Foundations of Software Technology and Theoretical Computer Science,
15th Conference, Proceedings, P.S. Thiagarajan (Ed.),
Lecture Notes in Computer Science **1026**, Springer-Verlag (1995) 456–470.

Determinizing Büchi Asynchronous Automata

Nils Klarlund[1], Madhavan Mukund[2], Milind Sohoni[3]

[1] BRICS[†], 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 Rd
Madras 600 017, India. E-mail: madhavan@ssf.ernet.in
[3] Dept of Computer Sc and Engg, Indian Institute of Technology
Bombay 400 076, India. E-mail: sohoni@cse.iitb.ernet.in

Abstract. Büchi asynchronous automata are a natural distributed machine model for recognizing ω-regular *trace languages*. Like their sequential counterparts, these automata need to be non-deterministic in order to capture all ω-regular languages. Thus complementation of these automata is non-trivial. Complementation is an important operation because it is fundamental for treating the logical connective "not" in decision procedures for monadic second-order logics.

In this paper, we present a direct determinization procedure for Büchi asynchronous automata, which generalizes Safra's construction for sequential Büchi automata. As in the sequential case, the blow-up in the state space is essentially that of the underlying subset construction.

Introduction

Finite-state automata are, by definition, sequential. To describe finite-state *concurrent* computations, Zielonka introduced *asynchronous automata* [Zie1]. An asynchronous automaton consists of a set of independent processes which cooperate to read their input. Each letter a in the alphabet is associated with a subset $\theta(a)$ of processes which jointly decide on a move when a is read.[1] The distribution function θ introduces an *independence relation* I between letters: $(a, b) \in I$ iff a and b are read by disjoint sets of processes.

Earlier, Mazurkiewicz had proposed a framework for studying concurrent systems where the alphabet Σ is equipped with a pre-specified independence relation I, describing the concurrency in the system [Maz]. Two words w and w' describe the same computation iff w' can be obtained from w by a finite sequence of permutations of adjacent independent letters. This gives rise to an equivalence relation on words over Σ. The equivalence class $[w]$ containing w is called a *trace*. A set of words L is a *trace language* if it obeys the equivalence relation generated by I— for each word w in L, all of $[w]$ is contained in L.

[†] Basic Research in Computer Science, Centre of the Danish National Research Foundation.

[1] Calling these automata *asynchronous* is, in a sense, misleading. The processes communicate synchronously. The asynchrony refers to the fact that different components of the network can proceed independently while reading the input.

Zielonka proved that any regular trace language over a *concurrent* alphabet (Σ, I) can be recognized by a deterministic asynchronous automaton over a *distributed* alphabet (Σ, θ), such that the independence relation generated by θ is exactly I. Gastin and Petit have extended the connection between asynchronous automata and trace languages to the setting of infinite inputs. In [GP], they introduce the class of Büchi asynchronous automata which accept precisely the class of ω-regular trace languages.

Like automata over infinite strings, Büchi asynchronous automata have close connections to logic [EM, Thi]. In order to exploit these connections—for instance, to automate verification of formulae defined using these logics—we need to develop techniques for manipulating these automata. Basic operations include complementation and determinization.

As in the sequential case, complementing Büchi asynchronous automata is non-trivial, since they are necessarily non-deterministic: deterministic Büchi asynchronous automata cannot recognize all ω-regular trace languages [GP]. With a Muller acceptance condition, deterministic automata suffice [DM], but a direct determinization procedure has so far been elusive.

Contributions of this paper

We extend the subset construction for asynchronous automata [KMS1] to a direct determinization construction for Büchi asynchronous automata, based on Safra's technique for determinizing Büchi automata on infinite strings [Saf]. The determinized automaton we construct has an acceptance condition described in terms of "Rabin pairs". As in the sequential case, we can efficiently simulate the complement of the determinized automaton using a non-deterministic Büchi asynchronous automaton. So, we also have a direct complementation construction for Büchi asynchronous automata. In both the determinized Rabin automaton and the complementary Büchi automaton, the number of local states of each process is exponential in the number of global states of the original automaton. As in Safra's original construction, this blow-up is essentially that of the underlying subset construction for these automata.

In related work, Muscholl [Mus] has described a complementation construction for Büchi asynchronous *cellular* automata, which are an alternative distributed model for recognizing trace languages [Zie2]. Her construction does not involve determinization—she makes use of progress measures [Kla] and directly constructs a non-deterministic complement automaton.

An asynchronous cellular automaton allocates a separate process for each letter in the input alphabet—even when the underlying system is completely sequential, a cellular automaton will have a number of components. Processes communicate using a non-standard variant of a shared memory. As a result, though both the approaches are formally equivalent, asynchronous automata seem to be more natural models for describing distributed systems.

Converting between asynchronous automata and asynchronous cellular automata involves a blow-up in the state space of each process which is exponential

in $|\Sigma|$, the size of the input alphabet. However, since $|\Sigma|$ could itself be exponential in the size of the global state space of the automaton, there is effectively a double exponential blow-up in this translation. So, complementing asynchronous automata directly using our construction can be significantly more efficient than complementing them indirectly via the construction described in [Mus].

In general, it appears to be advantageous to work directly with asynchronous automata for automating decision procedures in logic, instead of using asynchronous cellular automata. Incorporating the alphabet into the state space of the automaton is known to be expensive in such applications—for example, the decision procedure for monadic second-order logic on strings generates alphabets that are exponential in the number of free variables in the input formula; see [HJJ] for techniques which allow automata with exponentially sized alphabets to be represented and manipulated within polynomial bounds.

Working directly with asynchronous automata is also relevant to *model checking* —a technique for mechanically verifying if a program satisfies a property specified in a logical language. If the same kind of automata are used both for describing the program and for checking satisfiability, the model checking problem reduces to a simple intersection problem involving the automata [VW]. Since asynchronous automata are a natural model for distributed programs, automata-theoretic model checking can be applied to the logics considered in [EM, Thi].

The paper is organized as follows. We begin with some definitions regarding asynchronous automata. In Section 2 we introduce Büchi and Rabin asynchronous automata and formulate the problem. The next three sections recapitulate some basic techniques developed in [KMS1, MS] for manipulating asynchronous automata. In Section 6 we show how to apply these techniques to determinize Büchi asynchronous automata. Due to space limitations we have been forced to omit many examples and proofs. For a more detailed exposition, the reader is referred to [KMS2].

1 Preliminaries

The following definitions are essentially those of [KMS1] adapted to the setting of infinite inputs.

Distributed alphabets Let $\mathcal{P}$ be a finite set of processes, where the size of $\mathcal{P}$ is N. 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 non-empty 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 $P \subseteq \mathcal{P}$, V_P denotes the product $\prod_{p \in P} V_p$. An element $\vec{v}$ of V_P is a tuple or *joint state* that determines a local state for each p in P. We refer to a joint state from V_P as a *P-state*. A $\mathcal{P}$-state is also called a *global state*.

Given $\vec{v} \in V_P$, and $P' \subseteq P$, $\vec{v}_{P'}$ denotes the projection of $\vec{v}$ onto $V_{P'}$. Also, $\vec{v}_{\overline{P'}}$ abbreviates $\vec{v}_{P - P'}$. For a singleton $p \in P$, we write $\vec{v}_p$ for $\vec{v}_{\{p\}}$. 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 $\vec{v} \in V_P$ and $\theta(a) \subseteq P$, we write $\vec{v}_a$ for $\vec{v}_{\theta(a)}$ and $\vec{v}_{\overline{a}}$ for $\vec{v}_{\overline{\theta(a)}}$.

Asynchronous automata An *asynchronous automaton* $\mathfrak{A}$ over (Σ, θ) is of the form $(\{V_p\}_{p\in\mathcal{P}}, \{\rightarrow_a\}_{a\in\Sigma}, \mathcal{V}_0)$, where $\rightarrow_a \subseteq V_a \times V_a$ is the *local transition relation* for a, and $\mathcal{V}_0 \subseteq V_\mathcal{P}$ is a set of *initial* global states. Each relation $\rightarrow_a$ specifies how the processes $\theta(a)$ 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 $\vec{v}_a \rightarrow_a \vec{v}'_a$ and $\vec{v}_{\overline{a}} = \vec{v}'_{\overline{a}}$.

$\mathfrak{A}$ is called *deterministic* if the global 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 α be an infinite word over Σ. It is convenient to think of α as a function of time; i.e., $\alpha : \mathbf{N} \rightarrow \Sigma$. (We use $\mathbf{N}$ to denote the set $\{1, 2, \ldots\}$ and $\mathbf{N}_0$ for $\{0, 1, 2, \ldots\}$.) A *global run* of $\mathfrak{A}$ on $\alpha : \mathbf{N} \rightarrow \Sigma$ is a function $\rho : \mathbf{N}_0 \rightarrow V_\mathcal{P}$ such that $\rho(0) \in \mathcal{V}_0$ and, for $i \in \mathbf{N}$, $\rho(i{-}1) \stackrel{\alpha(i)}{\Longrightarrow} \rho(i)$.

Analogously, we represent a finite word $u \in \Sigma^*$ of length m as a function $u : [1..m] \rightarrow \Sigma$, where $[i..j]$ abbreviates the set $\{i, i{+}1, \ldots, j\}$. A global run of $\mathfrak{A}$ on $u : [1..m] \rightarrow \Sigma$ is a function $\rho : [0..m] \rightarrow V_\mathcal{P}$ such that $\rho(0) \in \mathcal{V}_0$ and, for $i \in [1..m]$, $\rho(i{-}1) \stackrel{u(i)}{\Longrightarrow} \rho(i)$.

Let ρ be a global run. For $P \subseteq \mathcal{P}$, ρ_P denotes the projection of ρ onto the P-components. So, ρ_P is a sequence of P-states. As usual, $inf(\rho_P)$ denotes the set of P-states which occur infinitely often in ρ_P; $inf(\rho_P) = \{\vec{v} \in V_P \mid \text{for infinitely many } i,\ \rho_P(i) = \vec{v}\}$.

2 Asynchronous automata on infinite inputs

To define how an asynchronous automaton accepts an infinite word α, we have to examine the communication between processes in the limit, as α is read.

Limit graphs With each infinite word α, we associate an undirected graph $\mathcal{G}_\alpha = (\mathcal{P}, E_\alpha)$ called the *limit graph* of α. The graph has an edge between processes p and q provided they synchronize infinitely often while $\mathfrak{A}$ processes α. In other words, $(p, q) \in E_\alpha$ iff for infinitely many i, $\{p, q\} \subseteq \theta(\alpha(i))$. Let $Conn_\alpha$ denote the maximal connected components of $\mathcal{G}_\alpha$.

Let $Finite_\alpha$ denote the set of processes which move only finitely often while $\mathfrak{A}$ reads α—i.e., p belongs to $Finite_\alpha$ if there are only finitely many i such that $p \in \theta(\alpha(i))$. Clearly, if $p \in Finite_\alpha$ then the singleton $\{p\}$ belongs to $Conn_\alpha$.

Büchi asynchronous automata A *Büchi asynchronous automaton* is a pair $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ where $\mathfrak{A}$ is an asynchronous automaton and $\mathcal{T}_B$ is a *Büchi acceptance table*. The table $\mathcal{T}_B$ is a list $(\tau_1, \tau_2, \ldots, \tau_k)$. Each entry τ_i in $\mathcal{T}_B$ is of the form $(\mathcal{C}, T, \{(p_C, G_C)\}_{C\in\mathcal{C}})$, where $\mathcal{C}$ is a partition of $\mathcal{P}$, T is a subset of $\mathcal{P}$ and, for each subset $C \in \mathcal{C}$, p_C is a designated process from C and G_C is a set of p_C-states. We call the processes $\{p_C\}_{C\in\mathcal{C}}$ the *signalling processes* in τ_i.

A run ρ of the automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ on an input α is said to *satisfy* an entry $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C\in\mathcal{C}})$ in $\mathcal{T}_B$ provided $\mathcal{C} = Conn_\alpha$, $T = Finite_\alpha$ and, for each signalling process p_C, $inf(\rho_{p_C}) \cap G_C \neq \emptyset$. The automaton accepts α if there is a run ρ on α and a table entry τ such that ρ satisfies τ.

Recall that every process p in $Finite_\alpha$ constitutes a separate singleton component in $Conn_\alpha$. For a signalling process $p \in T$, the set G_p denotes the set of possible *terminating states* for p. On the other hand, for a signalling process p which does not belong to T, G_p is a set of *recurring states*, one of which must be visited infinitely often by $\mathfrak{A}$ for ρ to satisfy τ.

Our definition of Büchi asynchronous automata is adapted from [Mus] and differs from the original formulation of Gastin and Petit [GP]. The crucial part of our definition is the extra information we record about $Conn_\alpha$ in each entry of the acceptance table. This allows us to separate the processes in $\mathfrak{A}$ into independent groups. After a finite prefix of α has been read, there will be no further synchronizations between processes in different connected components of $\mathcal{G}_\alpha$. So, in the limit, each subset $C \in Conn_\alpha$ moves as a separate, independent unit.

Since non-deterministic Büchi asynchronous automata are strictly more powerful than their deterministic counterparts [GP], to determinize these automata we have to strengthen the acceptance condition. We shall work with a generalization of the "pairs" condition proposed by Rabin [Rab].

Rabin asynchronous automata A *Rabin asynchronous automaton* is a pair $\mathbf{R}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_R)$ where $\mathfrak{A}$ is an asynchronous automaton and $\mathcal{T}_R$ is a *Rabin acceptance table*. The table $\mathcal{T}_R$ is a list $(\tau_1, \tau_2, \ldots, \tau_k)$. Each entry τ_i in $\mathcal{T}_R$ is of the form $(\mathcal{C}, T, \{(p_C, pairs_C)\}_{C \in \mathcal{C}})$, where $\mathcal{C}$, T and p_C are as in a Büchi acceptance table and, for each signalling process p_C, $pairs_C$ is a list $\{(G_C^j, R_C^j)\}_{j \in [1..k_C]}$ such that for each pair (G_C^j, R_C^j), both G_C^j and R_C^j are subsets of V_{p_C}.

The automaton $\mathbf{R}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_R)$ accepts an input α if there is a run ρ of $\mathfrak{A}$ on α such that for some entry $\tau = (\mathcal{C}, T, \{(p_C, pairs_C)\}_{C \in \mathcal{C}})$ in the table $\mathcal{T}_R$, $\mathcal{C} = Conn_\alpha$, $T = Finite_\alpha$ and, for each signalling process p_C, there is an entry (G_C^j, R_C^j) in $pairs_C$ such that $inf(\rho_{p_C}) \cap G_C^j \neq \emptyset$ and $inf(\rho_{p_C}) \cap R_C^j = \emptyset$.

The problem For a given non-deterministic Büchi asynchronous automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ over (Σ, θ), construct a deterministic Rabin asynchronous automaton $\mathbf{R}\mathfrak{B} = (\mathfrak{B}, \mathcal{T}_R)$ over (Σ, θ), such that $\mathbf{B}\mathfrak{A}$ and $\mathbf{R}\mathfrak{B}$ accept the same set of infinite words over Σ.

Notice that an asynchronous automaton where $\mathcal{P}$ is a singleton $\{p\}$ is just a conventional sequential finite state automaton. Further, if $\mathcal{P} = \{p\}$, our definitions of Büchi and Rabin asynchronous automata reduce to the standard formulations of these automata in the setting of infinite strings [Tho].

For sequential Büchi automata, Safra has described an elegant determinization construction which is an extension of the classical subset construction for finite automata [Saf]. We do not have space to describe this construction here. However, it is possible to understand the salient features of our construction *without* getting into the precise details of how Safra's construction works.

To determinize Büchi asynchronous automata, we shall apply Safra's construction in a distributed setting. Let $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ be a Büchi asynchronous automaton. Our strategy will be to construct a deterministic Rabin automaton $\mathbf{R}\mathfrak{B}_\tau = (\mathfrak{B}_\tau, \mathcal{T}_{R_\tau})$ corresponding to each entry τ in the table $\mathcal{T}_B$. The automaton $\mathbf{R}\mathfrak{B}_\tau$ accepts an input α provided there is a run ρ of $\mathbf{B}\mathfrak{A}$ which satisfies τ.

We can then combine the individual automata $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau\in\mathcal{T}_B}$ into a deterministic Rabin automaton $\mathbf{R}\mathfrak{B}$ which accepts exactly the same infinite strings as $\mathbf{B}\mathfrak{A}$.

To construct the automaton $\mathbf{R}\mathfrak{B}_\tau$ for the table entry $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C\in\mathcal{C}})$ we have to check that for each signalling process p_C, $inf(\rho_{p_C}) \cap G_C \neq \emptyset$. To do this, we run Safra's construction for each signalling process p_C, using the subset construction for asynchronous automata [KMS1] in place of the classical subset construction for sequential automata.

The catch is that each signalling process p_C may meet its recurring set G_C infinitely often along a different run. So, we have to further ensure that the accepting runs detected by the independent copies of Safra's construction at each signalling process are mutually consistent. This will involve some analysis of the way information is passed between the connected components in $\mathcal{G}_\alpha$ *before* they branch out as independent groups.

3 Local and global views

We represent words over a distributed alphabet as labelled partial orders. The notions we use are essentially those of trace theory [Maz].

Events With $\alpha : \mathbf{N} \to \Sigma$, we associate a set of *events* $\mathcal{E}_\alpha$. Each event $(i, \alpha(i))$ consists of a letter $\alpha(i)$ together with the time i of its occurrence. 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 $\mathcal{E}$ for $\mathcal{E}_\alpha$. If $e = (i, a)$ is an event, then we may use e instead of a in abbreviations such as V_e, which stands for V_a, i.e., $V_{\theta(a)}$, or $\rightarrow_e$, which is just $\rightarrow_a$. For $p \in \mathcal{P}$ and $e = (i, a)$, we write $p \in e$ to denote that $p \in \theta(a)$. When $e = 0$, we define $p \in e$ to hold for all $p \in \mathcal{P}$. If $p \in e$, then we say that e is a *p-event*.

Ordering relations on $\mathcal{E}$ The word α naturally imposes a total order $\leq$ on events: $e \leq f$ if e happens at time i and f happens at time j with $i \leq j$.

Each process p imposes a total order $\leq_p$ on the events in which it participates. Thus $e \leq_p f$ if p participates in both e and f and $e \leq f$. If e is the p-event that immediately precedes the p-event f, then we write $e \lhd_p f$. Thus $e \lhd_p f$ if $e \leq_p f$ and no g with $e < g < f$ is a p-event.

The asynchronous nature of the automaton is reflected more accurately by the partial order generated by the relations $\{\lhd_p\}_{p\in\mathcal{P}}$ than by the temporal order $\leq$. We say that e is an immediate predecessor of f and write $e \lhd f$ if $e \lhd_p f$ for some p. Let $\sqsubseteq$ be the reflexive and transitive closure of $\lhd$. If $e \sqsubseteq f$, then we say e is *below* f. Note that the initial event 0 is below any event. The set of events below e is denoted $e{\downarrow}$. They represent the only synchronizations that may have affected the state of $\mathfrak{A}$ at e.

Ideals An *ideal* I is any set of events closed with respect to $\sqsubseteq$. Ideals represent possible partial computations of the system. We assume that every ideal I we consider is non-empty—i.e., 0 always belongs to I. Let α_m denote the prefix of α of length m. Then the events $\{(i, \alpha(i)) \mid i \leq m\} \cup \{0\}$ form an ideal. Conversely, every ideal gives rise to a subword of α—if I is the finite

ideal $\{0,(i_1,a_1),(i_2,a_2),\ldots,(i_m,a_m)\}$, then $\alpha[I] : [1..m] \to \Sigma$ is the word $\alpha(i_1)\alpha(i_2)\cdots\alpha(i_m) = a_1a_2\cdots a_m$. Similarly, we can associate infinite ideals in $\mathcal{E}$ with infinite subsequences of α. Even when I is finite, $\alpha[I]$ is not, in general, a prefix of α because of the asynchronous manner in which α is processed. Clearly the entire set $\mathcal{E}$ is an ideal, as is the set $e{\downarrow}$ for any event $e \in \mathcal{E}$.

P-views Let I be an ideal. The *p-view of I*, $\partial_p(I)$, is the set $\{e \in I \mid \exists f \in I.\ p \in f \text{ and } e \sqsubseteq f\}$. So, $\partial_p(I)$ is the set of all events in I which p can "see". If the number of p-events in I is finite—for instance, if I itself is finite—it is easy to see that $\partial_p(I) = max_p(I){\downarrow}$, where $max_p(I)$ is the $\sqsubseteq$-maximum p-event in I.

For $P \subseteq \mathcal{P}$, the P-view of I, denoted $\partial_P(I)$, is $\bigcup_{p\in P} \partial_p(I)$. Notice that $\partial_P(I)$ is always an ideal. In particular, we have $\partial_{\mathcal{P}}(I) = I$.

4 Local runs and histories

For the rest of this section, we fix a (non-determinstic) asynchronous automaton $\mathfrak{A} = (\{V_p\}_{p\in\mathcal{P}}, \{\rightarrow_a\}_{a\in\Sigma}, \mathcal{V}_0)$.

Neighbourhoods The *neighbourhood* of an event e, $nbd(e)$, consists of e together with its immediate predecessors; i.e., $nbd(e) = \{e\} \cup \{f \mid f \lhd e\}$. Notice that if $e \in \partial_P(I)$ for some $P \subseteq \mathcal{P}$, then $nbd(e) \subseteq \partial_P(I)$ as well.

Local runs A *local run* on an ideal I assigns a joint state to each event in I in such a way that all neighbourhoods are consistently labelled. More precisely, a local run on I is a function r that assigns to each $e \in I$ an e-state—i.e., a state in V_e—such that $r(0) \in \mathcal{V}_0$ and for all $e \neq 0$, r is consistent with $\rightarrow_e$ in $nbd(e)$ in the following sense: suppose that $\vec{v}$ is the e-state whose p-component, for each $p \in e$, is the same as the p-component of $r(f_p)$, where f_p is the immediate p-predecessor of e. In other words, for each $p \in e$, $\vec{v}_p = r(f_p)_p$, where $f_p \lhd_p e$. Then r is such that $\vec{v} \rightarrow_e r(e)$. Given a local run r, there is a natural "last" global state $\vec{v}$ defined by $\vec{v}_p = r(\max_p(I))$ for all p. We say that $\vec{v}$ is a state of $\mathfrak{A}$ *on* I. Similarly, a P-state of $\mathfrak{A}$ on I is $\vec{v}_P$, where $\vec{v}$ is a state of $\mathfrak{A}$ *on* I.

Let $\mathcal{R}(I)$ denote the set of all local runs on I. The following is easy to verify.

Proposition 1. *Let $\alpha : \mathbf{N} \to \Sigma$ and $I \subseteq \mathcal{E}_\alpha$ an ideal. Then, there is a 1-1 correspondence between $\mathcal{R}(I)$ and the set of global runs of $\mathfrak{A}$ on $\alpha[I]$.*

Histories A history on an ideal I is a partial function h that assigns joint states to some events in I. Thus $dom(h) \subseteq I$ and when $h(e)$ is defined it denotes a tuple in V_e. A history is *reachable* if there is some local run r on I such that $h(e) = r(e)$ for e in $dom(h)$. A set of histories H is *consistent* if each pair of histories h and h' in the set agree on all common events; i.e., for each $h, h' \in H$, for each e in $dom(h) \cap dom(h')$, $h(e) = h(e')$.

History Products Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of ideals with $J = \bigcup_{j\in[1..n]} I_j$. Let $\{h_1, h_2, \ldots, h_n\}$ be a consistent set of histories such that h_j is a history over I_j for each $j \in [1..n]$. We define the *product* $h = \bigotimes_{j\in[1..n]} h_j$ as follows:

$dom(h) = \{e \in J \mid \text{ for all } j \in [1..n], \text{ if } e \in I_j \text{ then } e \in dom(h_j)\}$

$h(e) = h_k(e)$, where k is such that $e \in dom(h_k)$ (the choice of k does not matter since $\{h_j\}_{j\in[1..n]}$ is consistent).

In other words, h is a history over J which inherits its values from the set $\{h_j\}_{j\in[1..n]}$. The value $h(e)$ is defined whenever $h_j(e)$ is defined *for all* j such that $e \in I_j$. This means that if e is in $I_j \cap I_k$ for some pair $\{I_j, I_k\} \subseteq \mathcal{I}$ and $e \in dom(h_j)$ but $e \notin dom(h_k)$, then $e \notin dom(h)$.

We extend the notion of product to sets of histories spanning a set of ideals. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of ideals, with $J = \bigcup_{j\in[1..n]} I_j$. Let $\mathcal{H}_\mathcal{I} = \{H_1, H_2, \ldots, H_n\}$ where H_j is a set of histories over I_j for each $j \in [1..n]$. A *choice* from $\mathcal{H}_\mathcal{I}$ is a set $\{h_j\}_{j\in[1..n]}$ which picks out a history $h_j \in H_j$ for each $j \in [1..n]$. The choice is consistent if the set $\{h_j\}_{j\in[1..n]}$ is. We then define

$$\bigotimes \mathcal{H}_\mathcal{I} = \{ \bigotimes_{j\in[1..n]} h_j \mid \{h_j\}_{j\in[1..n]} \text{ is a consistent choice from } \mathcal{H}_\mathcal{I}\}.$$

So, $\bigotimes \mathcal{H}_\mathcal{I}$ contains all histories on J that may be pieced together from mutually consistent histories in the collection $\mathcal{H}_\mathcal{I}$.

Products of histories play a crucial role in the subset construction for asynchronous automata [KMS1]. Suppose X_p and X_q are the sets of possible states of p and q on ideal I. The set of possible joint $\{p,q\}$-states on I is *not*, in general, the naïve product $X_p \times X_q$. To determine which states from $X_p \times X_q$ are valid $\{p,q\}$-states on I, p and q have to record additional information about the runs leading to each state in the current subsets X_p and X_q. Since the amount of information that a process can store is bounded, it can at best record histories defined over a finite subset of the events it has seen.

In the subset construction of [KMS1], on an ideal I, each process p maintains the set H_p of all reachable histories over a specific bounded subset of $\partial_p(I)$. This subset includes $max_p(I)$, so H_p has, in particular, information about all the possible states that p can be in on I. Suppose a subset $P \subseteq \mathcal{P}$ synchronizes. In terms of the notation above, we have $\mathcal{I} = \{\partial_p(I)\}_{p\in P}$, $J = \partial_P(I)$ and $\mathcal{H}_\mathcal{I} = \{H_p\}_{p\in P}$. The goal is to ensure that $\bigotimes \mathcal{H}_\mathcal{I}$ generates all possible consistent "joint" histories of P over an appropriate subset of $\partial_P(I)$. This will allow the processes in P to jointly compute all the possible moves they can make on reading the new letter from the input.

The key step is to characterize when the product of a set of reachable histories $\{h_j\}_{j\in[1..n]}$ over $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ remains a reachable history over the joint ideal $J = \bigcup_{j\in[1..n]} I_j$. For this, we need the notion of a frontier.

Frontiers Let I and J be ideals and p a process. We say that event e of I is an *p-sentry* for I relative to J if e is also in J and its p-successor is in J but not in I. Thus the process p "leaves" I at e. Let $border(I,J)$ be the set of all such sentries. Note that there is at most one p-sentry for each p, so there are at most N events in $border(I,J)$—recall that $N = |\mathcal{P}|$. In general,

$border(I,J) \neq border(J,I)$. We are normally interested in the two sets together, which we denote $frontier(I,J)$; i.e., $frontier(I,J) = border(I,J) \cup border(J,I)$. It is clear that $frontier(I,J) = frontier(J,I)$ and $frontier(I,I) = \emptyset$. We then have the following crucial result which is proved in [KMS1].

Lemma 2. *Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of ideals and $\{h_j\}_{j \in [1..n]}$ a consistent set of histories such that for each $j \in [1..n]$:*

(i) h_j is a reachable history over I_j; and
(ii) $dom(h_j)$ includes $\bigcup_{k \in [1..n]} frontier(I_j, I_k)$.

Then $h = \bigotimes_{j \in [1..n]} h_j$ is a reachable history over $J = \bigcup_{j \in [1..n]} I_j$.

So, whenever the reachable histories $\{h_j\}_{j \in [1..n]}$ span all the frontiers between the ideals in $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$, their product is also reachable.

Recall that each process p maintains H_p, the set of all reachable histories over a specific subset of $\partial_p(I)$. Suppose that this specific subset of $\partial_p(I)$ includes $frontier_p(I)$, where $frontier_p(I) = \bigcup_{q \in \mathcal{P}} frontier(\partial_p(I), \partial_q(I))$. Then, if $\mathcal{I} = \{\partial_p(I)\}_{p \in P}$ and $\mathcal{H}_\mathcal{I} = \{H_p\}_{p \in P}$, the previous lemma guarantees that every history in $\bigotimes \mathcal{H}_\mathcal{I}$ is reachable in $\partial_P(I)$.

The problem now is for a process p to compute the bounded set of events $frontier_p(I)$. This can be done using slightly larger, but still bounded, sets of events called primary and secondary information, which between them subsume the frontiers.

5 Primary and secondary information

Primary information Let I be a finite ideal. Recall that $max_p(I)$ denotes the $\sqsubseteq$-maximum p-event in I. The *primary information* of I, $primary(I)$, is the set of events $\{max_p(I)\}_{p \in \mathcal{P}}$. We can define $primary(I)$ analogously for infinite ideals as well, where we include the events $max_p(I)$ for only those processes p such that there are only finitely many p-events in I.

Secondary and tertiary information Let I be a finite ideal. The *secondary information* in I, $secondary(I)$, is the set of events $\bigcup_{p \in \mathcal{P}} primary(\partial_p(I))$. The *tertiary information* in I, $tertiary(I)$, is the set $\bigcup_{p,q \in \mathcal{P}} primary(\partial_p(\partial_q(I)))$.

The primary information of I represents the latest information available in I about each process in the system. Similarly, the secondary information $primary(\partial_p(I))$ is the latest information that process p has in I about the other processes in the system, while the tertiary information $primary(\partial_q(\partial_p(I)))$ is the latest information that p has about the primary information of q in I.

It is clear that every event in $primary(I)$ also belongs to $secondary(I)$, since $max_p(\partial_p(I)) = max_p(I)$ for all $p \in P$. Similarly, every event in $secondary(I)$ belongs to $tertiary(I)$.

Let I and J be ideals. If I and J satisfy a simple condition, the events in $frontier(I,J)$ can be characterized in terms of the primary and secondary information of I and J, as described in the following lemma.

Lemma 3 [KMS1]. *Let I and J be ideals such that $I = \partial_P(K)$ and $J = \partial_Q(K)$, where K is an ideal and $P, Q \subseteq \mathcal{P}$ are sets of processes. Let e be a p-sentry for I with respect to J. Then $e = max_p(I)$ and, for some process q, $e = max_p(\partial_q(J))$. Thus, $e \in primary(I) \cap secondary(J)$.*

Let I be an ideal. From the previous lemma, it is clear that for a process P to maintain reachable histories over $frontier_p(I)$, it is sufficient for p to maintain reachable histories over $secondary(\partial_p(I))$. Processes can unambiguously keep track of their primary and secondary information by using time-stamps.

Time-stamps and the subset construction Let I be a finite ideal. Then, there are at most N^3 distinct events in $tertiary(I)$. We can thus use a finite set $\mathfrak{L}$ of labels to *time-stamp* each event in this set—let this assignment of time-stamps be denoted by a function $\lambda : tertiary(I) \to \mathfrak{L}$. For $p \in \mathcal{P}$, let λ_p denote the restriction of λ to $\partial_p(I)$. It turns out that the processes in $\mathcal{P}$ can locally maintain and update the functions λ_p so that, overall, the events in $tertiary(I)$ are assigned consistent time-stamps.

Theorem 4 Time-stamping [MS]. *For any distributed alphabet (Σ, θ), we can fix a finite set of labels $\mathfrak{L}$ and construct a deterministic asynchronous automaton $\mathfrak{A}_T$ over (Σ, θ) in which, on any finite ideal I, each process p maintains $\lambda_p : secondary(\partial_p(I)) \to \mathfrak{L}$, where λ_p is the restriction to $\partial_p(I)$ of a consistent labelling $\lambda : tertiary(I) \to \mathfrak{L}$. Process p maintains λ_p as a function from $\mathcal{P} \times \mathcal{P}$ to $\mathfrak{L}$. The value $\lambda_p(q, r)$ is the label assigned to the event $max_r(max_q(\partial_p(I)))$.*

The automaton $\mathfrak{A}_T$ allows each process to maintain reachable histories over the set $secondary(\partial_p(I))$—each history h is maintained as a partial function assigning joint states to labels in $\mathfrak{L}$ such that whenever $\lambda(e) = \ell$ for some event $e \in secondary(\partial_p(I))$, $h(\ell)$ is defined and yields an e-state. In conjunction with Lemmas 2 and 3, this yields the following result.

Theorem 5 Subset construction [KMS1]. *Let $\mathfrak{A}$ be a non-deterministic asynchronous automaton over (Σ, θ). Then, we can construct a deterministic asynchronous automaton $\mathfrak{A}_S$ over (Σ, θ) such that for any finite ideal I, the unique global state $\vec{v}$ reached by $\mathfrak{A}_S$ on I has the following properties:*

(i) For each process p, $\vec{v}_p$ contains H_p, the set of all reachable histories over $secondary(\partial_p(I))$.
(ii) For any subset P of $\mathcal{P}$, we can compute from the information in the P-state $\vec{v}_P$ the set of all possible P-states of $\mathfrak{A}$ on I. In particular, from $\vec{v}$ we can recover all possible global states of $\mathfrak{A}$ on I.

6 Determinizing Büchi asynchronous automata

We now have enough machinery at hand to apply Safra's construction in a distributed setting. Recall that we are initially given a non-deterministic Büchi asynchronous automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$. Our goal is to construct a deterministic

Rabin asynchronous automaton $\mathbf{R}\mathfrak{B} = (\mathfrak{B}, \mathcal{T}_R)$ which accepts the same set of infinite strings that $\mathbf{B}\mathfrak{A}$ does.

As we remarked earlier, our strategy is to construct a separate deterministic Rabin automaton $\mathbf{R}\mathfrak{B}_\tau = (\mathfrak{B}_\tau, \mathcal{T}_{R_\tau})$ for each entry τ in the Büchi table $\mathcal{T}_B$ such that $\mathbf{R}\mathfrak{B}_\tau$ accepts an input α iff there is a run ρ of $\mathbf{B}\mathfrak{A}$ on α which satisfies τ. We shall then combine these individual automata $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau \in \mathcal{T}_B}$ into a single automaton $\mathbf{R}\mathfrak{B}$ which accepts the same inputs as $\mathbf{B}\mathfrak{A}$.

Let $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C \in \mathcal{C}})$ be an entry from $\mathcal{T}_B$. We first describe how to construct the corresponding Rabin automaton $\mathbf{R}\mathfrak{B}_\tau$. For simplicity, we assume that $T = \emptyset$—i.e., a run ρ of $\mathbf{B}\mathfrak{A}$ on an input α can satisfy τ only if $Finite_\alpha = \emptyset$. In other words, every process moves infinitely often as $\mathfrak{A}$ reads α. Later, we shall see how to eliminate this "progress" assumption.

The automaton $\mathbf{R}\mathfrak{B}_\tau$ has to check that there is a run ρ of $\mathfrak{A}$ on α such that along ρ, each signalling process p_C visits some recurring state from G_C infinitely often. Each process p_C can detect whether there is some local run ρ_C of $\mathfrak{A}$ on α which meets G_C infinitely often by running Safra's construction locally. However, we have to check that the individual runs $\{\rho_C\}_{C \in \mathcal{C}}$ are mutually consistent.

Let $\mathcal{I}_\alpha = \{I_C\}_{C \in Conn_\alpha}$ be the set of ideals such that $I_C = \partial_C(\mathcal{E})$ for each $C \in Conn_\alpha$. (Recall that each set C is a subset of $\mathcal{P}$, so the C-view of $\mathcal{E}$ is well-defined.) If there is a run of $\mathfrak{A}$ satisfying τ, it must be the case that $\mathcal{C} = Conn_\alpha$, so we can alternatively regard $\mathcal{I}_\alpha$ as the collection $\{\partial_C(\mathcal{E})\}_{C \in \mathcal{C}}$.

Let I_α^{joint} denote the set of events which occur in more than one ideal in the collection $\mathcal{I}_\alpha$—i.e., $I_\alpha^{joint} = \{e \in \mathcal{E} \mid \exists C, C' \in \mathcal{C}.\ C \neq C' \text{ and } e \in I_C \cap I_{C'}\}$. Since $\mathcal{C} = Conn_\alpha$, it must be the case that I_α^{joint} is finite—"above" I_α^{joint}, the ideals in $\mathcal{I}_\alpha$ are pairwise disjoint. Moreover, the union $\cup\mathcal{I}_\alpha$ is the entire set $\mathcal{E}$. So, if we can ensure that the local runs $\{r_C\}_{C \in \mathcal{C}}$ agree on the events in I_α^{joint}, they can be "pasted" together to form a global run ρ of $\mathfrak{A}$ satisfying τ.

Actually, it is not necessary that the local runs $\{r_C\}_{C \in \mathcal{C}}$ agree on the *entire* set I_α^{joint} in order to synthesize a global run ρ satisfying τ. It is sufficient for these local runs to agree along the frontiers of the ideals in $\mathcal{I}_\alpha$.

Lemma 6. *Let $\alpha : \mathbf{N} \to \Sigma$ be an infinite word. For $I_C \in \mathcal{I}_\alpha$, let $frontier(I_C, \mathcal{I}_\alpha)$ denote the set of events spanning the frontiers of I_C with respect to all the ideals in $\mathcal{I}_\alpha$—i.e., $frontier(I_C, \mathcal{I}_\alpha) = \bigcup_{C' \in \mathcal{C}} frontier(I_C, I_{C'})$.*

Let $\mathcal{R} = \{r_C\}_{C \in \mathcal{C}}$ be a set of local runs of $\mathfrak{A}$ on α such that:

(i) For $C \in \mathcal{C}$, r_C is a local run over I_C.
(ii) For each pair $C, C' \in \mathcal{C}$, the local runs r_C and $r_{C'}$ agree on $frontier(I_C, I_{C'})$.

Then, there is a local run r of $\mathfrak{A}$ over $\mathcal{E}$ which agrees with each run $r_C \in \mathcal{R}$ for all events $e \in I_C$ "above" $frontier(I_C, \mathcal{I}_\alpha)$. In other words, for each $C \in \mathcal{C}$, for each $e \in I_C$, if there exists $f \in frontier(I_C, \mathcal{I}_\alpha)$ such that $f \sqsubseteq e$, then $r(e) = r_C(e)$.

Proof For $C \in \mathcal{C}$, let h_C be the history generated by restricting r_C to the set $\{e \in I_C \mid \exists f \in frontier(I_C, \mathcal{I}_\alpha).\ f \sqsubseteq e\}$. It is easy to check that the histories in $\{h_C\}_{C \in \mathcal{C}}$ satisfy the assumptions of Lemma 2. So $h = \bigotimes_{C \in \mathcal{C}} h_C$ is a reachable history over $\cup\mathcal{I}_\alpha = \mathcal{E}$. Let r be the local run extending h to all of $\mathcal{E}$. □

So, if the local runs $\{r_C\}_{C \in \mathcal{C}}$ detected by the copies of Safra's construction agree along the frontiers in $\mathcal{I}_\alpha$, we can synthesize a local run r over $\mathcal{E}$ which agrees with each local run r_C outside I_α^{joint}. It is clear that the global run ρ of $\mathfrak{A}$ on α which corresponds to r does in fact satisfy τ. Of course, to check the conditions of the previous lemma, we have to verify that, in the limit, the local runs detected by each signalling process agree on the frontier events. In principle, this involves an infinite amount of computation. However, since there is only a finite amount of communication across the ideals in $\mathcal{I}_\alpha$, the frontier events of interest get "frozen" at some finite stage.

Lemma 7. *Let $\alpha : \mathbf{N} \rightarrow \Sigma$ be an infinite word. Let J be an ideal such that $I_\alpha^{joint} \subseteq J \subseteq \mathcal{E}$. Then, for each pair $\{C, C'\}$ in $Conn_\alpha$, $frontier(\partial_C(J), \partial_{C'}(J)) = frontier(\partial_C(\mathcal{E}), \partial_{C'}(\mathcal{E}))$.*

Proof Observe that for every J such that $I_\alpha^{joint} \subseteq J$, and for every pair $\{C, C'\}$ in $Conn_\alpha$, $\partial_C(J) \cap \partial_{C'}(J) = \partial_C(\mathcal{E}) \cap \partial_{C'}(\mathcal{E})$. The result then follows. □

Let α be an infinite word and C, C' be components in $Conn_\alpha$. From Lemma 3, we know that the events in $frontier(\partial_C(\mathcal{E}), \partial_{C'}(\mathcal{E}))$ are contained in $secondary(\partial_C(\mathcal{E}))$ and $secondary(\partial_{C'}(\mathcal{E}))$.

Let p_C and $p_{C'}$ be processes in C and C' respectively. From the definition of $Conn_\alpha$, it follows that $\partial_C(\mathcal{E}) = \partial_{p_C}(\mathcal{E})$ and $\partial_{C'}(\mathcal{E}) = \partial_{p_{C'}}(\mathcal{E})$. So, $frontier(\partial_C(\mathcal{E}), \partial_{C'}(\mathcal{E}))$ is, in fact, contained in the secondary information of both $\partial_{p_C}(\mathcal{E})$ and $\partial_{p_{C'}}(\mathcal{E})$.

Let $e \in secondary(\partial_{p_C}(\mathcal{E}))$. From the definition of secondary information, $e = max_r(\partial_q(\partial_{p_C}(\mathcal{E})))$ for some pair of processes q and r. In other words, there are only finitely many r-events in $\partial_q(\partial_{p_C}(\mathcal{E}))$. There are two possibilities:

- The ideal $\partial_q(\partial_{p_C}(\mathcal{E}))$ is itself finite, in which case $q \in (\mathcal{P} - C)$.
- The ideal $\partial_q(\partial_{p_C}(\mathcal{E}))$ is infinite, but the number of r-events in $\partial_q(\partial_{p_C}(\mathcal{E}))$ is finite. This means that $q \in C$ but $r \in (\mathcal{P} - C)$.

This observation prompts the following definition:

Stable information Let $\alpha : \mathbf{N} \rightarrow \Sigma$ be an infinite word and let $p_C \in C$ for some connected component $C \in Conn_\alpha$. For any ideal I, the *stable information* of p_C in I, $stable\text{-}info_{p_C}(I)$ is the subset of $secondary(\partial_{p_C}(I))$ given by

$$\{max_r(\partial_q(\partial_{p_C}(I))) \mid q \notin C, r \in \mathcal{P}\} \cup \{max_r(\partial_q(\partial_{p_C}(I))) \mid q \in C, r \notin C\}.$$

The events in $stable\text{-}info_p(I)$ are frozen once I grows beyond the finite initial portion I_α^{joint} in $\mathcal{E}$. In other words, for any ideal $J \supseteq I_\alpha^{joint}$, $stable\text{-}info_{p_C}(J) = stable\text{-}info_{p_C}(\mathcal{E})$. By our earlier observations, this means that for any $J \supseteq I_\alpha^{joint}$, $stable\text{-}info_{p_C}(J)$ subsumes the events lying in the sets $\bigcup_{C' \in Conn_\alpha} frontier(\partial_C(\mathcal{E}), \partial_{C'}(\mathcal{E}))$.

Let us get back to our distributed version of Safra's construction corresponding to an entry $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C \in \mathcal{C}})$ in $\mathcal{T}_B$. Suppose that each signalling process p_C ensures that it has crossed the finite portion $\mathcal{I}_\alpha^{joint}$ before starting

Safra's construction. Then, along with each successful run r_C on $\partial_C(\mathcal{E})$ that it detects, it can record the value of r_C on *stable-info*$_{p_C}(\mathcal{E})$. If the successful runs $\{r_C\}_{C \in \mathcal{C}}$ agree on the stable information across all the signalling processes, we know that the runs satisfy the assumption of Lemma 6, which means that there is some global run of $\mathfrak{A}$ on α which satisfies τ.

The catch is that the signalling processes have no way of knowing when the finite portion I_α^{joint} is over. However, since $\mathfrak{B}_\tau$ includes the subset automaton for $\mathfrak{A}$, $\mathfrak{B}_\tau$ also incorporates the time-stamping automaton $\mathfrak{A}_T$ which maintains consistent labels across *tertiary*(I) at the end of any ideal I. If the time-stamps assigned by $\mathfrak{A}_T$ to the events in *stable-info*$_{p_C}(I)$ change, the process p_C knows that I_α^{joint} is *not* yet over.

So, we adopt the following strategy. Initially, each signalling process p_C starts off Safra's construction. Whenever it detects that *stable-info*$_{p_C}(I)$ has changed, it "kills" the old copy of Safra's construction and restarts a new copy. In fact, the process starts a separate copy of Safra's construction for each distinct history over *stable-info*$_{p_C}(I)$. So, in the limit, p_C can signal whether or not there is an accepting local run r_C for each history over its stable information.

The structure of $\mathbf{R}\mathfrak{B}_\tau$ Let $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C \in \mathcal{C}})$. The local state of each signalling process p_C in $\mathfrak{B}_\tau$ consists of the following information:

(i) The local state of the subset automaton for $\mathfrak{A}$. This includes the set H_{p_C} of all reachable histories over *secondary*$(\partial_{p_C}(I))$ at the end of any ideal I. This component incorporates the local state of the time-stamping automaton $\mathfrak{A}_T$, which stores the labels of events in *secondary*$(\partial_{p_C}(I))$ as a function $\lambda_{p_C} : \mathcal{P} \times \mathcal{P} \rightharpoonup \mathcal{L}$. The time-stamps assigned to the events in *stable-info*$_{p_C}(I)$ are the values $\lambda_p(q, r)$ where either $q \notin C$ or $(q \in C$ and $r \notin C)$.

(ii) Let H_S be the set of reachable histories over *stable-info*$_{p_C}(I)$. For each $h \in H_S$, p_C maintains an independent copy of Safra's construction.

The non-signalling processes need not run Safra's construction; it is sufficient for them to maintain the first component of the state.

On reading a letter a, each process p in $\theta(a)$ updates its local states as follows (where we have left out some important details due to a lack of space):

(i) First p updates the local state components corresponding to the time-stamping automaton and the subset automaton.

(ii) If p is a signalling process and if the time-stamps assigned to *stable-info*$_p(I)$ have not changed, then p updates each copy of Safra's construction using the new information provided by the subset automaton.
On the other hand, if the time-stamp corresponding to any event in *stable-info*$_p(I)$ changes, p erases all the existing copies of Safra's construction and begins a fresh copy for each history in the new set H_S.

The single entry τ in $\mathcal{T}_B$ generates a table $\mathcal{T}_{R_\tau}$ in $\mathbf{R}\mathfrak{B}_\tau$ with multiple entries. Each possible history h over $\bigcup_{C \in \mathcal{C}}$ *stable-info*$_{p_C}(\mathcal{E})$ generates a distinct entry τ_h

of the form $(\mathcal{C}, T, \{(p_C, pairs_C)\}_{C \in \mathcal{C}})$ in $\mathcal{T}_{R_\tau}$. In τ_h, the entries $\mathcal{C}$, T and the set of signalling processes $\{p_C\}_{C \in \mathcal{C}}$ are as in the original entry $\tau \in \mathcal{T}_B$.

The sequential version of Safra's construction uses labels $\{\ell_1, \ell_2, \ldots, \ell_{2M}\}$, where M is the number of states of the original automaton. The acceptance condition of the determinized automaton consists of pairs $\{(G^j, R^j)\}_{j \in [1..2m]}$, where each pair (G^j, R^j) expresses some constraints on the label ℓ_j.

In the distributed construction, each signalling process uses labels in the same manner as in the sequential case and generates the same number of acceptance pairs. Let $M_C = |V_{p_C}|$ be the number of possible local states for each signalling process p_C in $\mathfrak{A}$. Then $pairs_C = \{(G_C^j, R_C^j)\}_{j \in [1..2M_C]}$, where, for $j \in [1..2M_C]$, the sets G_C^j and R_C^j consist of all possible states of p_C in which label ℓ_j meets the criteria required by Safra's construction and, in addition, the set of histories stored in the state includes the projection h_{p_C} of h onto $stable\text{-}info_{p_C}(\mathcal{E})$.

It can then be verified that $\mathbf{R}\mathfrak{B}_\tau$ accepts an input $\alpha : \mathbf{N} \to \Sigma$ iff there is a run of $\mathbf{B}\mathfrak{A}$ on α satisfying τ.

Removing the progress assumption So far we have assumed that $T = \emptyset$ in the Büchi table entry τ. Suppose $T \neq \emptyset$ and there is a run of $\mathfrak{A}$ on α which satisfies τ. Then each process $p \in T$ moves only finitely often while $\mathfrak{A}$ reads α. So, we just run the subset construction for p and verify that it terminates in one of the states in G_p.

Combining the individual automata $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau \in \mathcal{T}_B}$ We can combine the individual automata $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau \in \mathcal{T}_B}$ using a standard product construction which preserves determinacy. The construction is essentially the same as in the sequential case and we omit the details.

A complementation construction Following the technique proposed by Vardi (described in [Saf]), we can use a non-deterministic Büchi asynchronous to efficiently simulate the complement of a deterministic Rabin asynchronous automaton. So, from $\mathbf{R}\mathfrak{B}$ we can construct an Büchi automaton $\mathbf{B}\overline{\mathfrak{A}}$ such that $\mathbf{B}\overline{\mathfrak{A}}$ accepts an infinite string α iff $\mathbf{R}\mathfrak{B}$ does not accept α. Since $\mathbf{R}\mathfrak{B}$ accepts the same inputs that $\mathbf{B}\mathfrak{A}$ does, $\mathbf{B}\overline{\mathfrak{A}}$ is a complement automaton for $\mathbf{B}\mathfrak{A}$.

Complexity analysis In the input automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$, let N be the number of processes in $\mathfrak{A}$, M the size of the largest set in the collection $\{V_p\}_{p \in \mathcal{P}}$ and K the number of entries in $\mathcal{T}_B$.

Then, in the deterministic Rabin automaton $\mathbf{R}\mathfrak{B}$ which we construct, the number of local states of each process p is bounded by $2^{KM^{O(N^3)}}$, while in the complement automaton $\mathbf{B}\overline{\mathfrak{A}}$, the number of local states of each process p is bounded by $2^{K^2 M^{O(N^4)}}$ (see [KMS2] for details).

In [KMS1], it is shown that in the subset automaton for $\mathfrak{A}$, the number of states of each process p is bounded by $2^{M^{O(N^3)}}$. So, the blow-up involved in the construction of $\mathbf{R}\mathfrak{B}$ and $\mathbf{B}\overline{\mathfrak{A}}$ is essentially that of the subset construction.

Consolidating the results of this section, we have our main result.

Theorem 8. *Let $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ be a non-deterministic Büchi asynchronous automaton over (Σ, θ). Then, we can construct a deterministic Rabin asynchronous*

automaton $\mathbf{R}\mathfrak{B} = (\mathfrak{B}, \mathcal{T}_R)$ *over* (Σ, θ) *such that* $\mathbf{R}\mathfrak{B}$ *accepts the same set of infinite strings that* $\mathbf{B}\mathfrak{A}$ *does. From* $\mathbf{R}\mathfrak{B}$*, we can construct a complementary non-deterministic Büchi automaton* $\mathbf{B}\overline{\mathfrak{A}}$ *over* (Σ, θ) *which accepts an infinite string* α *iff the original automaton* $\mathbf{B}\overline{\mathfrak{A}}$ *does not accept* α*.*

The number of local states of each process in $\mathbf{R}\mathfrak{B}$ *and* $\mathbf{B}\overline{\mathfrak{A}}$ *is essentially exponential in the number of global states of the original automaton* $\mathbf{B}\mathfrak{A}$*.*

References

[DM] V. Diekert, A. Muscholl: Deterministic asynchronous automata for infinite traces, *Acta Inf.*, **31** (1994) 379–397.

[EM] W. Ebinger, A. Muscholl: Logical definability on infinite traces, *Proc. ICALP '93*, *LNCS* **700** (1993) 335–346.

[GP] P. Gastin, A. Petit: Asynchronous cellular automata for infinite traces, *Proc. ICALP '92*, *LNCS* **623** (1992) 583–594.

[HJJ] J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, A. Sandholm: Mona: Monadic Second-order logic in practice, *Report RS-95-21*, BRICS, Department of Computer Science, Aarhus University, Aarhus, Denmark (1995).

[Kla] N. Klarlund: Progress measures for complementation of ω-automata with applications to temporal logic, *Proc. 32nd IEEE FOCS*, (1991) 358–367.

[KMS1] N. Klarlund, M. Mukund, M. Sohoni: Determinizing asynchronous automata, *Proc. ICALP '94*, *LNCS* **820** (1994) 130–141.

[KMS2] N. Klarlund, M. Mukund, M. Sohoni: Determinizing asynchronous automata on infinite traces, *Report TCS-95-6*, School of Mathematics, SPIC Science Foundation, Madras (1995).

[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, *Proc. FST&TCS '93, LNCS* **761** (1993) 388–399.

[Mus] A. Muscholl: On the complementation of Büchi asynchronous cellular automata, *Proc. ICALP '94*, *LNCS* **820** (1994) 142–153.

[Rab] M.O. Rabin: Decidability of second order theories and automata on infinite trees, *Trans. AMS*, **141**(1969) 1–37.

[Saf] S. Safra: On the complexity of ω-automata, *Proc. 29th IEEE FOCS*, (1988) 319–327.

[Thi] P.S. Thiagarajan: TrPTL: A trace based extension of linear time temporal logic, *Proc. 9th IEEE LICS*, (1994) 438–447.

[Tho] W. Thomas: Automata on infinite objects, in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science, Volume B*, North-Holland, Amsterdam (1990) 133–191.

[VW] M. Vardi, P. Wolper: An automata theoretic approach to automatic program verification, *Proc. 1st IEEE LICS*, (1986) 332–345.

[Zie1] W. Zielonka: Notes on finite asynchronous automata, *R.A.I.R.O.—Inf. Théor. et Appl.*, **21** (1987) 99–135.

[Zie2] W. Zielonka: Safe executions of recognizable trace languages, in *Logic at Botik*, *LNCS* **363** (1989) 278–289.