Internal Report TCS-95-6
October, 1995

Determinizing Asynchronous Automata on Infinite Inputs

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

Abstract

Asynchronous automata are a natural distributed machine model for recognizing *trace languages*—languages defined over an alphabet equipped with an independence relation.

To handle *infinite* traces, Gastin and Petit introduced Büchi asynchronous automata, which accept precisely the class of 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.

Subsequently, Diekert and Muscholl solved the complementation problem by showing that with a Muller acceptance condition, deterministic automata suffice for recognizing ω-regular trace languages. However, a direct determinization procedure, extending the classical subset construction, has proved elusive.

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.

[1]BRICS Centre, Aarhus University, Ny Munkegade, DK 8000 Aarhus C, Denmark. E-mail: klarlund@daimi.aau.dk. The author is supported by a fellowship from the Danish Research Council.

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

[3]Department of Computer Science and Engineering, Indian Institute of Technology, Bombay 400 076, India. E-mail: sohoni@cse.iitb.ernet.in.

Introduction

Finite-state automata are, by definition, sequential. To describe finite-state *concurrent* computations, Zielonka introduced *asynchronous automata* [Ziel]. 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.[4] 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 comes equipped with a pre-specified independence relation I, describing the concurrency in the system [Maz]. In this setting, 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 said to be 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.

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, which in logic is the equivalent of replacing a second-order existential quantifier with a second-order universal quantifier, and determinization, which is equivalent to replacing a second-order quantifier by first-order quantifiers.

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 [KMS] 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 easily complement the determinized automaton by viewing the Rabin condition as a Streett condition. This Streett automaton can then be converted efficiently into a non-deterministic Büchi asynchronous automaton. So, we also have a

[4] 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.

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 [KMS, MS] for manipulating asynchronous automata. In Section 6 we show how to apply these techniques to determinize Büchi asynchronous automata. To preserve continuity, some detailed explanations, examples and proofs have been moved from the main text into separate Appendices.

1 Preliminaries

The following definitions are essentially those of [KMS] 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 $(\{V_p\}_{p \in \mathcal{P}}, \{\to_a\}_{a \in \Sigma}, \mathcal{V}_0)$, where $\to_a \subseteq V_a \times V_a$ is the *local transition relation* for a, and $\mathcal{V}_0 \subseteq V_{\mathcal{P}}$ is a set of *initial* global states. Each relation $\to_a$ 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 $\vec{v}_a \to_a \vec{v}'_a$ 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., $\alpha : \mathbf{N} \to \Sigma$. (We use $\mathbf{N}$ to denote the set $\{1, 2, \ldots\}$ and $\mathbf{N}_0$ for $\{0, 1, 2, \ldots\}$.) We shall also deal with finite words over Σ. Let $u \in \Sigma^*$ be a word of length m. We denote u as a function $u : [1..m] \to \Sigma$, where $[i..j]$ abbreviates the set $\{i, i{+}1, \ldots, j\}$.

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

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

2 Asynchronous automata on infinite inputs

To define how an asynchronous automaton accepts an infinite input α, we have to analyze the communication pattern between processes in the limit, as α is being processed.

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]. We discuss the relationship between the two definitions in Appendix A. 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.

As in the case of Büchi automata on infinite strings, non-deterministic Büchi asynchronous automata are strictly more powerful than their deterministic counterparts [GP]. So, 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 [Saf]—see Appendix B for a brief sketch of the construction. 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, $\rho_{p_C} \cap G_C \neq \emptyset$. To do this, we run Safra's construction for each signalling process p_C, using the subset construction for asynchronous automata [KMS] 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 components *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]. Appendix C has a few examples illustrating the ideas introduced in the next couple of sections.

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 $\to_e$, which is just $\to_a$. For $p \in \mathcal{P}$ and $e = (i, a)$, we write $p \in e$ to denote that $p \in \theta(a)$ when $e \neq 0$; for $e = 0$, 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 is 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 $e \lessdot_p f$. Thus $e \lessdot_p f$ 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 $\{\lessdot_p\}_{p\in\mathcal{P}}$ than by the temporal order $\leq$. We say that e is an immediate predecessor of f and write $e \lessdot f$ if $e \lessdot_p f$ for some p. Let $\sqsubseteq$ be the reflexive and transitive closure of $\lessdot$. 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 $\alpha[I] : [1..m] \rightarrow \Sigma$ 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 $\{e \in I \mid \exists f \in I.\ p \in f$ 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 maximum p-event in I with respect to $\sqsubseteq$.

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., $nbd(e) = \{e\} \cup \{f \mid f \lessdot e\}$. 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 $f_p \lessdot_p e$. 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:

$$\begin{aligned} dom(h) &= \{e \in J \mid \text{ for all } j \in [1..n], \text{ if } e \in I_j \text{ then } e \in dom(h_j)\} \\ h(e) &= h_k(e), \text{where } k \text{ is such that } e \in dom(h_k) \text{ (the choice of } k \text{ does not} \\ &\quad\ \text{matter since } \{h_j\}_{j\in[1..n]} \text{ is consistent).} \end{aligned}$$

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 can 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 as before, 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 can 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 [KMS]. When determinizing an asynchronous automaton, it is not sufficient for each process to maintain just the subset of states it can be in after reading a part of the input. 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 [KMS], after 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 after reading a part of the input. 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 [KMS].

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 of events $\bigcup_{p,q \in \mathcal{P}} primary(\partial_p(\partial_q(I)))$.

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 $tertiary(I)$. (Actually, $primary(I)$, $secondary(I)$ and $tertiary(I)$ are *indexed* sets of events—an event $e \in I$ may be both $max_p(I)$ and $max_q(I)$ for different processes p and q and must hence be represented *twice* in $primary(I)$, say as the pairs (e,p) and (e,q). However, we shall normally ignore this aspect and just treat all these indexed collections as sets of events.)

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. (A proof of the lemma can be found in [KMS].)

Lemma 3 *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 $\mathcal{L}$ of labels to *time-stamp* each event in this set. We can denote the assignment of time-stamps to the events in $tertiary(I)$ as a function $\lambda : tertiary(I) \to \mathcal{L}$. 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 $\mathcal{L}$ and construct a deterministic asynchronous automaton $\mathfrak{A}_T$ over (Σ, θ) in which, on any finite ideal I, each process p maintains $\lambda_p : secondary(\partial_p(I)) \to \mathcal{L}$, where λ_p is the restriction to $\partial_p(I)$ of a consistent labelling $\lambda : tertiary(I) \to \mathcal{L}$. Process p maintains λ_p as a function from $\mathcal{P} \times \mathcal{P}$ to $\mathcal{L}$. 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 $\mathcal{L}$ 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 [KMS]) *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 the set of all possible P-states of $\mathfrak{A}$ on I from the information in the P-state $\vec{v}_P$. In particular, from $\vec{v}$ we can recover the set of 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 r_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 $\{r_C\}_{C \in \mathcal{C}}$ 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 $\{e \in I_C \mid \exists f \in \mathit{frontier}(I_C, \mathcal{I}_\alpha).\ f \sqsubseteq e\}$. 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 $\alpha : \mathbf{N} \to \Sigma$ 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 $\alpha : \mathbf{N} \to \Sigma$ 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 $stable\text{-}info_{p_{_C}}(\mathcal{E})$. 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 $stable\text{-}info_{p_{_C}}(I)$ 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 $stable\text{-}info_{p_{_C}}(I)$ 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 $stable\text{-}info_{p_{_C}}(I)$. 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 input 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 $\lambda_{p_{_C}} : \mathcal{P} \times \mathcal{P} \to \mathcal{L}$. The time-stamps assigned to the events in $stable\text{-}info_{p_{_C}}(I)$ 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 $stable\text{-}info_{p_{_C}}(I)$. For each $h \in H_S$, $p_{_C}$ maintains an independent labelled coloured tree as dictated by 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, corresponding to the subset automaton.

On reading an input letter a, each process p in $\theta(a)$ updates its local states as follows:

(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 $\mathit{stable\text{-}info}_p(I)$ have not changed, then p updates the trees in 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 $\mathit{stable\text{-}info}_p(I)$ 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 $\bigcup_{C\in\mathcal{C}} \mathit{stable\text{-}info}_{p_C}(\mathcal{E})$ generates a distinct entry τ_h of the form $(\mathcal{C}, T, \{(p_C, \mathit{pairs}_C)\}_{C\in\mathcal{C}})$ 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 $\tau \in \mathcal{T}_B$.

Let $|V_{p_C}| = M_C$ be the number of possible local states for p_C in $\mathfrak{A}$. Then, there are $2M_C$ possible pairs (G_C^j, R_C^j) in the list pairs_C, corresponding to the labels $\{\ell_1, \ell_2, \ldots, \ell_{2M_C}\}$ used by Safra's construction to label the nodes in the tree associated with the current subset of possible p_C states.

The set G_C^j consists of all possible states of p_C in which the set of histories over $\mathit{stable\text{-}info}_{p_C}(\mathcal{E})$ includes the projection h_{p_C} of h onto $\mathit{stable\text{-}info}_{p_C}(\mathcal{E})$ and, moreover, in the labelled tree from Safra's construction corresponding to h_{p_C}, the node labelled ℓ_j is coloured green.

The set R_C^j consists of all possible states of p_C in which the set of histories over $\mathit{stable\text{-}info}_{p_C}(\mathcal{E})$ includes the projection h_{p_C} of h onto $\mathit{stable\text{-}info}_{p_C}(\mathcal{E})$ and, moreover, in the labelled tree from Safra's construction corresponding to h_{p_C}, there is no node labelled ℓ_j.

It is straightforward though tedious to verify 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.

In other words, for each entry τ_h of $\mathbf{R}\mathfrak{B}_\tau$, we have a single pair (G_p^1, R_p^1) in $\mathit{pairs}_{\{p\}}$, where $R_p^1 = \emptyset$ and G_p^1 consists of all possible states of the subset automaton such that there is a history h' in H_p which agrees with h on $\mathit{stable\text{-}info}_p(\mathcal{E})$ where the terminal state assigned to p by h' belongs to 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 We can complement a Rabin asynchronous automaton by viewing the acceptance table as a Streett condition, as in sequential automata [Tho]. This Streett condition can then be checked efficiently by a non-deterministic Büchi asynchronous automaton using the technique proposed by Vardi (described in [Saf]). 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 $\mathbf{B}\mathfrak{A}$. See Appendix D for details of how to construct $\mathbf{B}\overline{\mathfrak{A}}$.

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 $2^{K^2M^{O(N^4)}}$. Details of how these bounds are derived can be found in Appendix E.

In [KMS], 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 the same as that of the subset construction.

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

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

[CMZ] R. Cori, Y. Metivier, W. Zielonka: Asynchronous mappings and asynchronous cellular automata, *Inf. and Comput.*, **106** (1993) 159–202.

[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.

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

[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: Gossiping, asynchronous automata and Zielonka's theorem, *Report TCS-94-2*, School of Mathematics, SPIC Science Foundation, Madras (1994). See also "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.

[Nie] P. Niebert: A μ-calculus with local views for systems of sequential agents, *Proc. MFCS '95*, to appear.

[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, P. 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.

A Comparison with Gastin-Petit automata

The original formulation of Büchi asynchronous automata by Gastin and Petit [GP] differs from the one we use in this paper. In the original definition, information about $Conn_\alpha$ and $Finite_\alpha$ is not part of the acceptance condition. Instead, in the acceptance table, each entry is a tuple of the form $\{G_p\}_{p\in\mathcal{P}}$. A run ρ is accepting if there is some table entry $\{G_p\}_{p\in\mathcal{P}}$ such that $inf(\rho_p) \supseteq G_p$ for each process p. It is not difficult to verify that our automata are equivalent to those of Gastin and Petit.

Notice that the Gastin-Petit acceptance table uses a "covering" condition on the sets G_p, rather than the simple recurring set condition we have in our automata. This covering condition is awkward to work with when trying to determinize these automata.

Actually, to eliminate the "covering" condition it is sufficient to record information about $Finite_\alpha$ in the acceptance table [Nie]. The extra information we record about $Conn_\alpha$ 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 unit, independent of the others.

From Gastin-Petit automata to our automata ...

We can use our automata to simulate Büchi automata with Gastin-Petit acceptance tables as follows. On reading an input α, the simulating automaton guesses a partition $\mathcal{C}$ of $\mathcal{P}$ corresponding to $Conn_\alpha$ together with a set of processes T corresponding to $Finite_\alpha$. It also guesses a table entry $\tau = \{G_p\}_{p\in\mathcal{P}}$ from the Gastin-Petit table. It then generates a run ρ of the original automaton on the input α and checks if ρ satisfies τ.

To do this, we fix a signalling process $p_C \in C$ for each component $C \in \mathcal{C}$. The process p_C has to verify that every other process $p' \in C$ meets every state in $G_{p'}$ infinitely often along the run being simulated. This can be achieved by organizing the processes in C in a spanning tree rooted at p_C, where the edges in the tree come from the graph $\mathcal{G}_\alpha$.

Each process p_ℓ corresponding to a leaf in this spanning tree can check that it meets G_{p_ℓ} infinitely often using a simple counter. Each time p_ℓ cycles through G_{p_ℓ} completely, it informs its parent in the tree—if $\mathcal{C} = Conn_\alpha$, p_ℓ will synchronize infinitely often with its parent and so will pass on this signal infinitely often.

Each internal process p_i in the tree maintains two counters: one for itself and one correponding to its children. The first counter checks that p_i cycles through G_{p_i} infinitely often. The second counter checks that all the children report success infinitely often. Each time both counters complete a cycle, p_i informs its parent in the tree.

Eventually, all the information propagating up the tree reaches the root, which is the signalling process p_C. So, p_C can use a simple recurring Büchi condition to check that all the processes in C meet their "covering" Büchi condition corresponding to the table entry τ.

The blow-up in this simulation corresponds to guessing a partition $\mathcal{C}$, a set T and a table entry τ. The number of choices of $\mathcal{C}$ is bounded by 2^{N^2}, where N is the number of processes—each choice of $\mathcal{C}$ corresponds to dropping some edges from the complete graph on N vertices, which has N^2 edges. The number of choices for T is bounded by 2^N, while the choice of τ depends on K, the number of entries in the original table. Thus, overall the blow-up is $O(K2^{N^2})$.

(Actually, we can do better. The connectivity we begin with does not correspond, in general, to the complete graph on N vertices. The alphabet (Σ, θ) restricts the communication pattern in the system. So, the number of different ways of partitioning the processes in $\mathcal{P}$ is actually bounded by $2^{|E|}$, where E is the number of edges in the "connectivity graph" induced by (Σ, θ). The overall blow-up is then $O(K2^{|E|})$.)

... and back

The simulation in the other direction is as follows. Let $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$ be one of our Büchi asynchronous automata. At the initial state, the simulating automaton guesses an entry $\tau = (\mathcal{C}, T, \{(p_C, G_C)\}_{C \in \mathcal{C}})$ in $\mathcal{T}_B$ and sees if $\mathbf{B}\mathfrak{A}$ has a run on the input which satisfies τ.

The simulating automaton has to check the following:

- While reading the input, each process in T must move only finitely often, while each process outside T must move infinitely often.
- Each component $C \in \mathcal{C}$ must actually constitute a connected component in the limit graph of the input.
- For each component $C \in \mathcal{C}$, the signalling process p_C must visit G_C infinitely often along the run.

The first condition can be checked by tagging the state space of each process with a two element counter.

For processes in T, one of the counter values denotes that it is "live" and the other that it is "dead"—in other words, that it will not make any further moves. There are no transitions enabled from the "dead" part of the state space. The simulating automaton non-deterministically sends each terminated process into a dead state when it feels that it has finished its finite quota of moves over a particular input.

For processes not in T, the simulating automaton ensures that with each move, the process switches between the counter values. So, the simulating automaton can record whether or not a process moves infinitely often in the original automaton by checking that it visits both copies of the state space infinitely often. Notice that to check this, the simulating automaton *must* use a covering Büchi condition—in general, a simple recurring Büchi condition will not suffice.

To check the second condition, the simulating automaton keeps track of when the gossip information of each process changes. Let p belong to a component $C \in \mathcal{C}$. Then, while reading the input, p's primary information about other processes in C must change infinitely often, while p's primary information about processes outside C becomes frozen at some stage. So, p non-deterministically guesses when the primary information of all processes outside C is frozen. After this, if p ever has to update it primary information for some q outside C, it moves into a reject state.

After making this guess, p uses a counter to cycle through the processes in C, waiting for its primary information about each process to change. When it completes each cycle, it goes into a "good" state.

It is easy to verify that C is a connected component of the limit graph iff p does not go into its reject state and, in addition, p visits its "good" state infinitely often.

Finally, each signalling process p_C can easily check on the side that it visits some state from G_C infinitely often.

The blow-up in the state space is linear in the size of $\mathcal{T}_B$—each entry of the table generates a copy of the original state space together with at most two counters, one of size two and one whose size is bounded by N, the number of processes in the system. However, the number of entries in the new table could be exponential in the size of $\mathcal{T}_B$ since recurring Büchi conditions in the original automaton have to be encoded in terms of covering Büchi conditions in the simulating automaton.

B Safra's construction

Let BA be a non-deterministic sequential Büchi automaton with n states and a set G of recurring states. Safra constructs a deterministic sequential Rabin automaton RA with a table $\{(G_j, R_j)\}_{j\in[1..2n]}$ such that RA accepts an input α iff there is some run of BA passing through G infinitely often.

Safra uses the classical subset construction of Rabin and Scott to record the possible states that BA can be in after reading any finite prefix of α. The subset of possible states is maintained as a labelled tree—the elements of the subset are in 1–1 correspondence with the nodes of the tree and each node has a distinct label drawn from a set of names of size $2n$. In addition, each node of the tree is coloured either white or green.

After reading a letter from the input, the subset construction updates the set of possible current states of BA. As a result, the shape of the corresponding tree changes. In the process some nodes are discarded from the tree and some new nodes are added. However, Safra's construction ensures that the labels associated with the old nodes are not reused in the same step for the new nodes. In other words, if a node is dropped at some stage from the tree, its label temporarily disappears from the tree.

These labelled trees are used to keep track of runs in the underlying computation. The colours white and green are used to signal when a run of BA visits the recurring states G. Thus, when updating the trees, the colours of the nodes may also change. When a node is added to the tree, it is assigned the colour white. During its "lifetime" in the tree, it may periodically change colour from white to green and back to white again.

Recall that nodes are labelled using a set of names of size $2n$, say $\{\ell_1, \ell_2, \ldots, \ell_{2n}\}$. Let us look at the entry (G_j, R_j) in the table for RA. Condition R_j says that the label ℓ_j disappears from the tree only finitely often. In other words, a node with label ℓ_j is added to the tree at some point and this node is never deleted during the rest of the run of RA. Condition G_j then guarantees that this node turns green infinitely often.

C Examples

Consider the word $\alpha = bacbacd^\omega$ over the alphabet (Σ, θ) for $\mathcal{P} = \{p, q, r, s\}$, where $\Sigma = \{a, b, c, d\}$ and $\theta(a) = \{p, q\}$, $\theta(b) = \{q, r\}$, $\theta(c) = \{r, s\}$ and $\theta(d) = \{p\}$. The set of events $\mathcal{E}_\alpha$ is then $\{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, \ldots\} = \{0, (1, b), (2, a), (3, c), (4, b), (5, a), (6, c), (7, d), (8, d), \ldots\}$.

Figure 1 describes $(\mathcal{E}_\alpha, \sqsubseteq)$. The arrows between the events indicate the relations $\lessdot_p$, $\lessdot_q$, $\lessdot_r$ and $\lessdot_s$. For example, $e_0 \lessdot_r e_1$ holds, but $e_0 \lessdot_p e_1$ does not hold.

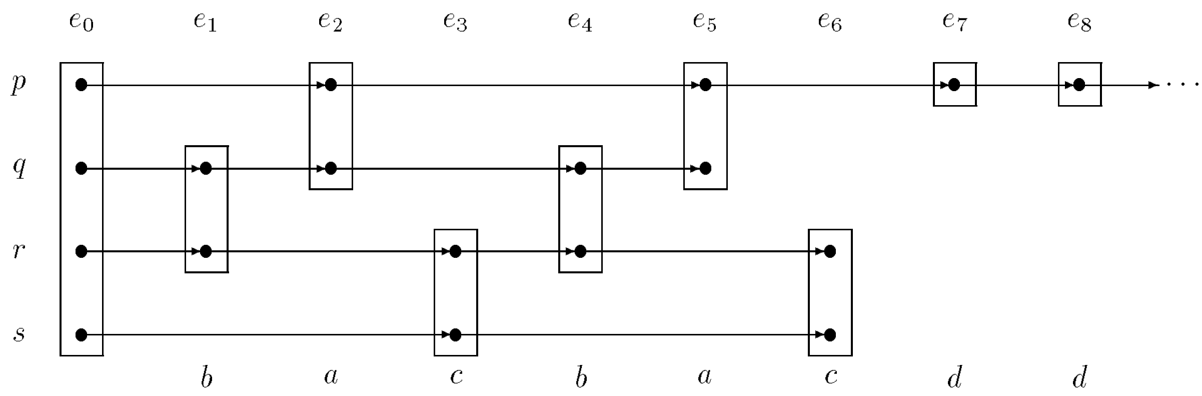

Figure 1: An example

Ideals and views The set of events $e_7{\downarrow}$ is $\{e_0, e_1, e_2, e_3, e_4, e_5, e_7\}$ while $e_6{\downarrow}$ is $\{e_0, e_1, e_2, e_3, e_4, e_6\}$.

In this example, $\partial_r(\mathcal{E}) = max_r(\mathcal{E}){\downarrow} = e_6{\downarrow} = \{e_0, e_1, e_2, e_3, e_4, e_6\}$. On the other hand, $\partial_p(\mathcal{E}) = \{e_0, e_1, e_2, e_3, e_4, e_5, e_7, e_8 \ldots\} = \mathcal{E} - \{e_6\}$. Notice that $max_p(\mathcal{E})$ is undefined.

Neighbourhoods The neighbourhood of e_5, $nbd(e_5)$, is $\{e_2, e_4, e_5\}$.

Let $\mathfrak{A} = (\{V_p\}_{p \in \mathcal{P}}, \{\rightarrow_a\}_{a \in \Sigma}, \mathcal{V}_0)$ be an asynchronous automaton over (Σ, θ). Each process has four local states. Thus, $V_p = \{1_p, 2_p, 3_p, 4_p\}$, $V_q = \{1_q, 2_q, 3_q, 4_q\}$ etc.

Let the local transition relations of $\mathfrak{A}$ be defined as in the table below:

$\rightarrow_a$	$\rightarrow_b$	$\rightarrow_c$	$\rightarrow_d$
$\{(\langle 1_p, 2_q\rangle, \langle 3_p, 3_q\rangle)$	$\{(\langle 1_q, 1_r\rangle, \langle 2_q, 2_r\rangle)$	$\{(\langle 2_r, 1_s\rangle, \langle 4_r, 4_s\rangle)$	$\{(\langle 3_p\rangle, \langle 4_p\rangle)$
$(\langle 1_p, 3_q\rangle, \langle 4_p, 4_q\rangle)$	$(\langle 1_q, 1_r\rangle, \langle 3_q, 3_r\rangle)$	$(\langle 2_r, 3_s\rangle, \langle 4_r, 4_s\rangle)$	$(\langle 4_p\rangle, \langle 3_p\rangle)\}$
$(\langle 3_p, 2_q\rangle, \langle 4_p, 4_q\rangle)$	$(\langle 3_q, 4_r\rangle, \langle 2_q, 2_r\rangle)$	$(\langle 2_r, 4_s\rangle, \langle 1_r, 1_s\rangle)$	
$(\langle 4_p, 2_q\rangle, \langle 3_p, 3_q\rangle)\}$	$(\langle 4_q, 3_r\rangle, \langle 2_q, 2_r\rangle)\}$	$(\langle 3_r, 1_s\rangle, \langle 3_r, 3_s\rangle)\}$	

$\mathcal{V}_0 = \{\langle 1_p, 1_q, 1_r, 1_s\rangle\}$ and $\mathcal{V}_F = \{\langle 4_p, 3_q, 1_r, 1_s\rangle, \langle 3_p, 4_q, 4_r, 4_s\rangle\}$.

Local runs Let I be the ideal $\{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$. Then, the local runs corresponding to the only two possible global runs of $\mathfrak{A}$ on I are shown in Figure 2. The left half of each event is labelled by the first run and the right half by the second run.

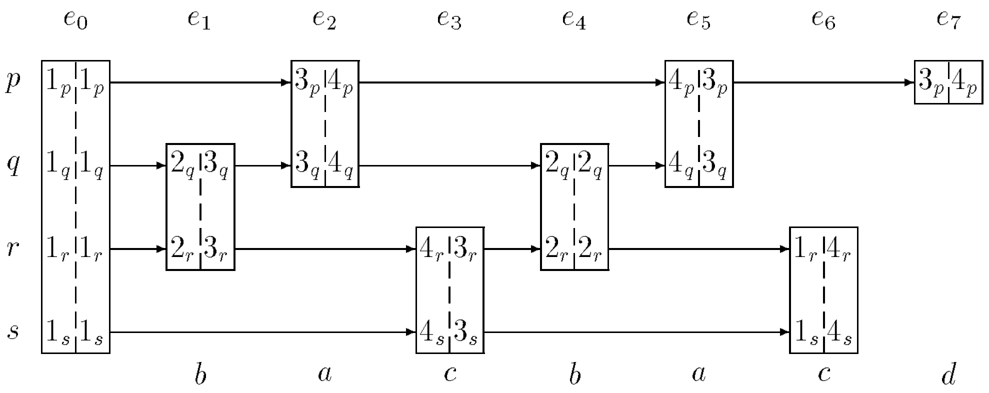

Figure 2: Local runs

Histories and products For p, let h_p^1 and h_p^2 be two histories over $\partial_p(I)$ where
$h_p^1 = \{(e_3 \mapsto \langle 4_r, 4_s\rangle, e_4 \mapsto \langle 2_q, 2_r\rangle, e_5 \mapsto \langle 4_p, 4_q\rangle, e_7 \mapsto \langle 3_p\rangle)\}$, and
$h_p^2 = \{(e_3 \mapsto \langle 3_r, 3_s\rangle, e_4 \mapsto \langle 2_q, 2_r\rangle, e_5 \mapsto \langle 3_p, 3_q\rangle, e_7 \mapsto \langle 4_p\rangle)\}$.
For s, let h_s^1 and h_s^2 be two histories over $\partial_s(I)$ where
$h_s^1 = \{(e_2 \mapsto \langle 3_p, 3_q\rangle, e_4 \mapsto \langle 2_q, 2_r\rangle, e_6 \mapsto \langle 4_r, 4_s\rangle)\}$, and
$h_s^2 = \{(e_2 \mapsto \langle 4_p, 4_q\rangle, e_4 \mapsto \langle 2_q, 2_r\rangle, e_6 \mapsto \langle 1_r, 1_s\rangle)\}$.
All four of these histories are reachable. The product $\bigotimes\left\{\{h_p^1, h_p^2\}, \{h_s^1, h_s^2\}\right\}$ generates four possible runs over $\partial_{\{p,s\}}(I)$. However, only two of these runs are reachable—those generated by $h_p^1 \otimes h_s^1$ and $h_p^2 \otimes h_s^2$. The "bad" entry $h_p^2 \otimes h_s^1$ implies that $\langle 4_p, 3_q, 1_r, 1_s\rangle$ is a valid global state of $\mathfrak{A}$ after I, which is not the case.

D Streett asynchronous automata and complementation

Let $\mathbf{R}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_R)$ be a Rabin asynchronous automaton. Then, it is easy to verify that $\mathbf{R}\mathfrak{A}$ does *not* accept an input α iff for every run ρ of $\mathfrak{A}$ on α and for every entry $\tau = (\mathcal{C}, T, \{(p_C, pairs_C)\}_{C\in\mathcal{C}})$ in $\mathcal{T}R$, the following holds:

If $\mathcal{C} = Conn_\alpha$ and $T = Finite_\alpha$
then $\exists C \in \mathcal{C}.\ \forall i \in [1..k_C].\ (inf_{p_C}(\rho) \cap G_{p_C} \neq \emptyset) \Rightarrow (inf_{p_C}(\rho) \cap R_{p_C} \neq \emptyset)$.

This corresponds to a "complemented pairs" condition, first investigated by Streett in the setting of automata over infinite strings [Tho].

So, we can formally define a Streett asynchronous automaton as a pair $\mathbf{S}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_S)$ where the table $\mathcal{T}_S$ has the same structure as a Rabin table—i.e., each entry τ in $\mathcal{T}_S$ is of the form $(\mathcal{C}, T, \{(p_C, pairs_C)\}_{C\in\mathcal{C}})$.

A run ρ of $\mathfrak{A}$ on α satisfies τ provided it meets the condition described above. The automaton $\mathfrak{A}$ accepts α if there is a run ρ on α which satisfies *every* table entry in $\mathcal{T}_S$.

(Actually, due to the extra quantification over the partitions in $\mathcal{C}$, this definition of a Streett asynchronous automaton is perhaps not the natural one. A more intuitive condition for a run ρ over α to satisfy a table entry $\tau = (\mathcal{C}, T, \{(p_C, pairs_C)\}_{C\in\mathcal{C}})$ is to stipulate that the following holds:

If $\mathcal{C} = Conn_\alpha$ and $T = Finite_\alpha$
then $\forall C \in \mathcal{C}.\ \forall i \in [1..k_C].\ (inf_{p_C}(\rho) \cap G_{p_C} \neq \emptyset) \Rightarrow (inf_{p_C}(\rho) \cap R_{p_C} \neq \emptyset)$.

This definition is *not* complementary to that of Rabin acceptance in our setting. Since our primary goal is to complement the given Rabin automaton, we shall stick to our "unnatural" definition of a Streett automaton.)

To complement Büchi asynchronous automata using our determinization construction, it suffices to be able to simulate a *deterministic* Streett asynchronous automaton by a Büchi asynchronous automaton. Starting with a non-deterministic Büchi asynchronous automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathcal{T}_B)$, determinization produces an equivalent Rabin asynchronous automaton $\mathbf{R}\mathfrak{B} = (\mathfrak{B}, \mathcal{T}_R)$. Since $\mathbf{R}\mathfrak{B}$ is deterministic, if we interpret $\mathcal{T}_R$ as a Streett table $\mathcal{T}_S$, we get a deterministic Streett automaton $\mathbf{S}\mathfrak{B} = (\mathfrak{B}, \mathcal{T}_S)$ which is complementary to $\mathbf{B}\mathfrak{A}$. We can then simulate $\mathbf{S}\mathfrak{B}$ using a non-deterministic Büchi asynchronous automaton to obtain the complementary automaton $\mathbf{B}\overline{\mathfrak{A}}$ which we set out to construct.

To simulate a deterministic Streett asynchronous automaton $\mathbf{S}\mathfrak{A}$ by a non-deterministic Büchi asynchronous automata $\mathbf{B}\mathfrak{B}$, we proceed as follows:

(i) The simulating automaton should accept an input α if there is no entry $\tau = (\mathcal{C}, T, \{(p_C, pairs_C)\}_{C\in\mathcal{C}})$ for which $\mathcal{C} = Conn_\alpha$ and $T = Finite_\alpha$. This can be achieved using an automaton which has just one state for each process along with one table entry for each pair $(\mathcal{C}, T)$ which does not occur in the table for $\mathbf{S}\mathfrak{A}$. In all these new table entries, the Büchi condition for each process is the trivial one.

(ii) If the first case does not hold, let $X_\alpha = \{\tau_1, \tau_2, \ldots, \tau_m\}$ be the set of all entries in $\mathcal{T}_S$ such that each entry $\tau_i \in X_\alpha$ is of the form $(\mathcal{C}, T, \{(p_C, pairs_C)\}_{C\in\mathcal{C}})$ where $\mathcal{C} = Conn_\alpha$ and $T = Finite_\alpha$.

In [Saf], Safra describes an elegant construction due to Vardi for simulating a (sequential) Streett automaton by a (sequential) Büchi automaton. For each entry $\tau_i \in X_\alpha$, we construct a Büchi asynchronous automaton $\mathbf{B}\mathfrak{B}_i$ in which each signalling process p_C runs Vardi's construction to check if the (unique) run of the original automaton on α satisfies all the complemented pairs $\{(G_C^j, R_C^j)\}_{j\in[1..k_C]}$.

The acceptance table for $\mathbf{B}\mathfrak{B}_i$ has one entry τ_C for each $C \in \mathcal{C}$. All entries in the new table have the same partition $\mathcal{C}$ and set of terminated processes T as in τ_i. The difference between the entries lies in the acceptance condition—the recurring set for p_C in the entry τ_C checks that p_C satisfies its complemented pairs condition in the original automaton. The recurring condition in τ_C for signalling processes $p_{C'}$ corresponding to components $C' \neq C$ is the trivial one.

It is straightforward to see that $\mathbf{B}\mathfrak{B}_i$ accepts an input α provided the (unique) run ρ_α of $\mathbf{S}\mathfrak{A}$ on α satisfies τ_i. Notice that ρ_α trivially satisfies all table entries not in X_α. We can check that ρ_α satisfies all the entries in X_α by intersecting the automata $\{\mathbf{B}\mathfrak{B}_i\}_{i\in[1..m]}$. (We omit the details of how to intersect two Büchi asynchronous automata.)

Overall, we can partition $\mathcal{T}_S$ according to the values of $\mathcal{C}$ and T in the entries. For each pair $(\mathcal{C}, T)$, we can then construct a Büchi asynchronous automaton as described in step (ii) above. To get a single Büchi automaton simulating $\mathbf{S}\mathfrak{A}$, we take the union of all the automata constructed in this fashion together with the automaton of step (i) which catches all inputs which do not match any of the table entries.

E Complexity analysis

Determinization In the input automaton $\mathbf{B}\mathfrak{A} = (\mathfrak{A}, \mathfrak{T}_B)$, 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 $\mathfrak{T}_B$.

We first estimate the number of bits required to describe the state of a signalling process p_C in $\mathbf{R}\mathfrak{B}_\tau$ corresponding to a single entry $\tau \in \mathfrak{T}_B$. This state consists of the state of the subset automaton for $\mathfrak{A}$ together with one copy of the labelled tree used by Safra's construction for each history in H_S, the set of histories over $\textit{stable-info}_{p_C}(I)$.

The state of the subset automaton for $\mathfrak{A}$ can be written down using $M^{O(N^3)}$ bits. Each labelled tree maintained by Safra's construction requires $O(M \log M)$ bits [Saf]. In [KMS], it is shown that there are at most $M^{O(N^3)}$ elements in H_{p_C}, the set of all reachable histories over $\textit{secondary}(\partial_{p_C}(I))$. Since this is also a bound on $|H_S|$, p_C maintains at most $M^{O(N^3)}$ copies of Safra's construction and, overall, the state of p_C can be written down using $M^{O(N^3)} \cdot O(M \log M) = M^{O(N^3)}$ bits. Thus, the number of local states of p_C in $\mathbf{R}\mathfrak{B}_\tau$ is $2^{M^{O(N^3)}}$, which is essentially the same as in the subset construction of [KMS].

When combining the individual automata in $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau\in\mathfrak{T}_B}$, we take the K-fold product of the state space of each individual process p. So, overall, the number of local states of a process is $2^{KM^{O(N^3)}}$.

The number of entries in the Rabin table $\mathfrak{T}_R$ of the final deterministic automaton $\mathbf{R}\mathfrak{B} = (\mathfrak{B}, \mathfrak{T}_R)$ is $KM^{O(N^4)}$: each individual automaton $\mathbf{R}\mathfrak{B}_\tau = (\mathfrak{B}_\tau, \mathfrak{T}_{R_\tau})$ has as many entries in $\mathfrak{T}_{R_\tau}$ as there are possible histories over the events $\bigcup_{C\in\mathcal{C}} \textit{stable-info}_{p_C}(I)$. The number of such histories is bounded by $M^{O(N^4)}$. Taking the product of the automata $\{\mathbf{R}\mathfrak{B}_\tau\}_{\tau\in\mathfrak{T}_B}$ results in the tables $\{\mathfrak{T}_{R_\tau}\}_{\tau\in\mathfrak{T}_B}$ being concatenated together. Since K copies are concatenated, the final table has at most $KM^{O(N^4)}$ entries.

The product construction does not affect the lengths of the lists $\textit{pairs}_C$. So, for each entry $\tau_R = (\mathcal{C}, T, \{(p_C, \textit{pairs}_C)\}_{C\in\mathcal{C}})$ in $\mathfrak{T}_R$, the number of pairs in the list $\textit{pairs}_C$ for each $C \in \mathcal{C}$ is bounded by $2M$.

Complementation The complexity of complementation is that of determinzation together with the cost of simulating a Streett automaton by a Büchi automaton. Let us analyze the second cost independently.

In the input Streett automaton $\mathbf{S}\mathfrak{A} = (\mathfrak{A}, \mathfrak{T}_S)$, let N' be the number of processes in $\mathfrak{A}$, M' the size of the largest set in the collection $\{V_p\}_{p\in\mathcal{P}}$, K' the number of entries in $\mathfrak{T}_S$ and L' the length of the longest list of pairs $\{G_C^j, R_C^j\}_{j\in[1..k_C]}$ across all signalling processes and all table entries.

Vardi's procedure for simulating an n-state Streett automaton with h pairs in the acceptance condition generates a Büchi automaton with $n2^h$ states. So, in step (ii) of the simulation procedure for Streett automata, each signalling process in the automaton $\mathbf{B}\mathfrak{B}_i$ we construct will have at most $M'2^{L'}$ local states. The intersection of the automata $\{\mathbf{B}\mathfrak{B}_i\}_{\tau_i\in X_\alpha}$ will involve at most K' automata at a time, since the number of table entries overall is bounded by K'. Intersecting K' Büchi asynchronous automata essentially generates the K' fold product of each process's local state space together with a modulo K' counter. Thus, each process in the resulting automaton will effectively have $(M'2^{L'})^{K'} = M'^{K'}2^{K'L'}$ local states.

When we take the union of all the automata constructed according to step (ii), we will make upto K' disjoint copies of each process's local state space. We then merge this with the trivial automaton constructed in step (i), which has only one local state per process. So, overall the Büchi asynchronous automaton simulating $\mathbf{S}\mathfrak{A}$ will have not more than $K'M'^{K'}2^{K'L'}$ local states for each process.

If we plug in values for K', M' and L' after determinization, we see that $K' = KM^{O(N^4)}$, $M' = 2^{KM^{O(N^3)}}$ and $L' = 2M$, where K, M and N are the parameters corresponding to the original non-deterministic Büchi asynchronous automaton. So, number of local states of each process in the complementary Büchi asynchronous automaton $\mathbf{B}\overline{\mathfrak{A}}$ is bounded by $KM^{O(N^4)} \cdot (2^{KM^{O(N^3)}})^{KM^{O(N^4)}} \cdot 2^{KM^{O(N^4)} \cdot 2M}$, which is $2^{K^2M^{O(N^4)}}$.