Internal Report TCS-93-5
October, 1993

Determinizing Asynchronous Automata

Nils Klarlund[1], Madhavan Mukund[2], Milind Sohoni[2]

Abstract

A concurrent version of a finite-state automaton is a set of processes that cooperate in processing letters of the input. Each letter read prompts some of the processes to synchronize and decide on a joint move according to a non-deterministic transition relation. Such automata are known as *asynchronous automata*.

The question whether these automata can be determinized while retaining the synchronization structure has already been answered in the positive, but indirectly, by means of sophisticated algebraic techniques.

In this paper we present an elementary proof, which generalizes the classic subset construction for finite-state automata. The proof uses in an essential way an earlier finite-state construction by Mukund and Sohoni for maintaining each process's latest knowledge about other processes.

Our construction is only double-exponential and thus is the first to essentially match the lower bound.

In conjunction with earlier results of Ochmanski and Pighizzini, our construction provides a new (and in a sense "classical") proof of Zielonka's theorem that every recognizable trace language is accepted by a deterministic asynchronous automaton whose structure precisely captures the independence relation of the given trace alphabet.

[1]Computer Science Department, Aarhus University, Ny Munkegade, DK 8000 Aarhus C, Denmark. E-mail: klarlund@daimi.aau.dk

[2]School of Mathematics, SPIC Science Foundation, 92 G N Chetty Road, T Nagar, Madras 600 017, India. E-mail: {madhavan,sohoni}@ssf.ernet.in

This report is also published as Report DAIMI PB-460, Computer Science Department, Aarhus University, Aarhus, Denmark.

Introduction

Asynchronous automata were introduced by Zielonka as a natural generalization of finite-state automata for concurrent systems [Zie]. An asynchronous automaton consists of a set of components, or processes, which periodically synchronize to process their input. Each letter a in the alphabet is associated with a subset $\theta(a)$ of processes. The processes in $\theta(a)$ synchronize on reading a and jointly decide on a move. The processes outside $\theta(a)$ remain unchanged during this move—in fact, they are oblivious to the occurrence of a.

A distributed alphabet of this type gives rise to an *independence relation* I between letters: $(a, b) \in I$ iff $\theta(a) \cap \theta(b) = \emptyset$. Thus a and b are independent when processed by disjoint sets of components.

An alphabet with an independence relation is also called a *concurrent alphabet*. This notion was introduced by Mazurkiewicz as a technique for studying concurrent systems from the viewpoint of formal language theory [Maz]. Given a concurrent alphabet (Σ, I), I induces a natural equivalence relation $\sim$ on Σ^*: two words w and w' are related by $\sim$ iff w' can be obtained from w by a sequence of permutations of adjacent independent letters. The equivalence class $[w]$ containing w is called a *trace*.

A language $L \subseteq \Sigma^*$ is said to be a *trace language* if L is closed under $\sim$—i.e., for each $w \in \Sigma^*$, w is in L iff every word in $[w]$ is L. A trace language is *recognizable* if it is accepted by a conventional finite state automaton.

However, since conventional automata are sequential, it is quite awkward to precisely characterize the class of automata which recognize trace languages. Asynchronous automata, on the other hand, are natural machines for recognizing these languages. If we distribute Σ in such a way that the induced independence relation is precisely I, we are guaranteed that the language accepted by the automaton is closed under $\sim$.

Despite this simple connection, it is hard to prove that the class of languages accepted by asynchronous automata coincides with the class of recognizable trace languages. This result was first established by Zielonka [Zie]. Given a conventional finite automaton recognizing a trace language over (Σ, I), he showed how to construct directly a *deterministic* asynchronous automaton over a distributed alphabet (Σ, θ) recognizing the same language, such that the independence relation induced by θ is precisely I. The proof involves combinatorics over partially commutative monoids and is quite difficult to grasp. A comprehensive survey of the theory of recognizable trace languages can be found in [Die].

Contributions of this paper

In this paper, we generalize the classic subset construction of Rabin and Scott in order to obtain a direct procedure for determinizing an asynchronous automaton $\mathfrak{A}$. To our knowledge, this construction is the first that involves only a double-exponential blow-up in the size of the state spaces. We also show that this bound is essentially optimal.

The only other known way to determinize a non-deterministic asynchronous automaton $\mathfrak{A}$ is to view it as a normal non-deterministic automaton at the level of "global states" and then apply Zielonka's construction to obtain a deterministic asynchronous automaton.

Our construction is the last piece needed for a "classical" proof of Zielonka's theorem that recognizable trace languages coincide with the languages accepted by (deterministic) asynchronous automata. Even before Zielonka's result, Ochmanski had defined a class of rational expressions that precisely generate recognizable trace languages [Och]. Ochmanski's characterization is effective—in particular, given a finite automaton recognizing a trace language we can construct a rational expression for the language. Recently, Pighizzini has shown how to construct, inductively, a non-deterministic automaton $\mathfrak{A}(E)$ for any (Ochmanski)-rational expression E such that $\mathfrak{A}(E)$ accepts exactly the rational subset of Σ^* associated with E [Pig]. Further, the size of $\mathfrak{A}(E)$ is polynomial in the length of E. Given our construction, we can now convert $\mathfrak{A}(E)$ into a deterministic asynchronous automaton $\mathfrak{B}(E)$ which also accepts the same set. This proof of Zielonka's theorem is analogous to the "textbook" proof that sets defined by regular expressions coincide with sets recognized by deterministic finite state automata [HU].

The paper is organized as follows. We begin by describing asynchronous automata and fixing some notation that we use in the paper. Then, we describe the distinction between local and global views of a word over a distributed alphabet. In Section 3 we define local runs and show how to characterize global runs of asynchronous automata as special products of local runs. The rest of the paper is devoted to showing how to maintain finite sets of local runs which retain all the necessary information about global runs. The crucial notion is that of a frontier, defined in Section 4. Section 5 introduces primary and secondary events, which between them subsume the frontiers. The "gossip automaton" of [MS], which locally updates primary and secondary information is described in the next section. All these notions are finally put together in Section 7 which presents the overall determinization construction. In Section 8 we analyze the complexity of our construction and provide a simple lower bound.

1 Preliminaries

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

State spaces With each process p, we associate a finite set of states denoted V_p. Each state in V_p is called a *local state*. For $P \subseteq \mathcal{P}$, we use V_P to denote the product $\prod_{p\in P} V_p$. An element $\vec{v}$ of V_P is called a *P-state*. A $\mathcal{P}$-state is also called a *global state*. Given $\vec{v} \in V_P$, and $P' \subseteq P$, we use $\vec{v}_{P'}$ to denote the projection of $\vec{v}$ onto $V_{P'}$. Also, $\vec{v}_{\overline{P'}}$ abbreviates $\vec{v}_{P\setminus P'}$. For a singleton $p \in P$, we write $\vec{v}_p$ rather than $\vec{v}_{\{p\}}$. For $a \in \Sigma$, we write V_a to mean $V_{\theta(a)}$ and $V_{\overline{a}}$ to mean $V_{\overline{\theta(a)}}$. Similarly, if $\vec{v} \in V_P$, we write $\vec{v}_a$ to denote $\vec{v}_{\theta(a)}$ and $\vec{v}_{\overline{a}}$ to denote $\vec{v}_{\overline{\theta(a)}}$.

Asynchronous automaton An *asynchronous automaton* $\mathfrak{A}$ over (Σ, θ) is of the form

$$(\{V_p\}_{p\in\mathcal{P}}, \{\to_a\}_{a\in\Sigma}, \mathcal{V}_0, \mathcal{V}_F),$$

where $\to_a \subseteq V_a \times V_a$ is the *local transition relation* for a, and $\mathcal{V}_0, \mathcal{V}_F \subseteq V_{\mathcal{P}}$ are sets of *initial* and *final* global states. Intuitively, each $\to_a$ specifies how the process $\theta(a)$ that meet on a may decide on a joint move. Other processes do not change their state. Thus we define the *global transition relation* $\Rightarrow \subseteq V_{\mathcal{P}} \times \Sigma \times V_{\mathcal{P}}$ by $\vec{v} \stackrel{a}{\Longrightarrow} \vec{v}'$ if $\vec{v}_a \to_a \vec{v}'_a$ and $\vec{v}_{\overline{a}} = \vec{v}'_{\overline{a}}$.

$\mathfrak{A}$ is called *deterministic* if the transition relation of $\mathfrak{A}$ is a function from $V_{\mathcal{P}} \times \Sigma$ to $V_{\mathcal{P}}$ and if the set of initial states $\mathcal{V}_0$ is a singleton.

Runs Let $u \in \Sigma^*$ be of length of m. It is convenient to think of u as a function $u : [1..m] \to \Sigma$, where for natural numbers $i < j$, $[i..j]$ abbreviates the set $\{i, i+1, \ldots, j\}$.

A *(global) run* of $\mathfrak{A}$ on u is a function $\rho : [0..m] \to V_{\mathcal{P}}$ such that $\rho(0) \in \mathcal{V}_0$ and for $i \in [1..m]$, $\rho(i-1) \overset{u(i)}{\Longrightarrow} \rho(i)$.

The word u is *accepted* by $\mathfrak{A}$ if there is a run ρ of $\mathfrak{A}$ on u such that $\rho(m) \in \mathcal{V}_F$. $L(\mathfrak{A})$, the language accepted by $\mathfrak{A}$, is the set of words u accepted by $\mathfrak{A}$.

The problem Given a non-deterministic asynchronous automaton $\mathfrak{A}$ over (Σ, θ), we shall construct a deterministic asynchronous automaton $\mathfrak{B}$ over (Σ, θ), such that $L(\mathfrak{A}) = L(\mathfrak{B})$.

2 Local and global views

Events Given $u : [1..m] \to \Sigma$, we associate a set of *events* $\mathcal{X}_u$. Each event x is of the form $(i, u(i))$, where $i \in [1..m]$. In addition, we define an *initial event* denoted 0. The initial event marks the beginning when all processes synchronize and agree on an initial global state. Usually, we will write $\mathcal{X}$ for $\mathcal{X}_u$. We write $p \in x$ to denote that $p \in \theta(u(i))$ when $x = (i, u(i))$; for $x = 0$, we define $p \in x$ to hold for all $p \in \mathcal{P}$. If $p \in x$, then we say that x is a *p-event*.

If $x = (i, a)$ is an event, then we may use x instead of a in abbreviations such as V_x, which stands for V_a, i.e., $V_{\theta(a)}$.

EXAMPLE: Consider the word $u = abcd$ over the alphabet (Σ, θ) for $\mathcal{P} = \{p, q, r, s\}$, where $\Sigma = \{a, b, c, d\}$ and $\theta(a) = \{p, q\}$, $\theta(b) = \{q, r, s\}$, $\theta(c) = \{r, s\}$, and $\theta(d) = \{p, q\}$. The set $\mathcal{X}_u$ of events is then

$$\{x_0, x_1, x_2, x_3, x_4\} = \{0, (1, a), (2, b), (3, c), (4, d)\}$$

.

Ordering relations on $\mathcal{X}$ A word u imposes a total order on events: define $x < y$ if $x \neq y$ and either $x = 0$ or $x = (i, u(i))$, $y = (j, u(j))$, and $i < j$. We write $x \leq y$ if $x = y$ or $x < y$. Moreover, each process p orders the events in which it participates: define $\lessdot_p$ to be the strict ordering

$$x \lessdot_p y \textit{ if } x < y, \ p \in x \cap y \text{ and for all } x < z < y, \ p \notin z.$$

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

Define $x \sqsubset y$ if for some p, $x \lessdot_p y$ and $x \sqsubseteq y$ if $x = y$ or $x \sqsubset y$. The *causality relation* $\sqsubseteq^*$ is the transitive closure of $\sqsubseteq$. If $x \sqsubseteq^* y$ then we say that x is *below* y. Note that 0 is below any event. The set of events below x is denoted $x\downarrow$. These represent the only synchronizations in $\mathcal{X}$ that may have affected the state of the processes in x when x occurs. The *neighbourhood* of x, $nbd(x)$, consists of x together with all its "$\sqsubset$-predecessors"—i.e., $nbd(x) = \{x\} \cup \{y \mid y \sqsubset x\}$.

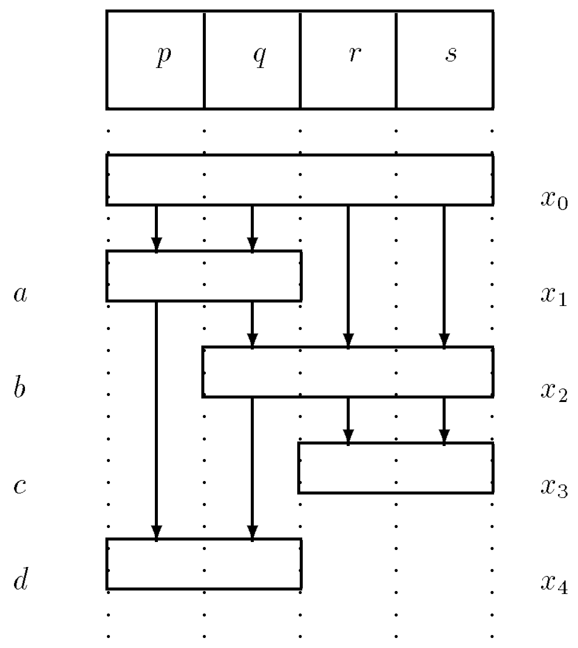

Figure 1: An example

EXAMPLE: Continuing our example, in Figure 1 an arrow has been drawn between each pair of events related by $\sqsubset$; the vertical dashed lines further partition these arrows into $\lhd_p$, $\lhd_q$, $\lhd_r$ and $\lhd_s$. For example, $x_0 \lhd_p x_1$ holds, but $x_0 \lhd_r x_1$ does not hold. $x_4\!\downarrow$, the x_4-cone, is the set $\{x_0, x_1, x_2, x_4\}$ and $x_3\!\downarrow$, the x_3-cone, is $\{x_0, x_1, x_2, x_3\}$. Thus $x_3\!\downarrow \cup\, x_4\!\downarrow = \mathcal{X}$. $nbd(x_4)$, the neighbourhood of x_4, is $\{x_1, x_2, x_4\}$.

Ideals A set of events $I \subseteq \mathcal{X}$ is called an *ideal* if I is closed with respect to $\sqsubseteq^*$—i.e., $x \in I$ and $y \sqsubseteq^* x$ implies $y \in I$ as well. Clearly the entire set $\mathcal{X}$ is an ideal, as is $x\!\downarrow$ for any $x \in \mathcal{X}$.

***P*-views** Let I be an ideal. 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 $P \subseteq \mathcal{P}$, the P-view of I, denoted $I|_P$, is $\bigcup_{p \in P} I|_p$. Notice that $I|_P$ is always an ideal. In particular, we have $I|_{\mathcal{P}} = I$. Also notice that if $y \in I|_P$ for some $P \subseteq \mathcal{P}$, then $nbd(y) \subseteq I|_P$ as well.

EXAMPLE: In the example of Figure 1, $max_q(\mathcal{X}) = x_4$. So $\mathcal{X}|_q = x_4\!\downarrow = \{x_4, x_2, x_1, x_0\}$. On the other hand, $max_s(\mathcal{X}) = x_3$ and $\mathcal{X}|_s = x_3\!\downarrow = \{x_3, x_2, x_1, x_0\}$.

3 Local runs and histories

Local runs Let I be an ideal. A *local run on I* is a function r that assigns to each $y \in I$ a y-state—i.e., a state in V_y—such that $r(0) \in \mathcal{V}_0$ and for all $y \neq 0$, r is consistent with $\rightarrow_y$ in the neighbourhood $nbd(y)$. In other words, for $y \neq 0$ we have $\vec{v} \rightarrow_y r(y)$, where $\vec{v}$

the y-state such that for all $q \in y$, $\vec{v}_q = r(z)_q$, where $z \lessdot_q y$. Let $\mathcal{R}(I)$ denote the set of all local runs on I.

So, a local run on $\mathcal{X}$ is an assignment of a y-state to each $y \in \mathcal{X}$ such that all neighbourhoods in $\mathcal{X}$ are consistently labelled.

Proposition 1 *Given $u : [1..m] \to \Sigma$, there is a 1-1 correspondence between local runs on $\mathcal{X}_u$ and global runs on u.*

Proof For $i \in [1..m]$ let w_i denote the prefix of u up to i. Also, for $i \in [1..m]$ let x_i denote the event $(i, u(i))$ in $\mathcal{X}_u$.

Given a local run r on $\mathcal{X}_u$, we define a global run $\rho : [0..m] \to V_{\mathcal{P}}$ on u by $\rho(0) = r(0)$ and for $i \in [1..m]$, $\rho(i)_p = r(max_p(\mathcal{X}_{w_i}))_p$ for all $p \in \mathcal{P}$.

Given a global run $\rho : [0..m] \to V_{\mathcal{P}}$ on u , we define a local run r on $\mathcal{X}_u$ by $r(0) = \rho(0)$ and for $i \in [1..m]$, $r(x_i) = \rho(i)_{\theta(u(i))}$. $\square$

Histories Let I be an ideal. A *history* on I is a partial function h such that $dom(h) \subseteq I$ and $h(y) \in V_y$ for each $y \in dom(h)$. A history h is *reachable* if there is some local run r on I such that $h(y) = r(y)$ for all $y \in dom(h)$. Let $\mathcal{H}(I)$ denote the set of all histories on I. Clearly, $\mathcal{R}(I) \subseteq \mathcal{H}(I)$.

Choices Let I be an ideal. Given a collection $\{H_p\}_{p \in P}$ of sets of histories on the p-views $I|_p$, a *P-choice* $\{h_p\}_{p \in P}$ of $\{H_p\}_{p \in P}$ assigns to each $p \in P$, a history h_p from H_p. The choice is *consistent* if for each $p, q \in P$, for every $y \in dom(h_p) \cap dom(h_q)$, $h_p(y) = h_q(y)$.

Let $\{H_p\}_{p \in P}$ be a collection of sets histories for $P \subseteq \mathcal{P}$. We define the product $\bigotimes_{p \in P} H_p$ as follows.

$$\bigotimes_{p \in P} H_p = \{h \in \mathcal{H}(I|_P) \mid \begin{array}{l} \textit{There exists a consistent } P\textit{-choice } \{h_p\}_{p \in P} \textit{ of} \\ \{H_p\}_{p \in P} \textit{ such that } dom(h) = \bigcup_p dom(h_p) \textit{ and} \\ \forall p \in P.\ \forall y \in dom(h_p).\ h(y) = h_p(y)\} \end{array}$$

So each element in $\bigotimes_{p \in P} H_p$ is a history on $I|_P$ pieced together from a choice of mutually consistent histories on $I|_p$ from the sets H_p, for $p \in P$.

In particular, for $P \subseteq \mathcal{P}$ we may form the product $\bigotimes_{p \in P} \mathcal{R}(I|_p)$, which generates the set $\mathcal{R}(I|_P)$ of local runs on $I|_P$:

Lemma 2 *Let I be an ideal and $P \subseteq \mathcal{P}$. $\mathcal{R}(I|_P) = \bigotimes_{p \in P} \mathcal{R}(I|_p)$*

Proof Let $r \in \mathcal{R}(I|_P)$. Then for each $p \in P$, clearly r restricted to $I|_p$ is a local run r_p on $I|_p$. So, using the consistent P-choice $\{r_p\}_{p \in P}$ of $\{\mathcal{R}(I|_p)\}_{p \in P}$ we get $r \in \bigotimes_{p \in P} \mathcal{R}(I|_p)$.

On the other hand, let $r \in \bigotimes_{p \in P} \mathcal{R}(I|_p)$ where the consistent P-choice is $\{r_p\}_{p \in P}$. Clearly $dom(r) = I|_P$. We just have to check that $nbd(y)$ is labelled consistently by r for each $y \in I|_P$. But, any $y \in I|_P$ must also belong to $I|_p$ for some $p \in P$. We know that $y \in I|_p$ implies $nbd(y) \subseteq I|_p$. Since r_p is a local run on $I|_p$, r_p must have assigned consistent values to $nbd(y)$. But r agrees with r_p on all events in $I|_p$, so r must assign the same values to $nbd(y)$ and we are done. $\square$

An infinite-state deterministic automaton

The preceding lemma tells us that we can reconstruct $\mathcal{R}(I|_{\mathcal{P}}) = \mathcal{R}(I)$ by taking the product $\bigotimes_{p\in\mathcal{P}} \mathcal{R}(I|_p)$—i.e., we can recover all the local runs on $\mathcal{X}$ by taking the product of local runs on $\mathcal{X}|_p$. We already know that local runs on $\mathcal{X}$ correspond precisely to the global runs on u.

This immediately gives us an infinite-state deterministic asynchronous automaton $\mathfrak{A}' = (\{V'_p\}_{p\in\mathcal{P}}, \{\rightarrow'_a\}_{a\in\Sigma}, \mathcal{V}'_0, \mathcal{V}'_F)$ which accepts $L(\mathfrak{A})$. For $p \in \mathcal{P}$, $V'_p = \{\mathcal{R}(\mathcal{X}_u|_p) \mid u \in \Sigma^*\}$. In other words, each local state of p consists of the set of local runs on $\mathcal{X}_u|_p$ for each word u.

Initially, each process starts off in the state $\{\langle 0 \mapsto \vec{v}\rangle \mid \vec{v} \in \mathcal{V}_0\}$. The global initial state $\mathcal{V}'_0$ is the cross product of these initial states.

The transition relations $\rightarrow'_a$ are defined in the natural way. Suppose $w = ua$. Let x_a be the new event associated with a—i.e., $\{x_a\} = \mathcal{X}_w \setminus \mathcal{X}_u$. Clearly, for $p \notin \theta(a)$, $\mathcal{X}_w|_p = \mathcal{X}_u|_p$ and, as desired, the local state of p does not change—$\mathcal{R}(\mathcal{X}_w|_p) = \mathcal{R}(\mathcal{X}_u|_p)$. For $p \in \theta(a)$, it is easy to check that $\mathcal{X}_w|_p = \mathcal{X}_u|_{\theta(a)} \cup \{x_a\}$. So, any run $r_p \in \mathcal{R}(\mathcal{X}_w|_p)$ consists of a local run on $\mathcal{X}_u|_{\theta(a)}$ together with an assignment of a $\theta(a)$-state to x_a which is consistent with $\rightarrow_a$ in the neighbourhood $nbd(x_a)$. But, by Lemma 2, $\mathcal{R}(\mathcal{X}_u|_{\theta(a)})$ is precisely the product $\bigotimes_{p\in\theta(a)} \mathcal{R}(\mathcal{X}_u|_p)$. So, when the processes in $\theta(a)$ synchronize, they can pool their information and compute for each $p \in \theta(a)$ the new state $\mathcal{R}(\mathcal{X}_w|_p)$. In fact, for all $p \in \theta(a)$, the new local state $\mathcal{R}(\mathcal{X}_w|_p)$ will be identical.

To decide whether to accept u, we have to check if $\mathfrak{A}$ could have been in an accepting global state after u. By Lemma 2 and Proposition 1, we can associate with each local run $r \in \bigotimes_{p\in\mathcal{P}} \mathcal{R}(\mathcal{X}_u|_p)$ a global run ρ on u. The global state $\vec{v}$ of $\mathfrak{A}$ after r is obtained by setting $\vec{v}_p = r(max_p(\mathcal{X}_u))_p$ for each $p \in \mathcal{P}$. Since $\bigotimes_{p\in P} \mathcal{R}(\mathcal{X}_u|_p)$ generates exactly the set of local runs on $\mathcal{X}_u$, we can compute all reachable global states after u in this manner. So, a global state in our new automaton is accepting if one of the global states it generates is an accepting state for $\mathfrak{A}$.

In the following, we formulate a finite-state version of the automaton above. We do not store the entire set $\mathcal{R}(\mathcal{X}_u|_p)$ in each p. Instead, at any ideal I, each p will keep track of the set of reachable histories on a special bounded subset of $I|_p$. This bounded subset will be such that the product of reachable histories across this subset will also be reachable. In this manner, we ensure that the processes retain enough global information about runs to compute exactly the reachable global states after I.

4 Finite histories and frontiers

Frontiers Let I be an ideal and $p, q, s \in \mathcal{P}$. We say that event y is an *s-sentry* for p with respect to q if $y \lessdot_s z$ for some $z \in I|_q \setminus (I|_p \cap I|_q)$. Thus it is an event known to p and q, but whose s-successor is known only to q. Notice that for some $s \in \mathcal{P}$, there may be no s-sentry for p with respect to q.

Define $\mathit{frontier}_{pq}(I)$ to be the set of all s-sentries which exist for p with respect to q. Notice that this definition is asymmetric—$\mathit{frontier}_{pq}(I) \neq \mathit{frontier}_{qp}(I)$.

EXAMPLE: In the example of Figure 1, $\mathcal{X}|_p \cap \mathcal{X}|_s = \{x_0, x_1, x_2\}$. $\mathit{frontier}_{ps}(\mathcal{X}) = \{x_2\}$, whereas $\mathit{frontier}_{sp}(\mathcal{X}) = \{x_1, x_2\}$. Notice that x_2 belongs to both frontiers—it is an r-sentry and an s-sentry in $\mathit{frontier}_{ps}(\mathcal{X})$ and a q-sentry in $\mathit{frontier}_{sp}(\mathcal{X})$.

As our example shows, an event $y \in frontier_{pq}(I)$ could simultaneously be an s-sentry for several different s. The following observation guarantees that $frontier_{pq}(I)$ is always a bounded set.

Lemma 3 *Let I be an ideal and $p, q \in \mathcal{P}$, For each $s \in \mathcal{P}$ there is at most one s-sentry $y \in frontier_{pq}(I)$.*

Proof Suppose not. Then we have $y, y' \in frontier_{pq}(I)$ and z, z' such that $y \lessdot_s z$ and $y' \lessdot_s z'$. We know that all events involving s are totally ordered by $\lessdot_s$. So either $y \lessdot_s^* y'$ or $y' \lessdot_s^* y$. Without loss of generality assume that $y \lessdot_s^* y'$. Then we must have $y \lessdot_s z \lessdot_s^* y' \lessdot_s z'$. Since $y' \in I|_p \cap I|_q$ and $z \sqsubseteq^* y'$, $z \in I|_p \cap I|_q$ as well. But this contradicts the assumption that $z \in I|_q \setminus (I|_p \cap I|_q)$. $\square$

For $P \subseteq \mathcal{P}$ and $p \in P$, the P-frontier of p at I is the set

$$\bigcup_{q \in P \setminus \{p\}} frontier_{pq}(I) \cup frontier_{qp}(I).$$

Lemma 4 *Let I be an ideal and $\{h_p\}_{p \in P}$ be a consistent P-choice of $\{\mathcal{H}(I|_p)\}_{p \in P}$ such that for each $p \in P$:*

- *h_p is reachable; and*
- *the P-frontier of p is included in $dom(h_p)$.*

Then $\bigotimes_{p \in P} h_p$ is a reachable history in $\mathcal{H}(I|_P)$.

Proof Let us order the processes in P as $p_1, p_2, \ldots, p_k$. For $i \in [1..k]$, let $P_i = \bigcup_{j \in [1..i]} p_j$. By assumption, for each p_i, h_{p_i} is a reachable history. So, we have a local run r_{p_i} on $I|_{p_i}$ which agrees with h_{p_i} on $dom(h_{p_i})$. To show that $h = \bigotimes_{p \in P} h_p$ is reachable, we must construct a local run r on $I|_P$ which agrees with h on $dom(h) = \bigcup_{p \in P} dom(h_p)$.

Define r as follows:

- For all $y \in I|_{p_1}$, $r(y) = r_{p_1}(y)$.
- For $i \in [2..k]$, for all $y \in I|_{p_i} \setminus I|_{P_{i-1}}$, $r(y) = r_{p_i}(y)$.

So, we "sweep across" $I|_P$ starting from $I|_{p_1}$ and ending at $I|_{p_k}$, assigning states according to $r_{p_1}, r_{p_2}, \ldots, r_{p_k}$ in k "stages" as we go along. Clearly $dom(r) = I|_P$ and r agrees with h on $dom(h)$. We have to show that r is a local run; i.e., we have to show that r is consistent with $\rightarrow_y$ across $nbd(y)$ for each $y \in I|_P$.

Let $y \in I|_P$. We know that $r(y)$ was assigned at some stage $i \in [1..k]$. Clearly, $y \in I|_{p_i}$ and so $nbd(y) \subseteq I|_{p_i}$ as well. If $nbd(y) \subseteq I|_{p_i} \setminus I|_{P_{i-1}}$, then all the events in $nbd(y)$ are assigned r values at stage i according to r_{p_i}. Since r_{p_i} is a local run on I, these values must be consistent with $\rightarrow_y$.

The crucial case is when some $z \in nbd(y)$ lies in $I|_{P_{i-1}}$ and so has already been assigned a value according to r_{p_j} for some $j \in [1..i{-}1]$. But then $z \in I|_{P_{i-1}} \cap I|_{p_i}$ which is the same as $\bigcup_{j \in [1..i-1]}(I|_{p_j} \cap I|_{p_i})$. In other words, for some p_j, $j \in [1..i{-}1]$, z belongs to $frontier_{p_jp_i}(I)$. So $z \in dom(h_{p_j}) \cap dom(h_{p_i})$, by assumption. Therefore, the value $r(z)$ must agree with $h_{p_j}(z) = h_{p_i}(z)$ and hence must agree with $r_{p_i}(z)$ as well. In other words, even though $z \in nbd(y)$ has already been assigned a value before stage i, the value agrees with r_{p_i}. So, effectively, $nbd(y)$ is assigned values as given by r_{p_i} and these must be consistent with $\rightarrow_y$ since r_{p_i} is a local run on I. $\square$

This is a finite version of Lemma 2 above. Suppose that at the end of a word u, each process p maintains all reachable histories on a finite (bounded) set of events spanning the $\mathcal{P}$-frontier of p in $\mathcal{X}_u$ together with the maximum p-event $max_p(\mathcal{X}_u)$. By the previous lemma, the product of these histories will generate all the reachable global states of $\mathfrak{A}$ after u. Since the $\mathcal{P}$-frontier of p in any ideal is a finite set, the set of all reachable histories that p has to keep track of is also finite. So, using a bounded amount of information in each process, we can reconstruct all possible global states of $\mathfrak{A}$ after u.

The problem now is with maintaining frontier information locally—i.e., how can a process p compute and locally update its frontier? This is done using slightly larger, but still bounded, sets of events called primary and secondary information, which between them subsume the frontier. It turns out that these sets can be updated locally with each synchronization between processes. These then will be the domains of the histories maintained by each process.

5 Primary and secondary information

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

The *primary information* of p after I, $primary_p(I)$, is the set $\{latest_{p\rightarrow q}(I)\}_{q\in\mathcal{P}}$. As usual, for $P \subseteq \mathcal{P}$, $primary_P(I) = \bigcup_{p\in P} primary_p(I)$.

Remark: Since $q \in 0 \in I|_p$ for all $q \in \mathcal{P}$, the set $\{y \in I|_p \mid q \in y\}$ is always nonempty. Since all q-events are linearly ordered by $\lhd_q$, the maximum q-event in $I|_p$ is well-defined. Notice that $latest_{p\rightarrow p}(I) = max_p(I)$.

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

Each event in $secondary_p(I)$ is of the form $latest_{q\rightarrow s}(latest_{p\rightarrow q}(I)\!\downarrow)$ for some $q, s \in \mathcal{P}$. This is the latest s-event which q knows about upto the event $latest_{p\rightarrow q}(I)$. We abbreviate $latest_{q\rightarrow s}(latest_{p\rightarrow q}(I)\!\downarrow)$ by $latest_{p\rightarrow q\rightarrow s}(I)$. Notice that each primary event $latest_{p\rightarrow q}(I)$ is also a secondary event $latest_{p\rightarrow p\rightarrow q}(I)$. In other words, $primary_p(I) \subseteq secondary_p(I)$.

Example: In Figure 1, $latest_{s\rightarrow p}(\mathcal{X}) = x_1$ whereas $latest_{p\rightarrow s}(\mathcal{X}) = x_2$. Also, $latest_{s\rightarrow p\rightarrow r}(\mathcal{X}) = x_0$ while $latest_{p\rightarrow s\rightarrow r}(\mathcal{X}) = x_2$.

Lemma 5 *Let I be an ideal, $p, q \in \mathcal{P}$ and $y \in frontier_{pq}(I)$ an s-sentry. Then $y = latest_{p\rightarrow s}(I)$. Also, for some $s' \in \mathcal{P}$, $y = latest_{q\rightarrow s'\rightarrow s}(I)$. So, $y \in primary_p(I) \cap secondary_q(I)$.*

Proof Since y is an s-sentry, for some $z \in I|_q \setminus I|_p$, $y \lessdot_s z$. Suppose that $latest_{p \to s}(I) = y' \neq y$. Since all s-events are linearly ordered by $\lessdot_s$, we must have $y \lessdot_s^* y'$. However, $y \lessdot_s z$ as well, so we have $y \lessdot_s z \lessdot_s^* y'$. This means that $z \in I|_p$, which is a contradiction.

Next, we must show that $y = latest_{q \to s' \to s}(I)$ for some $s' \in \mathcal{P}$. We know that there is a path $y \sqsubset z_1 \sqsubset \ldots \sqsubset max_p(I)$, since $y \in I|_p$. 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 maximum p-event in I, it must be the maximum p-event in $I|_q$. So, $y = latest_{q \to p \to s}(I)$ and we are done.

If this path does leave $I|_p \cap I|_q$, we can find an event $y' \in frontier_{qp}(I)$ along the path such that y' is an s'-sentry for some $s' \in \mathcal{P}$—in other words, for some z', $y \sqsubseteq^* y' \lessdot_{s'} z' \sqsubseteq^* max_p(I)$. We know by our earlier argument that $y' = latest_{q \to s'}(I)$. It must be the case that $y = latest_{s' \to s}(y'{\downarrow})$. For, if $latest_{s' \to s}(y'{\downarrow}) = y'' \neq y$, then $y \lessdot_s^* y'' \sqsubseteq^* y' \sqsubseteq^* max_p(I)$. This implies that $y \lessdot_s^* y''$ and $y'' \in I|_p$, which contradicts the fact that $y = latest_{p \to s}(I)$. So, $y = latest_{s' \to s}(y'{\downarrow}) = latest_{q \to s' \to s}(I)$ and we are done. $\square$

So, for every $p \in \mathcal{P}$ and $u \in \Sigma^*$, each process p maintains all reachable histories over the finite set $secondary_p(\mathcal{X}_u)$. (Recall that $primary_p(\mathcal{X}_u) \subseteq secondary_p(\mathcal{X}_u)$.) By the preceding lemma, this set includes all events in the $\mathcal{P}$-frontier of $\mathcal{X}_u$ as well as the maximal event $max_p(\mathcal{X}_u) = latest_{p \to p \to p}(\mathcal{X}_u)$.

We now need to show that these sets may be updated locally—i.e., if $w = ua$, then $secondary_p(w)$ may be computed from $secondary_p(u)$ for each process $p \in \theta(a)$ using only the information available with the processes in $\theta(a)$. This involves running the "gossip automaton" [MS] in the background. In order to make this presentation self-contained, we describe the procedure of [MS] for comparing and updating primary and secondary information.

Comparing primary information

Lemma 6 *Let I be an ideal and $p, q, s \in \mathcal{P}$. Let $y_p = latest_{p \to s}(I)$ and $y_q = latest_{q \to s}(I)$. Then $y_p \sqsubseteq^* y_q$ iff $y_p \in secondary_q(I)$.*

Proof

($\Leftarrow$) Suppose $y_p \in secondary_q(I)$. Then, $y_p \in I|_q$ and so $y_p \sqsubseteq^* y_q \in I|_q$ by the definition of $latest_{q \to s}(I)$.

($\Rightarrow$) If $y_p = y_q$, $y_p \in primary_q(I) \subseteq secondary_q(I)$ and there is nothing to prove. If $y_p \neq y_q$, then, $y_p \lessdot_s^* y_q$ and so $y_p \in I|_p \cap I|_q$. Let y' be the s-successor of y. We know that $y' \in I|_q \setminus I|_p$, so y_p is an s-sentry in $frontier_{pq}(I)$. But then, by our previous lemma, $y_p \in primary_p(I) \cap secondary_q(I)$ and we are done. $\square$

Suppose p and q synchronize at an action a after u. At this point they "share" their primary and secondary information. If q can find the event $latest_{p \to s}(\mathcal{X}_u)$ in its set of secondary events $secondary_q(\mathcal{X}_u)$, q knows that its latest s-event $latest_{q \to s}(\mathcal{X}_u)$ is at least as recent as $latest_{p \to s}(\mathcal{X}_u)$. So, after the synchronization, $latest_{q \to s}(\mathcal{X}_{ua})$ is the same as $latest_{q \to s}(\mathcal{X}_u)$, whereas p inherits this information from q—i.e., $latest_{p \to s}(\mathcal{X}_{ua}) = latest_{q \to s}(\mathcal{X}_u)$. In this way, for each $s \in \mathcal{P}$, p and q can locally update their primary information about s in $\mathcal{X}_{ua}$. Clearly $latest_{p\to q}(\mathcal{X}_{ua}) = latest_{q\to p}(\mathcal{X}_{ua}) = x_a$, where x_a is the new event—i.e., $\mathcal{X}_{ua} \setminus \mathcal{X}_u = \{x_a\}$.

This procedure generalizes to any arbitrary set $P \subseteq \mathcal{P}$ which synchronize after u. The processes in P share their primary and secondary information and compare this information pairwise. Using Lemma 6, for each $q \in \mathcal{P} \setminus P$ they decide who has the "latest information" about q. Each process then comes away with the best primary information from P. Notice, that all processes in P will always have the *same* primary information after they synchronize.

Once we have compared primary information, updating secondary information is automatic. Clearly, if $latest_{q\to s}(I)$ is better than $latest_{p\to s}(I)$, then every secondary event $latest_{q\to s\to s'}(I)$ must also be better than $latest_{p\to s\to s'}(I)$. So, secondary information can be locally updated too. In other words, to consistently update primary and secondary information, it suffices to to correctly compare primary information, which is achieved by Lemma 6.

From the preceding argument, it is clear that each new event belongs to the primary (and hence secondary) information of the processes which synchronize at that event. Further, 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 7 *Let $u, w \in \Sigma^*$ such that $w = ua$ for some $a \in \Sigma$. Let x_a denote the new event in w—i.e., $\mathcal{X}_w \setminus \mathcal{X}_u = \{x_a\}$. Then:*

- $x_a \in secondary_{\mathcal{P}}(\mathcal{X}_w)$.
- $secondary_{\mathcal{P}}(\mathcal{X}_w) \subseteq \{x_a\} \cup secondary_{\mathcal{P}}(\mathcal{X}_u)$.

6 Locally updating primary/secondary information

To make Lemma 6 effective, we must make the assertions "locally checkable"—e.g., if $y_p = latest_{p\to s}(I)$, processes p and q must be able to decide if $y_p \in secondary_q(I)$. This is achieved by labelling each action in u in such a way that primary and secondary information can be maintained as sets of labelled actions.

We may naïvely assume that events in $\mathcal{X}_u$ are locally assigned distinct labels—in effect, at each action a, the processes in $\theta(a)$ assign a time-stamp to the new occurrence of a. In this manner, the processes in $\mathcal{P}$ can easily assign consistent local time-stamps for each action which will let them compute the relations between events which we are interested in.

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 would mean that several different occurrences of the same action will eventually end up with the same label. We have to ensure that this does not lead to any confusion when we try to update primary and secondary information.

However, from Lemma 6, we know that to compare primary information, we only need to look at the events 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 primary or secondary information of different processes, the corresponding event is actually the same.

Suppose we have such a labelling on u and we want to extend this to a labelling on $w = ua$—i.e., we need to assign a label to the new a-event. By Proposition 7, it suffices to use a label which is distinct from the labels of all the a-events currently in the secondary information of $\mathcal{X}_u$.

Unfortunately, the processes in $\theta(a)$ cannot directly see all the a-events which belong to the secondary information of the entire system. An a-event y may be part of the secondary information of processes *outside* $\theta(a)$—i.e., $y \in \mathit{secondary}_{\overline{a}}(\mathcal{X}_u) \setminus \mathit{secondary}_a(\mathcal{X}_u)$. To enable the processes in $\theta(a)$ to know about all a events in $\mathit{secondary}_{\mathcal{P}}(\mathcal{X}_u)$, we need to maintain tertiary information.

Tertiary information The *tertiary information* of p after I, $\mathit{tertiary}_p(I)$, is the set $\bigcup_{q \in \mathcal{P}} \mathit{secondary}_q(\mathit{latest}_{p \to q}(I) \downarrow)$. In other words, this is the latest information that p has in I about the secondary information of q, for all $q \in \mathcal{P}$. As before, for $P \subseteq \mathcal{P}$, $\mathit{tertiary}_P(I) = \bigcup_{p \in P} \mathit{tertiary}_p(I)$.

Each event in $\mathit{tertiary}_p(I)$ is of the form $\mathit{latest}_{q \to s \to s'}(\mathit{latest}_{p \to q}(I)\!\downarrow)$ for some $q, s, s' \in \mathcal{P}$. We abbreviate $\mathit{latest}_{q \to s \to s'}(\mathit{latest}_{p \to q}(I)\!\downarrow)$ by $\mathit{latest}_{p \to q \to s \to s'}(I)$. Just as $\mathit{primary}_p(I) \subseteq \mathit{secondary}_p(I)$, clearly $\mathit{secondary}_p(I) \subseteq \mathit{tertiary}_p(I)$ since each secondary event $\mathit{latest}_{p \to q \to s}(I)$ is also a tertiary event $\mathit{latest}_{p \to p \to q \to s}(I)$.

Lemma 8 *Let I be an ideal and $p \in \mathcal{P}$. If $y \in \mathit{secondary}_p(I)$ then for every $q \in y$, $y \in \mathit{tertiary}_q(I)$.*

Proof Let $y \in \mathit{secondary}_p(I)$ and $q \in y$. We know that $y \in I|_p \cap I|_q$ and there is a path $y \sqsubset z_1 \sqsubset \ldots \sqsubset \mathit{max}_p(I)$ leading from y to $\mathit{max}_p(I)$.

Suppose this path never leaves $I|_p \cap I|_q$. Then $\mathit{max}_p(I) \in I|_q$ and so $\mathit{max}_p(I) = \mathit{latest}_{q \to p}(I)$. This means that $y \in \mathit{secondary}_p(\mathit{latest}_{q \to p}(I)\!\downarrow) \subseteq \mathit{tertiary}_q(I)$ and we are done.

Otherwise, the path from y to $\mathit{max}_p(I)$ does leave $I|_p \cap I|_q$ at some stage. Concretely, let $y = \mathit{latest}_{p \to p' \to p''}(I)$ for some $p', p'' \in \mathcal{P}$. So the path from y to $\mathit{max}_p(I)$ passes through $y' = \mathit{latest}_{p \to p'}(I)$.

If $y' \notin I|_p \cap I|_q$ then for some $z, z' \in \mathcal{X}$ and some $s \in \mathcal{P}$ we have $z \in I|_p \cap I|_q$, $z' \in I|_p \setminus I|_q$ and $y \sqsubseteq^* z \lessdot_s z' \sqsubseteq^* y'$. This means that $z \in \mathit{frontier}_{qp}(I)$ is an s-sentry and by our earlier argument we know that $z = \mathit{latest}_{q \to s}(I)$. So $y = \mathit{latest}_{q \to s \to p''}(I) = \mathit{latest}_{q \to q \to s \to p''}(I) \in \mathit{tertiary}_q(I)$.

On the other hand, if $y' \in I|_p \cap I|_q$ we can find an s-sentry $z \in \mathit{frontier}_{qp}(I)$ on the path from y' to $\mathit{max}_p(I)$, for some $s \in \mathcal{P}$. We once again get $z = \mathit{latest}_{q \to s}(I)$ and so $y = \mathit{latest}_{q \to s \to p' \to p''}(I) \in \mathit{tertiary}_q(I)$.

□

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., consistently to update primary, secondary and tertiary information whenever a set of processes synchronize.

Each process maintains sets of primary, secondary and tertiary information. Each event in these sets is represented by a pair $\langle P, \ell \rangle$, where P is the subset of processes that synchronized at the event and $\ell \in \mathcal{L}$, a finite set of labels.

By Lemma 8, each p-event that appears in some primary or secondary set in the system also appears in the tertiary information of p. When a new event x occurs after u, the processes participating in x assign a label to this event which does not appear in $\mathit{tertiary}_x(\mathcal{X}_u)$. Proposition 7 guarantees that the new event is assigned a label which is distinct from those assigned to $\mathit{secondary}_{\mathcal{P}}(\mathcal{X}_u)$. Since each process keeps track of N^3 tertiary events, there need be only $O(N^3)$ labels in $\mathcal{L}$.

The processes participating in x now compare their primary information about each process $s \notin x$ by checking labels of events across their primary and secondary sets. Each process then updates its primary, secondary and tertiary sets according to the new information it receives. (Notice that tertiary information, like secondary information, can be locally updated once the processes have decided who has the best primary information.)

To implement this algorithm as a deterministic asynchronous automaton, we just observe that each local state of p will consist of its primary, secondary and tertiary information for p, stored as an collection of indexed labels. The initial state is the global state where for all processes p, these sets all contain only the initial event 0. The local transition relations $\rightarrow_a$ modify the local states for processes in $\theta(a)$ as described above. This automaton does not have any final states—it simply runs in the "background".

7 The determinization algorithm

We are now ready to present our deterministic asynchronous automaton

$$\mathfrak{B} = (\{V_p^{\mathfrak{B}}\}_{p\in\mathcal{P}}, \{\rightarrow_a^{\mathfrak{B}}\}_{a\in\Sigma}, \mathcal{V}_0^{\mathfrak{B}}, \mathcal{V}_F^{\mathfrak{B}})$$

corresponding to our original non-deterministic asynchronous automaton $\mathfrak{A}$ such that $L(\mathfrak{A}) = L(\mathfrak{B})$.

Let us assume we have a sufficiently large but finite set of labels $\mathcal{L}$. Formally, a state in $V_p^{\mathfrak{B}}$ consists of the following information:

- A labelling $\lambda_p : (\mathcal{P} \times \mathcal{P} \times \mathcal{P}) \rightarrow (\mathcal{P} \times \mathcal{L})$, which is partially injective, in that for each $q, q', r, r' \in \mathcal{P}$, $\lambda_p(q, q, r) = \lambda_p(q', q', r')$ implies $q = q'$ and $r = r'$.
- A set of histories $\mathcal{RH}_p$ where each $h \in \mathcal{RH}_p$ is a function from $\mathcal{P} \times \mathcal{P}$ to P-states, $P \subseteq \mathcal{P}$.

Intuitively, after reading a word u, the automaton represents the tertiary event $\mathit{latest}_{p\rightarrow q\rightarrow r\rightarrow s}(\mathcal{X}_u)$ as $\lambda_p(q, r, s)$.

In this representation, several copies exist of each primary and secondary event. For instance, the primary event $\mathit{latest}_{p\rightarrow q}(\mathcal{X}_u)$ corresponds to $\lambda_p(q, q, q)$, $\lambda_p(p, q, q)$ and $\lambda_p(p, p, q)$. We choose canonical representatives for each primary and secondary event. So, we shall regard $\lambda_p(q, q, q)$ as the label of the primary event $\mathit{latest}_{p\rightarrow q}(\mathcal{X}_u)$ and $\lambda_p(q, q, r)$ as the label of the secondary event $\mathit{latest}_{q\rightarrow r}(\mathcal{X}_u)$ as $\lambda_p(q, q, r)$. The partially injectivity of λ_p ensures that all labels assigned to primary and secondary events are distinct.

The set $\mathcal{RH}_p$ is supposed to contain all reachable histories over the secondary events $secondary_p(\mathcal{X}_u)$. Since these events are injectively labelled by λ_p, we can also view $\mathcal{RH}_p$ as a function from $\mathcal{P} \times \mathcal{L}$ to P-states.

Initially, each $p \in \mathcal{P}$ stores the following:

- $\forall \langle q, r, s\rangle \in \mathcal{P}^3.\ \lambda_p(q, r, s) = \langle \mathcal{P}, 0\rangle$, where $0 \in \mathcal{L}$ is some arbitrary but fixed label.
- For each $\vec{v} \in \mathcal{V}_0$, the set of initial states of $\mathfrak{A}$, we have a history $h \in \mathcal{RH}_p$ such that $h(q, r) = \vec{v}$ for all $\langle q, r\rangle \in \mathcal{P}^2$.

The initial state $\mathcal{V}_0^{\mathfrak{B}}$ of $\mathfrak{B}$ is the product of the initial states of all $p \in \mathcal{P}$. The transition rule $\rightarrow_a^{\mathfrak{B}}$ is described in the following. Suppose $\mathfrak{B}$ reads a when the global state of $\mathfrak{B}$ is $\{\langle \lambda_p, \mathcal{RH}_p\rangle\}_{p\in\mathcal{P}}$. Then we have the following procedure for updating the local states of processes in $\theta(a)$.

- For each $p \in \theta(a)$, we construct a new labelling function $\lambda'_p : \mathcal{P}^3 \rightarrow \mathcal{P} \times \mathcal{L}$.

  Fix a new label $\ell \in \mathcal{L}$ such that $\langle \theta(a), \ell\rangle$ is not in the range of λ_p for any $p \in \theta(a)$. For each $p \in \theta(a)$, assign $\lambda'_p(q, r, s) = \langle \theta(a), \ell\rangle$ for all $q, r, s \in \mathcal{P}$ such that $\{q, r, s\} \subseteq \theta(a)$.

  The other values of λ'_p for each $p \in \theta(a)$ are computed as they would be by the gossip automaton. In other words, for all $p \in \theta(a)$, for all $q, r, s \in \mathcal{P}$ such that $\{q, r, s\} \not\subseteq \theta(a)$, the new value $\lambda'_p(q, r, s)$ is copied from the old value $\lambda_{p'}(q, r, s)$ assigned by the process $p' \in \theta(a)$ which had the best primary information $\lambda_{p'}(q, q, q)$.

- Compute new histories $\mathcal{RH}'_p$ for each $p \in \theta(a)$ as follows. Consider $h_a \in \bigotimes_{p\in\theta(a)}\{\mathcal{RH}_p\}$. Let $\vec{v}$ be the global a-state corresponding to h_a—i.e., $\vec{v}_p = h_p(p, p)$ for each $p \in \theta(a)$. Let $\mathcal{V}_{h_a} = \{\vec{v}' \mid \vec{v} \rightarrow_a \vec{v}'\}$. So, $\mathcal{V}_{h_a}$ is the set of all possible a-states v' which can be used to extend h_a to cover the new event x_a so that $nbd(x_a)$ is consistently labelled with respect to $\rightarrow_a$.

  Now each element $\vec{v}' \in \mathcal{V}_{h_a}$ together with h_a generates a history h'_p in $\mathcal{RH}'_p$ as follows:

$$\forall \langle q, r\rangle \in \mathcal{P}^2.\ h'_p(q, r) = \begin{cases} \vec{v}' & \textit{if } \lambda'_p(q, q, r) = \langle \theta(a), \ell\rangle \\ h_a(\lambda'_p(q, q, r)) & \textit{otherwise} \end{cases}$$

  So, the new a-event is assigned the a-tuple $\vec{v}'$ while the other secondary events of p (after reading a) inherit their h'_p values from h_a.

  Repeat this procedure for each $h_a \in \bigotimes_{p\in\theta(a)}\{\mathcal{RH}_p\}$ to generate the entire set $\mathcal{RH}'_p$ for each $p \in \theta(a)$.

Having described the transition functions $\rightarrow_a^{\mathfrak{B}}$, we now need to define the final states of $\mathfrak{B}$. Let $\vec{\sigma}$ be a global state of $\mathfrak{B}$, where $\vec{\sigma}_p = \langle \lambda_p, \mathcal{RH}_p\rangle$ for each $p \in \mathcal{P}$. Each $h \in \bigotimes_{p\in\mathcal{P}} \mathcal{RH}_p$ gives rise to a global state $\vec{v}$ of $\mathfrak{A}$ as follows: for each $p \in \mathcal{P}$, $\vec{v}_p = h(\lambda_p(p, p, p))$. Let $subset(\vec{\sigma})$ denote the set of global states of $\mathfrak{A}$ generated from $\vec{\sigma}$ in this manner. Then we can define

$$\mathcal{V}_F^{\mathfrak{B}} = \{\vec{\sigma} \mid subset(\vec{\sigma}) \cap \mathcal{V}_F \neq \emptyset\}.$$

Theorem 9 $L(\mathfrak{A}) = L(\mathfrak{B})$.

Proof For $u \in \Sigma^*$, let $\vec{\sigma}$ be the global state of $\mathfrak{B}$ after reading u such that $\vec{\sigma}_p = \langle \lambda'_p, \mathcal{RH}'_p\rangle$ for each $p \in \mathcal{P}$. We claim the following:

Claim For each $p \in \mathcal{P}$, λ'_p labels precisely the events in $tertiary_p(\mathcal{X}_u)$ and $\mathcal{RH}'_p$ is precisely the set of all reachable histories on the set of events $secondary_p(\mathcal{X}_u)$.

Assuming the claim, we know from Proposition 1 and Lemmas 2, 4 and 5 that the global states in $subset(\vec{\sigma})$ are precisely the global states that $\mathfrak{A}$ could be in after u. So, $\mathfrak{B}$ accepts u iff $subset(\vec{\sigma}) \cap \mathcal{V}_F \neq \emptyset$ iff there is a run of $\mathfrak{A}$ on u leading to a final state iff $\mathfrak{A}$ accepts u and we are done.

Proof of Claim To prove the claim, we proceed by induction on $|u|$.

The base case is when $u = \varepsilon$, the empty word. The claim is trivially true at this state since all the tertiary events in $\mathcal{X}_u$ are the initial event 0 and each process maintains a set of histories which assign all possible initial states of $\mathfrak{A}$ to the initial event.

Suppose $u = wa$ and, inductively, after reading w, the local state $\langle \lambda_p, \mathcal{RH}_p \rangle$ for each $p \in \mathcal{P}$ satisfies the Claim. We have to argue that the procedure for updating the local states of $p \in \theta(a)$ maintains the property asserted in the Claim.

Look at the definition of $\rightarrow_a^{\mathfrak{B}}$. Proposition 7 and Lemma 8 guarantee that the label $\langle \theta(a), \ell \rangle$ we assign to the new event does not clash with any labels already assigned to events in $secondary_{\mathcal{P}}(\mathcal{X}_w)$. Lemma 6 then ensures that the computation of λ'_p from λ_p is correct—i.e., λ'_p labels precisely the events in $tertiary_p(\mathcal{X}_u)$.

Now, assume $\mathcal{RH}_p$ contains all reachable histories over $secondary_p(\mathcal{X}_w)$ for each $p \in \theta(a)$. We have to show that $\mathcal{RH}'_p$ contains all reachable histories over $secondary_p(\mathcal{X}_u)$. For all $p \in \theta(a)$, $\mathcal{X}_u|_p = \mathcal{X}_w|_{\theta(a)} \cup \{x_a\}$, where x_a is the new a-event. So, any local run on $\mathcal{X}_u|_p$ consists of a local run on $\mathcal{X}_w|_{\theta(a)}$ extended to cover x_a such that $nbd(x_a)$ is consistently labelled.

We argue that the product $\bigotimes_{p\in\theta(a)} \mathcal{RH}_p$ is precisely the projection of $\mathcal{R}(\mathcal{X}_w|_{\theta(a)})$ onto $secondary_{\theta(a)}(\mathcal{X}_w)$. By Lemma 4, every history $h \in \bigotimes_{p\in\theta(a)} \mathcal{RH}_p$ is a reachable history on $\mathcal{X}_w|_{\theta(a)}$ and so is the projection of some local run on $\mathcal{X}_w|_{\theta(a)}$ onto $secondary_{\theta(a)}(\mathcal{X}|_w)$.

Conversely, consider any local run r on $\mathcal{X}_w|_{\theta(a)}$. Decompose r into local runs r_p over $\mathcal{X}_w|_p$ for each $p \in \theta(a)$ by looking at r restricted to $\mathcal{X}_w|_p$. Since $\mathcal{RH}_p$ has all reachable histories on $secondary_p(\mathcal{X}_w)$, the projection h_p of r_p on $secondary_p(\mathcal{X}_w)$ belongs to $\mathcal{RH}_p$. So, the projection of r onto $secondary_{\theta(a)}(\mathcal{X}_w)$ belongs to $\bigotimes_{p\in\theta(a)} \mathcal{RH}_p$.

So, we can reconstruct all possible a-moves of $\mathfrak{A}$ after w by looking at $\bigotimes_{p\in\theta(a)} \mathcal{RH}_p$. The procedure for updating $\mathcal{RH}_p$ to $\mathcal{RH}'_p$ in the definition of $\rightarrow_p^{\mathfrak{B}}$ then guarantees that $\mathcal{RH}'_p$ contains all reachable p-histories over $secondary_p(\mathcal{X}_u)$ for each $p \in \theta(a)$.

$\square$

8 The complexity of determinization

Analysis of the construction

Theorem 10 *Let $\mathfrak{A} = (\{V_p\}_{p\in\mathcal{P}}, \{\rightarrow_a\}_{a\in\Sigma}, \mathcal{V}_0, \mathcal{V}_F)$ be a non-deterministic asynchronous automaton with N processes such that $max_{p\in\mathcal{P}}|V_p| = M$. Then, in the corresponding deterministic automaton $\mathfrak{B}$ that we construct, each process has at most $2^{M^{O(N^3)}}$ states.*

Proof Each local state of $\mathfrak{B}$ is of the form $\langle \lambda_p, \mathcal{RH}_p \rangle$. We have already argued that we can maintain labels with a set $\mathcal{L}$ of size $O(N^3)$. So, each entry $\langle P, \ell \rangle$ in λ_p requires $O(N)$

bits to write down P and $O(\log N)$ bits to write down ℓ—i.e., $O(N)$ bits in all. Since there are N^3 entries in λ_p, all of λ_p may be written down using $O(N^4)$ bits.

We now need to maintain histories over secondary events. Each history h consists of N^2 P-states. Since a P-state can be written down using $N \log M$ bits, h can be written down using $N^3 \log M$ bits. The number of different histories possible over N^2 is $\left(M^{O(N)}\right)^{N^2} = M^{O(N^3)}$—each event could be assigned an arbitrary P-state and the number of distinct P-states is bounded by $\sum_{i \in [1..N]} M^i$ which is $M^{O(N)}$. So, $\mathcal{RH}_p$ can be written down using $M^{O(N^3)} \cdot N^3 \log M$ bits.

So, overall, each local state of $\mathfrak{B}$ can be written down using $M^{O(N^3)}$ bits. Therefore, the number of local states of each process in $\mathfrak{B}$ is bounded by $2^{M^{O(N^3)}}$. $\square$

In certain cases, we can slightly improve the estimate given above. Let T be the size of the transition relation of $\mathfrak{A}$—i.e., $T = \sum_{a \in \Sigma} \mid \rightarrow_a \mid$ where $\mid \rightarrow_a \mid$, the size of $\rightarrow_a$, is just the number of pairs in the relation $\rightarrow_a$. Then, we know that the P-states assigned by each history h must elements of $\rightarrow_a$ for some $a \in \Sigma$. So, the number of P-states we can assign is bounded by T. In general, T and M^N are not directly related, so a more accurate bound for the number of histories possible over N^2 nodes is $max(T, M^{O(N)})^{N^2}$. Therefore, the number of local states of a process in $\mathfrak{B}$ is bounded by $2^{max(T^{N^2}, M^{O(N^3)})}$.

Unlike conventional finite state automata, where determinization results in an exponential blowup in the number of states, our algorithm exhibits a superexponential blowup (at the level of *local* states). A simple argument shows that this cannot be avoided, in general.

A superexponential lower bound for determinization

Theorem 11 *There is a sequence of languages L_{K^N} over distributed alphabets $(\Sigma_{K^N}, \theta_{K^N})$, $K, N \geq 2$, such that L_{K^N} is recognized by a non-deterministic asynchronous automaton whose local state spaces and transition relations are polynomial in size as functions of K and N, whereas L_{K^N} cannot be recognized by a deterministic asynchronous automaton unless it has at least one process with $2^{K^N/N}$ states.*

Proof For $\Sigma = \{a, b\}$, let L^1_m be the set of words whose m^{th} letter from the right is a b. Recall that L^1_m can be recognized by an NFA with $O(m)$ states whereas a DFA requires 2^m states to recognize this language. We generalize L^1_m to L^k_m for $k \geq 1$—L^k_m is the set of words whose km^{th} last letter is a b. Let $L_m = \bigcup_{k \geq 1} L^k_m$. It is not difficult to see that L_m is also regular and the exponential separation between NFAs and DFAs recognizing L^1_m continues to hold for L_m as well.

Consider L_{K^N}—i.e., L_m, where $m = K^N$—for some $K, N \geq 2$. We look at a variant of L_{K^N} which we call L'_{K^N}. We show that L'_{K^N} can be recognized by a small non-deterministic asynchronous automaton—the states of each component will be quadratic in K and the transition relation will be polynomial in K and N. On the other hand, it will turn out that the smallest deterministic asynchronous automaton recognizing this language has at least one process with $O(2^{K^N/N})$ states.

The idea is to implement a N-digit counter to the base K using N processes named $[1..N]$. When the counter value is m, process i holds the value of the i^{th} digit in the base K representation of m.

To count efficiently using asynchronous automata, we need to introduce *carry letters* $\{c_i\}_{i\in[1..N]}$ into our alphabet. So, c_1 corresponds to a carry at the least significant digit, whereas c_N corresponds to an "overflow" carry at the most significant digit of our counter. Given a word $w \in \Sigma^*$, we intersperse the carry bits as follows: every K^{th} letter from Σ is immediately followed by a c_1 and every K^{th} c_i is immediately followed by a c_{i+1} for each $i \in [1..N-1]$. We call the new string $\mathsf{C}(w)$, the *carry-extension of* w.

Let $\Sigma' = \{a,b\} \cup \{c_i\}_{i\in[1..N]}$. Let θ' be a distribution of Σ' such that $\theta'(a) = \theta'(b) = \{1\}$, $\theta'(c_N) = \{N\}$ and $\theta'(c_i) = \{i, i+1\}$ for $i \in [1..N-1]$. This introduces a natural independence relation over Σ' as we have already seen. As described in the Introduction, this independence relation can be lifted to finite words in the obvious way—$w \sim w'$ iff w' can be obtained from w by a finite sequence of permutations of adjacent independent letters. It is easy to check that $\sim$ is an equivalence relation. The equivalence classes induced by $\sim$ are usually called *traces* [Maz]. Let $[w]$ denote the trace generated by a word $w \in \Sigma'^*$.

The language we will work with is the set of traces generated by carry-extended words from L_{K^N}. Formally, we shall look at $L'_{K^N} = \{w \in \Sigma'^* \mid \exists w' \in L''_{KN}.\ w \in [w']\}$, where $L''_{KN} = \{\mathsf{C}(w) \mid w \in L_{K^N}\}$.

A deterministic asynchronous automaton recognizing L'_{KN} must have at least one process with a state space of size $2^{K^N/N}$. To see this, consider a normal DFA recognizing L'_{K^N}. Let u and u' be two distinct words over Σ of length K^N. Without loss of generality, we can assume that there is a position i such that $u(i) = b$ and $u'(i) = a$. Then, there is a word v over Σ of length at most K^N such that $uv \in L_{K^N}$ and $u'v \notin L_{K^N}$.

Let w be the string such that $\mathsf{C}(u)w = \mathsf{C}(uv)$. Clearly $\mathsf{C}(u)w = \mathsf{C}(uv) \in L'_{K^N}$. Notice that $w \neq \mathsf{C}(v)$ in general. However, it is the case that $w\!\downarrow_{\{a,b\}}$—the string obtained by erasing all letters other than a and b from w—is just v. Now consider $\mathsf{C}(u')w$—there are two cases to look at. If $\mathsf{C}(u')w$ is not trace equivalent to any valid carry-extended word, then clearly $\mathsf{C}(u')w$ does not belong to L'_{K^N}. On the other hand, if $\mathsf{C}(u')w$ is trace equivalent to some carry-extended word w', $w'\!\downarrow_{\{a,b\}}$ must be equal to $\mathsf{C}(u')\!\downarrow_{\{a,b\}}$ concatenated with $w\!\downarrow_{\{a,b\}}$, since a and b are not independent in (Σ', θ'). But $\mathsf{C}(u')\!\downarrow_{\{a,b\}}\ w\!\downarrow_{\{a,b\}} = u'v$. Since $u'v \notin L_{K^N}$, $\mathsf{C}(u'v) \notin L''_{K^N}$ and so $\mathsf{C}(u')w \notin L'_{K^N}$ either.

So the canonical right invariant equivalence relation $R_{L'_{KN}}$ generated by L'_{KN} has at least 2^{K^N} equivalence classes. By the Myhill-Nerode Theorem, the minimal DFA recognizing L'_{KN} has at least 2^{K^N} states. Thus any deterministic asynchronous automaton recognizing L'_{K^N} must have at least one component with $\sqrt[N]{2^{K^N}} = 2^{K^N/N}$ states.

We now describe a small non-deterministic asynchronous automaton $\mathfrak{A}$ accepting L'_{K^N}. $\mathfrak{A}$ keeps counting letters in its input, using the carry letters to increment higher order digits of the counter. At some point, on reading a b, $\mathfrak{A}$ non-deterministically decides to copy its current counter value into a register. Meanwhile, $\mathfrak{A}$ continues to count letters from where it left off—it does not restart its counter when it sets the register. At the end of its input, it checks to see if the current counter value is the same as the one saved in the register. If so, the number of letters read after the b is a multiple of K^N and the input is accepted.

For each process i, the set of local states V_i is given by $[0..K] \times [0..K-1] \times \{N, Y\}$. The first component of the state is a digit in the running counter—the value K indicates a pending carry. The second component represents a "frozen" register value. The third component indicates whether or not the register value has been loaded. Initially, each process is in the state $\langle 0, 0, N\rangle$. The final states are those where each process is in a state of the form $\langle j, j, Y\rangle$ where $j \in [0..K-1]$—different processes could have different values of j.

The transition relations are as follows:

- $\rightarrow_a$: (Affects only process 1)
  $\langle i, 0, N\rangle \rightarrow_a \langle i+1, 0, N\rangle$, provided $i < K$.
  $\langle i, j, Y\rangle \rightarrow_a \langle i+1, j, Y\rangle$, provided $i < K$.
- $\rightarrow_b$: (Affects only process 1)
  $\langle i, 0, N\rangle \rightarrow_b \langle i+1, 0, N\rangle$, provided $i < K$.
  $\langle i, j, Y\rangle \rightarrow_b \langle i+1, j, Y\rangle$, provided $i < K$.
  $\langle i, 0, N\rangle \rightarrow_b \langle i+1, i, Y\rangle$, provided $i < K$.
- $\rightarrow_{c_i}, i < N$: (Affects processes $i+1$ and i. Each transition is of the form $(v, w) \rightarrow_{c_i} (v', w')$ where v and v' are states of process $i+1$ and w and w' are states of process i.)
  $(\langle i, 0, N\rangle, \langle K, 0, N\rangle) \rightarrow_{c_i} (\langle i+1, 0, N\rangle, \langle 0, 0, N\rangle)$, provided $i < K$
  $(\langle i, j, Y\rangle, \langle K, j', Y\rangle) \rightarrow_{c_i} (\langle i+1, j, Y\rangle, \langle 0, j', Y\rangle)$, provided $i < K$.
  $(\langle i, 0, N\rangle, \langle K, j', Y\rangle) \rightarrow_{c_i} (\langle i+1, i, Y\rangle, \langle 0, j', Y\rangle)$, provided $i < K$.
- $\rightarrow_{c_N}$: (Affects only process N)
  $\langle K, 0, N\rangle \rightarrow_{c_N} \langle 0, 0, N\rangle$.
  $\langle K, j, Y\rangle \rightarrow_{c_N} \langle 0, j, Y\rangle$.

So, after K a's and b's have been read process 1 gets stuck—a c_1 must occur to propagate a carry before the next a or b can be read. After K c_1's have occurred, a c_2 must be read before the next c_1. However, since c_2 is independent of a and b, this c_2 need not be read immediately after the K^{th} c_1. In general, $\mathfrak{A}$ permits higher digit carries to propagate asynchronously while the first component continues to read a's and b's from the input. However, it is easy to check that $\mathfrak{A}$ ensures that after every K c_i's, a c_{i+1} is read before the next c_i.

At some point, on reading a b, process 1 non-deterministically copies its counter value into the register and sets the third component of its state to Y. In the next round of carries, all the other processes get a signal to load their registers. Eventually all digits of the counter value when b occurred are stored in the register. Meanwhile, the counter continues to run freely on the remaining input. At the end of a word u, $\mathfrak{A}$ accepts u if each process has loaded its register and has the same value stored in the current counter as in the register.

Given this, it is straightforward, though tedious, to verify that $\mathfrak{A}$ does indeed accept L'_{K^N}.

Each process has $O(K^2)$ states. The total number of entries in the transition relations of $\mathfrak{A}$ is $O(NK^3)$. So, $\mathfrak{A}$ can be described in space polynomial in K and N. By our earlier analysis, if we determinize $\mathfrak{A}$ using our construction, we obtain an automaton whose local state space is $2^{(KN)^{O(N^2)}}$.

(Notice that we can implement a much simpler N-digit counter by allowing all N processes to synchronize on a and b. Then, we can eliminate the carry bits embedded in the input word—carries can be propagated "internally" when the processes synchronize. However, this naïve counter has $O(K^N)$ entries in its transition table and so is much larger, in real terms, than the counter we have described.) $\square$

References

[Die] V. Diekert: *Combinatorics on Traces*, *LNCS* **454** (1990).

[HU] J. Hopcroft, J.D. Ullman: *Introduction to automata, languages and computation*, Addison-Wesley (1979).

[Maz] A. Mazurkiewicz: Basic notions of trace theory, in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.), *Linear time, branching time and partial order in logics and models for concurrency*, *LNCS* **354**, (1989) 285–363.

[MS] M. Mukund, M. Sohoni: Keeping track of the latest gossip: Bounded time-stamps suffice, to appear in *Proc. FST&TCS '93, LNCS* **761**. Also available as *Report TCS-93-3*, School of Mathematics, SPIC Science Foundation, Madras (1993).

[Och] E. Ochmanski: Regular behaviour of concurrent systems, *EATCS Bulletin*, **27** (1985) 56–67.

[Pig] G. Pighizzini: Synthesis of nondeterministic asynchronous automata, in V. Diekert, W. Ebinger eds., *Proc. ASMICS Workshop on Infinite Traces*, Report 4/92, Fakultät Informatik, Universität Stuttgart, Germany (1992).

[Zie] W. Zielonka: Notes on finite asynchronous automata, *R.A.I.R.O.—Inf. Théor. et Appl.*, **21** (1987) 99–135.