synchronous communication as the basic means of exchanging information.

However, when agents are widely separated in space, asynchronous communication is generally the only practical way of achieving coordination. So, there is a considerable body of literature devoted to distributed algorithms and protocols for asynchronous systems [16].

We have extended our approach to deal with message-passing systems where the communication medium is reliable, subject to certain restrictions on the number of unacknowledged messages that can be present in the system at any given time [21]. This shows that our analysis based on primary, secondary and tertiary information is applicable to a much wider range of distributed systems than the purely synchronous systems discussed in this paper.

7 Discussion

We now discuss the connections between our results and other work in related areas.

Studies of "gossiping" in networks have traditionally focussed on efficiently disseminating a fixed piece of information (or gossip) from one node to all other nodes in a network [9]. The main aim is to find an optimal sequence of communications to distribute data for a given network topology.

Israeli and Li [12] introduced the notion of "bounded time-stamps" and argued that these were fundamental in solving many problems in distributed systems – notably that of creating what are called "atomic registers" [17]. Their results have been extended and considerably simplified by a number of others [3, 5, 6, 8]. However, this line of work is based on a shared memory model, which is quite different in spirit from the asynchronous automaton model. Though it may be possible to formally translate their results to our framework, we feel that the intuition underlying their time-stamping algorithms is quite different from ours. In general, solutions to time-stamping problems seem to depend heavily on the underlying model of a distributed system that is used and it is difficult to compare such algorithms across models.

More closely connected to our work are the bounded time-stamping algorithms implemented using asynchronous cellular automata in the study of recognizable trace languages [1, 2]. The synchronization mechanism of asynchronous cellular automata is quite different from that of asynchronous automata. Asynchronous cellular automata correspond more closely to shared memory systems than to systems that communicate by direct interprocess communication. Thus, though the overall goal of the constructions in [1, 2] is closely related to the gossip problem which we have studied here, the two time-stamping algorithms appear to be incomparable since they are based on fundamentally different assumptions about how processes interact.

Our definition of locally computable functions is closely linked to the notion of *asynchronous mappings* which is fundamental to the constructions used in [2]. It would be interesting to formally characterize the class of locally computable functions and develop a methodology for automatically synthesizing an asynchronous automaton to compute a given locally computable function, as is done for asynchronous mappings in [2].

Though our algorithm can be implemented as an asynchronous automaton, it correctly computes the latest gossip function locally for *any* input word. In other words, the set of communication sequences generated by the underlying system need not be regular. Our algorithm will also work on sequences generated, for instance, by N communicating Turing machines. Thus, any distributed algorithm which needs to keep track of the flow of information between processes can incorporate the gossip automaton as a "background" routine. Moreover, since the gossip automaton can be constructed efficiently "on the fly", embedding it in another algorithm does not involve a very significant computational overhead.

Since our algorithm does not add any extra messages to the underlying computation, it is automatically resilient to failures – even if some processes stop functioning, our algorithm will continue to update primary information correctly for those parts of the system which are still active in the underlying computation. This in-built fault-tolerance is also present in the shared memory time-stamping protocols of [1, 2, 3, 5, 8, 12] and others.

In an asynchronous automaton, each move is *simultaneously* performed by all participating processes. This strong atomicity assumption is not crucial for our algorithm. Notice that the label of an event is *uniquely* fixed by the primary, secondary and tertiary information of the participating processes. Thus, we could implement each multi-way synchronization as a series of synchronous communications where processes exchange their local information pairwise and then proceed independently. The deterministic nature of our algorithm guarantees that each process which participates in an event will locally choose the *same* time-stamp for the new event, since all processes taking part in the event base their choice of time-stamp on identical shared information. Similarly, once each process has collected the primary, secondary and tertiary information of every other participating process, it can update its local information unambiguously. Moreover, it can also compute the updated information of every other process which took part in the event.

The construction of the gossip automaton establishes a non-obvious property for *all* systems with synchronous communication. Suppose an agent p_1 has a private variable X which no other agent can modify, and agents $\{p_2, p_3, \ldots, p_N\}$ keep track of the latest value of X that they have heard of from p_1 (either directly or indirectly). Then, along *any* run of the system, bounded time-stamps suffice for determining which of $\{p_2, p_3, \ldots, p_N\}$ have the most recent value of X. This is important, for example, for crash recovery. If the system crashes and p_1 fails to come alive after the crash, the other agents can get together and synthesize an optimal "last-known" state of p_1 by comparing their information about p_1.

There are obvious parallels between the notion of primary, secondary and tertiary information we use in this paper and the concept of levels of knowledge about events in a distributed system [10, 25]. It would be interesting to formally work out how our approach fits in with knowledge-theoretic techniques for analyzing distributed systems. As far as we are aware, none of the work in knowledge theory has addressed the synchronizing model that we consider here, so establishing a precise connection between the two approaches is not straightforward.

Acknowledgements. P.S. Thiagarajan suggested the gossip problem. We have benefited greatly from discussions with him. The reformulation of the gossip automaton in terms of ideals and frontiers emerged during work with Nils Klarlund on determinizing asynchronous automata. His suggestions have been a great help in cleaning up the presentation. We also thank Robert de Simone for pointing out some subtle technical errors. Finally, we thank the referees for several useful comments.

References

1. Cori R, Metivier Y: Approximations of a trace, asynchronous automata and the ordering of events in a distributed system. Proc ICALP'88. LNCS 317: 147–161 (1988)
2. Cori R, Metivier Y, Zielonka W: Asynchronous mappings and asynchronous cellular automata. Inf Comput 106: 159–202 (1993)
3. Cori R, Sopena E: Some combinatorial aspects of time-stamp systems. Eur J Comb 14: 95–102 (1993)
4. Diekert V: Combinatorics on traces. LNCS 454 (1990)
5. Dolev D, Shavit N: Bounded concurrent time-stamps are constructible. Proc 21st ACM STOC, pp 454–466, 1989

6. Dwork C, Waarts O: Simple and efficient bounded concurrent timestamping or bounded concurrent time-stamps are comprehensible. Proc 24th ACM STOC, pp 655–666, 1992
7. Gastin P, Petit A: Asynchronous cellular automata for infinite traces. Proc ICALP'92. LNCS 623: 583–594 (1992)
8. Gawlick R, Lynch N, Shavit N: Concurrent time stamping made simple. Proc ISTCS'92. LNCS 601: 171–183 (1992)
9. Hedetniemi SM, Hedetniemi ST, Liestman AL: A survey of gossiping and broadcasting in communication networks. Networks 18: 319–349 (1988)
10. Halpern J, Moses Y: Knowledge and common knowledge in a distributed environment. J Assoc Comput Mach 37: 549–578 (1990)
11. Hoare CAR: Communicating sequential processes. Prentice-Hall, London 1984
12. Israeli A, Li M: Bounded time-stamps. Proc 28th IEEE FOCS, pp 371–382, 1987
13. Klarlund N, Mukund M, Sohoni M: Determinizing asynchronous automata. Proc ICALP 1994. LNCS 820: 130–141 (1994)
14. Klarlund N, Mukund M, Sohoni M: Determinizing Büchi asynchronous automata. Proc FST&TCS 1995. LNCS 1026: 456–470 (1995)
15. Krishnan R, Venkatesh S: Optimizing the gossip automaton. Report TCS-94-3, School of Mathematics, SPIC Science Foundation, Madras, India 1994
16. Lamport L, Lynch N: Distributed computing: models and methods. In: van Leeuwen (ed) Handbook of theoretical computer science, vol B. North-Holland, Amsterdam 1990, pp 1157–1200
17. Li M, Vitanyi P: How to share concurrent asynchronous wait free variables. Proc ICALP'89. LNCS 372: 488–507 (1989)
18. Lodaya K, Parikh R, Ramanujam R, Thiagarajan PS: A logical study of distributed transition systems. Inf Comput 119: 91–118 (1995)
19. Mazurkiewicz A: Basic notions of trace theory. In: de Bakker JW, de Roever W-P, Rozenberg G (eds) Linear time, branching time and partial order in logics and models for concurrency. LNCS 354: 285–363 (1989)
20. Milner R: Communication and concurrency. Prentice-Hall, London 1989
21. Mukund M, Narayan Kumar K, Sohoni M: Keeping track of the latest gossip in message-passing systems. Proc Structures in Concurrency Theory (STRICT), Berlin 1995. Workshops in Computing Series, Springer, Berlin Heidelberg New York, 1995, pp 249–263
22. Mukund M, Sohoni M: Keeping track of the latest gossip: bounded time-stamps suffice. Proc FST&TCS'93. LNCS 761: 388–399 (1993)
23. Mukund M, Sohoni M: Gossiping, asynchronous automata and Zielonka's theorem. Report TCS-94-2, School of Mathematics, SPIC Science Foundation, Madras, India 1994
24. Muscholi A: On the complementation of Büchi asynchronous cellular automata. Proc ICALP 1994. LNCS 820: 142–153 (1994)
25. Parikh R, Ramanujam R: Distributed processes and the logic of knowledge. Proc Logics of programs'85. LNCS 193: 256–268 (1985)
26. Safra S: On the complexity of ω-automata. Proc 29th IEEE FOCS, pp 319–327, 1988
27. Thiagarajan PS: TrPTL: a trace based extension of linear time temporal logic. Proc 9th IEEE LICS, pp 438–447, 1994
28. Thomas W: Automata on infinite objects. In: van Leeuwen J (ed) Handbook of theoretical computer science, vol B. North-Holland, Amsterdam 1990, pp 133–191
29. Vardi M, Wolper P: An automata theoretic approach to automatic program verification. Proc 1st IEEE LICS, pp 332–345, 1986
30. Zielonka W: Notes on finite asynchronous automata. R.A.I.R.O. – Inf Théor Appl 21: 99–135 (1987)
31. Zielonka W: Safe executions of recognizable trace languages. Proc Logical Foundations of Computer Science. LNCS 363: 278–289 (1989)

Madhavan Mukund received his B. Tech. degree in Computer Science and Engineering from the Indian Institute of Technology (IIT), Bombay in 1986 and his Ph.D. in Computer Science from Aarhus University, Aarhus, Denmark in 1992. Since 1992, he has been on the faculty at the SPIC Mathematical Institute, Madras (formerly known as the School of Mathematics, SPIC Science Foundation) where he is presently a Reader. His research interests include partial order-based approaches to modelling concurrent systems and developing appropriate logics to reason about the behaviour of such systems.

Milind Sohoni also received his B. Tech. in Computer Science and Engineering from IIT, Bombay in 1986. He received his M.S. from the University of Illinois, Urbana-Champaign in 1989, and his Ph.D. from IIT, Bombay in 1993. During 1993–1994, he was a Fellow at the School of Mathematics, SPIC Science Foundation, Madras. He is currently an Assistant Professor in Computer Science and Engineering at IIT, Bombay. His research interests include Concurrency and Distributed Computing, Combinatorial Optimization and Algebraic Algorithms.