Automata, Languages and Programming, 25th International Colloquium
Proceedings: Kim G. Larsen, Sven Skyum, Glynn Winskel (eds.)
Springer Lecture Notes in Computer Science 1443 (1998), 188–199.

Robust Asynchronous Protocols Are Finite-State

Madhavan Mukund[1 *], K Narayan Kumar[1 **],
Jaikumar Radhakrishnan[2], and Milind Sohoni[3]

[1] SPIC Mathematical Institute, 92 G.N. Chetty Road, Madras 600 017, India.
E-mail: {madhavan,kumar}@smi.ernet.in
[2] Computer Science Group, Tata Institute of Fundamental Research, Homi Bhabha Road, Bombay 400 005, India. E-mail: jaikumar@tcs.tifr.res.in
[3] Department of Computer Science and Engineering, Indian Institute of Technology, Bombay 400 076, India. E-mail: sohoni@cse.iitb.ernet.in

Abstract. We consider networks of finite-state machines which communicate over reliable channels which may reorder messages. Each machine in the network also has a local input tape. Since channels are unbounded, the network as a whole is, in general, infinite-state.

An asynchronous protocol is a network equipped with an acceptance condition. Such a protocol is said to be *robust* if it never deadlocks and, moreover, it either accepts or rejects each input in an unambiguous manner. The behaviour of a robust protocol is insensitive to nondeterminism introduced by either message reordering or the relative speeds at which components read their local inputs.

Using an automata-theoretic model, we show that, at a global level, every robust asynchronous protocol has a finite-state representation. To prove this, we establish a variety of pumping lemmas. We also demonstrate a distributed language which does not admit a robust protocol.

1 Introduction

We analyze message-passing systems from a language-theoretic point of view. In such systems, computing agents run *protocols* to collectively process distributed inputs, using messages for coordination. These messages may undergo different relative delays and hence arrive out of order. Protocols need to be "robust" with respect to the irregular behaviour of the transmission medium.

Most protocols assume that the transmission medium has an unlimited capacity to hold messages—undelivered messages are assumed to be stored in a transparent manner in intermediate buffers. Can the unlimited capacity of the medium enhance the power of message passing protocols?

Unfortunately, the answer is *no*; we show that even for a benign medium which does not lose messages, a "robust" protocol cannot use unbounded buffers to its advantage. However, if a protocol need not always gracefully halt, then the medium can be exploited to accept a larger class of "distributed languages".

---

* Partly supported by IFCPAR Project 1502-1.

** Currently on leave at Department of Computer Science, State University of New York at Stony Brook, NY 11794-4400, USA. E-mail: kumar@cs.sunysb.edu.

Consider then a system of processes which interact independently with the environment and communicate internally via message-passing. The communication between the processes and the programs which they run impose restrictions on the distributed input—some interactions with the environment are valid and some are not. For instance, consider a banking network which is connected to the external world via a set of automated teller machines. The protocol may enforce a limit on the number of withdrawals by an individual across the network.

We are interested in finite-state processes, so we assume that the number of different types of messages used by the system is finite. This is not unreasonable if we distinguish "control" messages from "data" messages. In our model, channels may reorder or delay messages. For simplicity, we assume that messages are never lost. Since messages may be reordered, the state of each channel can be represented by a finite set of counters which record the number of messages of each type which have been sent along the channel but are as yet undelivered.

We say that an asynchronous protocol is *robust* if it never deadlocks on any distributed input and every distributed input is either accepted or rejected in a consistent manner. In other words finite delays, reordering of messages and nondeterministic choices made by the protocol do not affect the outcome of a robust protocol on a given distributed input.

Our main result is that every language of distributed inputs accepted by a robust asynchronous protocol can be "represented" by a regular sequential language. In other words, a robust asynchronous protocol always has a globally finite-state description. This implies that robust protocols essentially use messages only for "handshaking". Since robust protocols can be modelled as finite-state systems, they may, in principle, be verified using automated tools [5].

The paper is organized as follows. In the next section, we define message-passing networks. In Section 3 we state some basic results about these networks, including a Contraction Lemma which leads to the decidability of the emptiness problem. Section 4 develops a family of pumping lemmas which are exploited in Section 5 to prove our main result about robust protocols. We also describe a simple language for which no robust protocol exists. In the final section, we discuss the connection between our results and those in Petri net theory and point out directions for future work. We have had to omit detailed proofs in this extended abstract. Full proofs and related results can be found in [10].

2 Message-passing networks

Natural numbers and tuples As usual, $\mathbb{N}$ denotes the set $\{0, 1, 2, \ldots\}$ of natural numbers. For $i, j \in \mathbb{N}$, $[i..j]$ denotes the set $\{i, i{+}1, \ldots, j\}$, where $[i..j] = \emptyset$ if $i > j$. We compare k-tuples of natural numbers component-wise. For $\overline{m} = \langle m_1, m_2, \ldots, m_k\rangle$ and $\overline{n} = \langle n_1, n_2, \ldots, n_k\rangle$, $\overline{m} \leq \overline{n}$ iff $m_i \leq n_i$ for each $i \in [1..k]$.

Message-passing automata A *message-passing automaton* $\mathcal{A}$ is a tuple $(S_a, S_t, \Sigma, \#, \Gamma, T, s_{\text{in}})$ where:

- S_a and S_t are disjoint, non-empty, finite sets of *active* and *terminal states*, respectively. The *initial state* s_{in} belongs to S_a.

- Σ is a finite *input alphabet* and $\#$ is a special end-of-tape symbol which does *not* belong to Σ. Let $\Sigma^{\#}$ denote the set $\Sigma \cup \{\#\}$.
- Γ is a finite set of *counters*. With each counter C, we associate two symbols, C^+ and C^-. We write $\Gamma^{\pm}$ to denote the set $\{C^+ | C \in \Gamma\} \cup \{C^- | C \in \Gamma\}$.
- $T \subseteq (S_a \times (\Sigma \cup \Gamma^{\pm}) \times S_a) \cup (S_a \times \{\#\} \times S_t) \cup (S_t \times \Gamma^{\pm} \times S_t)$ is the *transition relation*.

A message-passing automaton begins reading its input in an active state. It remains within the set of active states until it reads the special end-of-tape symbol. At this point, the automaton moves into the set of terminal states where the only moves possible are those which increment or decrement counters.

Networks A *message-passing network* is a structure $\mathcal{N} = (\{\mathcal{A}_i\}_{i \in [1..n]}, Acc, Rej)$ where:

- For $i \in [1..n]$, $\mathcal{A}_i = (S_a^i, S_t^i, \Sigma_i, \#_i, \Gamma_i, T_i, s_{\text{in}}^i)$ is a message-passing automaton. As before, for $i \in [1..n]$, $\Sigma_i^{\#}$ denotes the set $\Sigma_i \cup \{\#_i\}$.
- For $i, j \in [1..n]$, if $i \neq j$ then $\Sigma_i \cap \Sigma_j = \emptyset$.
- A *global state* of $\mathcal{N}$ is an n-tuple $\langle s_1, s_2, \ldots, s_n \rangle$ where $s_i \in (S_a^i \cup S_t^i)$ for $i \in [1..n]$. Let $Q_{\mathcal{N}}$ denote the set of global states of $\mathcal{N}$. If $q = \langle s_1, s_2, \ldots, s_n \rangle$ is a global state, then q_i denotes the i^{th} component s_i.
  The *initial state* of $\mathcal{N}$ is given by $q_{\text{in}} = \langle s_{\text{in}}^1, s_{\text{in}}^2, \ldots, s_{\text{in}}^n \rangle$. The *terminal states* of $\mathcal{N}$, denoted $Q_{\mathcal{N}}^t$, are given by $\prod_{i \in [1..n]} S_t^i$. The sets Acc and Rej are disjoint subsets of $Q_{\mathcal{N}}^t$; Acc is the set of *accept states* of $\mathcal{N}$ while Rej is the set of *reject states*. We do not insist that $Q_{\mathcal{N}}^t = Acc \cup Rej$—there may be terminal states which are neither accepting nor rejecting.

Counters may be shared across the network—shared counters represent channels along which components send messages to each other. Strictly speaking, a point-to-point channel would consist of a set of counters shared by two processes, where one process only increments the counters and the other only decrements the counters. Our definition permits a more generous notion of channels.

The assumption that local alphabets are pairwise disjoint is not critical—we can always tag each input letter with the location where it is read.

Let $\Sigma_{\mathcal{N}}$ denote $\bigcup_{i \in [1..n]} \Sigma_i^{\#}$ and $\Gamma_{\mathcal{N}}$ denote $\bigcup_{i \in [1..n]} \Gamma_i$.

Global transitions For a network $\mathcal{N}$, we can define a *global transition relation* $T_{\mathcal{N}}$ as follows. For $q, q' \in Q_{\mathcal{N}}$ and $d \in \Sigma_{\mathcal{N}} \cup \Gamma_{\mathcal{N}}^{\pm}$, (q, d, q') belongs to $T_{\mathcal{N}}$ provided:

- For some $i \in [1..n]$, $d \in \Sigma_i^{\#} \cup \Gamma_i^{\pm}$ and $(q_i, d, q_i') \in T_i$.
- For $j \neq i$, $q_j = q_j'$.

Configurations A *configuration* of $\mathcal{N}$ is a pair (q, f) where $q \in Q_{\mathcal{N}}$ and $f : \Gamma \to \mathbb{N}$ records the values stored in the counters. If the counters are $C_1, C_2, \ldots, C_k$ then we represent f by an element $\langle f(C_1), f(C_2), \ldots, f(C_k) \rangle$ of $\mathbb{N}^k$. By abuse of notation, the k-tuple $\langle 0, 0, \ldots, 0 \rangle$ is uniformly denoted $\overline{0}$, for all values of k.

We use χ to denote configurations. If $\chi = (q, f)$, $Q(\chi)$ denotes q and $F(\chi)$ denotes f. Further, for each counter C, $C(\chi)$ denotes the value $f(C)$.

Moves The network *moves* from configuration χ to configuration χ' on $d \in \Sigma_{\mathcal{N}} \cup \Gamma_{\mathcal{N}}^{\pm}$ if $(Q(\chi), d, Q(\chi')) \in T_{\mathcal{N}}$ and one of the following holds:

- $d \in \Sigma_{\mathcal{N}}$ and $F(\chi) = F(\chi')$.
- $d = C^+$, $C(\chi') = C(\chi) + 1$ and $C'(\chi) = C'(\chi')$ for every $C' \neq C$.
- $d = C^-$, $C(\chi') = C(\chi) - 1 \geq 0$ and $C'(\chi) = C'(\chi')$ for every $C' \neq C$.

Such a move is denoted $\chi \stackrel{(q,d,q')}{\longrightarrow} \chi'$—that is, transitions are labelled by elements of $T_{\mathcal{N}}$. Given a sequence of transitions $t_1 t_2 \ldots t_m = (q_1, d_1, q_2)(q_2, d_2, q_3) \ldots (q_m, d_m, q_{m+1})$, the corresponding sequence $d_1 d_2 \ldots d_m$ over $\Sigma_{\mathcal{N}} \cup \Gamma_{\mathcal{N}}^{\pm}$ is denoted $\alpha(t_1 t_2 \ldots t_m)$.

Computations and runs A *computation* of $\mathcal{N}$ is a sequence $\chi_0 \xrightarrow{t_1} \chi_1 \xrightarrow{t_2} \ldots \xrightarrow{t_m} \chi_m$. We also write $\chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ to indicate that there is a computation labelled $t_1 t_2 \ldots t_m$ from χ_0 to χ_m. Notice that χ_0 and $t_1 t_2 \ldots t_m$ uniquely determine all the intermediate configurations $\chi_1, \chi_2, \ldots, \chi_m$. If the transition sequence is not relevant, we just write $\chi_0 \Longrightarrow \chi_m$. As usual, $\chi \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow}$ and $\chi \Longrightarrow$ denote that there exists χ' such that $\chi \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi'$ and $\chi \Longrightarrow \chi'$, respectively.

For $K \in \mathbb{N}$, a *K-computation* of $\mathcal{N}$ is a computation $\chi_0 \Longrightarrow \chi_m$ where $C(\chi_0) \leq K$ for each $C \in \Gamma_{\mathcal{N}}$.

If w is a string over a set X and $Y \subseteq X$, we write $w \restriction_Y$ to denote the subsequence of letters from Y in w.

An input to the network $\mathcal{N}$ is an n-tuple $\overline{w} = \langle w_1, w_2, \ldots, w_n \rangle$—each component w_i is a word over the local alphabet Σ_i. As we shall see when we define the notion of a run, each component w_i of the input $\overline{w}$ is assumed to be terminated by the end-of-tape symbol $\#_i$ which is not recorded as part of the input.

A *run of $\mathcal{N}$ over $\overline{w}$* is a 0-computation $\chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ where $Q(\chi_0) = q_{\text{in}}$, $Q(\chi_m) \in \mathit{Acc} \cup \mathit{Rej}$ and $\alpha(t_1 t_2 \ldots t_m)\restriction_{\Sigma_i^\#} = w_i \#_i$ for each $i \in [1..n]$. The run is said to be *accepting* if $Q(\chi_m) \in \mathit{Acc}$ and *rejecting* if $Q(\chi_m) \in \mathit{Rej}$. The input $\overline{w}$ is *accepted* by $\mathcal{N}$ if $\mathcal{N}$ has an accepting run over $\overline{w}$.

A 0-computation starting from the initial state which reads the entire input is not automatically a run—a run *must* end in an accept or a reject state. As usual, an input $\overline{w}$ is *not* accepted if all runs on $\overline{w}$ end in reject states—in particular, if the network does not admit any runs on $\overline{w}$, then $\overline{w}$ is not accepted by $\mathcal{N}$.

Languages A *tuple language* over $\langle \Sigma_1, \Sigma_2, \ldots, \Sigma_n \rangle$ is a subset of $\prod_{i \in [1..n]} \Sigma_i^*$. The *language accepted* by $\mathcal{N}$, denoted $L(\mathcal{N})$, is the set of all inputs accepted by $\mathcal{N}$. A tuple language L is said to be *message-passing recognizable* if there is a network $\mathcal{N} = \{\mathcal{A}_i\}_{i \in [1..n]}$ with input alphabets $\{\Sigma_i\}_{i \in [1..n]}$ such that $L = L(\mathcal{N})$.

We will also be interested in the connection between sequential languages over $\Sigma_{\mathcal{N}}$ and tuple languages accepted by message-passing networks. We say that a word w over $\Sigma_{\mathcal{N}}$ *represents* the tuple $\langle w_1, w_2, \ldots, w_n \rangle$ if $w\restriction_{\Sigma_i^\#} = w_i \#_i$ for each $i \in [1..n]$. We call the word $\langle w\restriction_{\Sigma_1}, w\restriction_{\Sigma_2}, \ldots, w\restriction_{\Sigma_n} \rangle$ represented by w the *$\mathcal{N}$-projection* of w. Let $L_s \subseteq \Sigma_{\mathcal{N}}^*$ and $L \subseteq \prod_{i \in [1..n]} \Sigma_i^*$. We say that L_s *represents* L if the following conditions hold:

– For each word w in L_s, the $\mathcal{N}$-projection of w belongs to L.
– For each tuple $\overline{w} = \langle w_1, w_2, \ldots, w_n \rangle$ in L, there is a word w in L_s which represents $\overline{w}$.

Example 2.1. Let $\Sigma_1 = \{a\}$ and $\Sigma_2 = \{b\}$. Let L_{ge} denote the language $\{\langle a^\ell, b^m \rangle \mid \ell \geq m\}$. Figure 1 shows a network which accepts L_{ge}. The initial state of each component is marked $\Downarrow$ while the terminal states are marked by double circles. There is one accept state, $\langle s, s \rangle$, and no reject state.

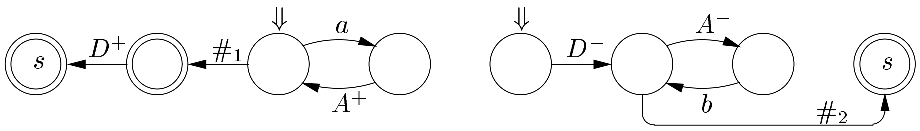

Fig. 1.

Each time the second process reads b it has to consume a message generated by the first process after reading a. Thus, the second process can read at most as many b's as the first process does a's. The counter D is used to signal that the first process's input has been read completely.

Robustness
In general, an asynchronous protocol may process the same distributed input in many different ways because of variations in the order in which components read their local inputs, reordering of messages due to delays in transmission as well as local nondeterminism at each component. Intuitively, a protocol is *robust* if its behaviour is insensitive to these variations—for each distributed input, all possible runs lead either to acceptance or rejection in a consistent manner. Further, a robust protocol should never deadlock along any computation—in other words, every input is processed fully and clearly identified as "accept" or "reject". This motivates following definition.

Robust networks Let $\mathcal{N}$ be a message-passing network. We say that $\mathcal{N}$ is *robust* if the following hold:

– For each input $\overline{w} = \langle w_1, w_2, \ldots, w_n \rangle$, $\mathcal{N}$ admits at least one run over $\overline{w}$. Moreover, if ρ and ρ' are two different runs of $\mathcal{N}$ over $\overline{w}$, either both are accepting or both are rejecting.
– Let $\overline{w} = \langle w_1, w_2, \ldots, w_n \rangle$ be any input and $\rho : \chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ a 0-computation of $\mathcal{N}$ such that $Q(\chi_0) = q_{\mathrm{in}}$. If $\alpha(t_1 t_2 \ldots t_m)\lceil_{\Sigma_i^\#}$ is a prefix of $w_i \#_i$ for each $i \in [1..n]$, then ρ can be extended to a run on $\overline{w}$.

It is easy to observe that if we interchange the accept and reject states of a robust network $\mathcal{N}$, we obtain a robust network for the complement of $L(\mathcal{N})$.

The network in Example 2.1 is not robust—if the number of b's exceeds the number of a's, the network hangs.

Example 2.2. We can make the network of Example 2.1 robust by changing the interaction between the processes. Rather than having the first process send a count of the number of inputs it has read, we make the processes read their inputs alternately, with a handshake in-between.

As before, let $\Sigma_1 = \{a\}, \Sigma_2 = \{b\}$ and $L_{\rm ge} = \{\langle a^\ell, b^m\rangle \mid \ell \geq m\}$. Figure 2 shows is a robust network for $L_{\rm ge}$. The initial states are marked $\Downarrow$ and the terminal states of each component are marked by double circles. There is one accept state, $\langle s,s\rangle$, and one reject state $\langle s,r\rangle$. The terminal states $\langle s,t\rangle$, $\langle t,r\rangle$, $\langle t,s\rangle$ and $\langle t,t\rangle$ are neither accepting nor rejecting.

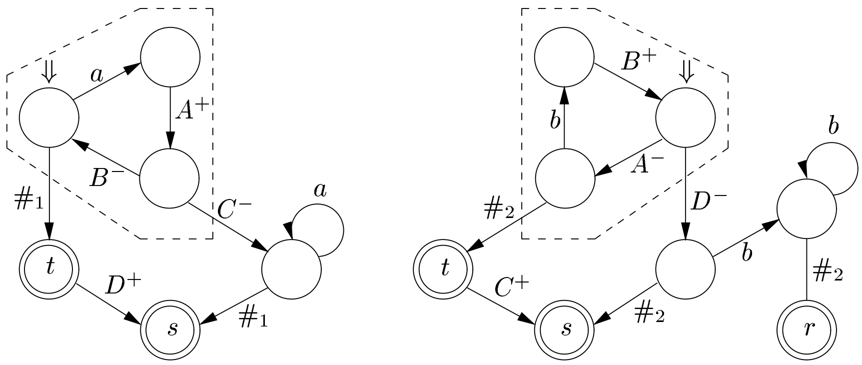

Fig. 2.

The loops enclosed by dashes represent the phase when the processes read their inputs alternately. This phase ends either when either process reads its end-of-tape symbol. If the loop ends with first process reading $\#_1$, the second process must immediately read $\#_2$. If the loop ends with the second process reading $\#_2$, the first process can go on to read any number of a's.

3 Analyzing Message-Passing Networks

The next two sections contain technical results about message-passing networks that we need to prove our main theorem. Many of these results have analogues in Petri net theory [13]—a detailed discussion is presented in the final section.

The following result is basic to analyzing the behaviour of message-passing networks. It follows from the fact that any infinite sequence of N-tuples of natural numbers contains an infinite increasing subsequence. We omit the proof.

Lemma 3.1. *Let X be a set with M elements and $\langle x_1, f_1\rangle, \langle x_2, f_2\rangle, \ldots, \langle x_m, f_m\rangle$ be a sequence over $X \times \mathbb{N}^N$ such that each coordinate of f_1 is bounded by K and for $i \in [1..m{-}1]$, f_i and f_{i+1} differ on at most one coordinate and this difference is at most 1. There is a constant ℓ which depends only on M, N and K such that if $m \geq \ell$, then there exist $i, j \in [1..m]$ with $i < j$, $x_i = x_j$ and $f_i \leq f_j$.*

Weak pumping constant We call the bound ℓ for M, N and K from the preceding lemma the *weak pumping constant* for (M, N, K), denoted $\pi_{M,N,K}$.

Using the weak pumping constant, we can identify when a run of a network can be contracted by eliminating a sequence of transitions. We omit the proof.

Lemma 3.2 (Contraction). *Let $\mathcal{N}$ be a message-passing network with M global states and N counters. For any K-computation $\chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ with $m > \pi_{M,N,K}$, there exist i and j, $m - \pi_{M,N,K} \leq i < j \leq m$, such that $\chi'_0 \stackrel{t_1 \ldots t_i t_{j+1} \ldots t_m}{\Longrightarrow} \chi'_{m-(j-i)}$ is also a K-computation of $\mathcal{A}$, with $\chi'_\ell = \chi_\ell$ for $\ell \in [0..i]$ and $Q(\chi_\ell) = Q(\chi'_{\ell-(j-i)})$ for $\ell \in [j..m]$.*

Corollary 3.3. *A message-passing network with M global states and N counters has an accepting run iff it has an accepting run whose length is bounded by $\pi_{M,N,0}$.*

It is possible to provide an explicit upper bound for $\pi_{M,N,K}$ for all values of M, N, and K. This fact, coupled with the preceding observation, yields the following result.

Corollary 3.4. *The emptiness problem for message-passing networks is decidable.*

4 A Collection of Pumping Lemmas

Change vectors For a string w and a symbol x, let $\#_x(w)$ denote the number of times x occurs in w. Let v be a sequence of transitions. Recall that $\alpha(v)$ denotes the corresponding sequence of letters. For each counter C, define $\Delta_C(v)$ to be $\#_{C^+}(\alpha(v)) - \#_{C^-}(\alpha(v))$. The *change vector* associated with v, denoted Δv, is given by $\langle \Delta_C(v) \rangle_{C \in \Gamma_\mathcal{N}}$.

Pumpable decomposition Let $\mathcal{N}$ be a message-passing network with N counters and let $\rho : \chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ be a computation of $\mathcal{N}$. A decomposition $\chi_0 \stackrel{u_1}{\Longrightarrow} \chi_{i_1} \stackrel{v_1}{\Longrightarrow} \chi_{j_1} \stackrel{u_2}{\Longrightarrow} \chi_{i_2} \stackrel{v_2}{\Longrightarrow} \chi_{j_2} \stackrel{u_3}{\Longrightarrow} \cdots \stackrel{u_y}{\Longrightarrow} \chi_{i_y} \stackrel{v_y}{\Longrightarrow} \chi_{j_y} \stackrel{u_{y+1}}{\Longrightarrow} \chi_m$ of ρ is said to be *pumpable* if it satisfies the following conditions:

(i) $y \leq N$.
(ii) For each $k \in [1..y]$, $Q(\chi_{i_k}) = Q(\chi_{j_k})$.
(iii) For each v_k, $k \in [1..y]$, Δv_k has at least one positive entry.
(iv) Let C be a counter and $k \in [1..y]$ such that $\Delta_C(v_k)$ is negative. Then, there exists $\ell < k$ such that $\Delta_C(v_\ell)$ is positive.

We refer to $v_1, v_2, \ldots, v_y$ as the *pumpable blocks* of the decomposition. We say that C is a *pumpable counter* if $\Delta_C(v_k) > 0$ for some pumpable block v_k. The following lemma shows that all the pumpable counters of a pumpable decomposition are simultaneously unbounded. We omit the proof. (This is similar to a well-known result of Karp and Miller in the theory of vector addition systems [7].)

Lemma 4.1 (Counter Pumping). *Let $\mathcal{N}$ be a network and ρ a K-computation of $\mathcal{N}$, $K \in \mathbb{N}$, with a pumpable decomposition of the form $\chi_0 \stackrel{u_1}{\Longrightarrow} \chi_{i_1} \stackrel{v_1}{\Longrightarrow} \chi_{j_1} \cdots \stackrel{u_y}{\Longrightarrow} \chi_{i_y} \stackrel{v_y}{\Longrightarrow} \chi_{j_y} \stackrel{u_{y+1}}{\Longrightarrow} \chi_m$. Then, for any $I, J \in \mathbb{N}$, with $I \geq 1$, there exist $\ell_1, \ell_2, \ldots, \ell_y \in \mathbb{N}$ and a K-computation ρ' of $\mathcal{N}$ of the form $\chi'_0 \stackrel{u_1}{\Longrightarrow} \chi'_{i'_1} \stackrel{v_1^{\ell_1}}{\Longrightarrow} \chi'_{j'_1} \cdots \stackrel{u_y}{\Longrightarrow} \chi'_{i'_y} \stackrel{v_y^{\ell_y}}{\Longrightarrow} \chi'_{j'_y} \stackrel{u_{y+1}}{\Longrightarrow} \chi'_p$ such that ρ' satisfies the following properties:*

(i) $\chi_0 = \chi'_0$.
(ii) $Q(\chi'_p) = Q(\chi_m)$.
(iii) *For* $i \in [1..y]$, $\ell_i \geq I$.
(iv) *For every counter* C, $C(\chi'_p) \geq C(\chi_m)$.
(v) *Let Γ_{pump} be the set of pumpable counters in the pumpable decomposition of ρ. For each counter $C \in \Gamma_{\text{pump}}$, $C(\chi'_p) \geq J$.*

Having shown that all pumpable counters of a pumpable decomposition can be simultaneously raised to arbitrarily high values, we describe a sufficient condition for a K-computation to admit a non-trivial pumpable decomposition.

Strong pumping constant For each $M, N, K \in \mathbb{N}$, we define the *strong pumping constant* $\Pi_{M,N,K}$ by induction on N as follows (recall that $\pi_{M,N,K}$ denotes the weak pumping constant for (M, N, K)):

$$\begin{array}{ll} \forall M, K \in \mathbb{N}. & \Pi_{M,0,K} = 1 \\ \forall M, N, K \in \mathbb{N}. & \Pi_{M,N+1,K} = \Pi_{M,N,\pi_{M,N+1,K}+K} + \pi_{M,N+1,K} + K \end{array}$$

Lemma 4.2 (Decomposition). *Let $\mathcal{N}$ be a network with M global states and N counters and let $K \in \mathbb{N}$. Let $\rho : \chi_0 \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} \chi_m$ be any K-computation of $\mathcal{N}$. Then, there is a pumpable decomposition $\chi_0 \stackrel{u_1}{\Longrightarrow} \chi_{i_1} \stackrel{v_1}{\Longrightarrow} \chi_{j_1} \cdots \stackrel{u_y}{\Longrightarrow} \chi_{i_y} \stackrel{v_y}{\Longrightarrow} \chi_{j_y} \stackrel{u_{y+1}}{\Longrightarrow} \chi_m$ of ρ such that for every counter C, if $C(\chi_j) > \Pi_{M,N,K}$ for some $j \in [0..m]$, then C is a pumpable counter in this decomposition.*

Proof Sketch: The proof is by induction on N, the number of counters. In the induction, the key step is to identify a prefix ρ' of ρ containing two configurations χ_r and χ_s such that $Q(\chi_r) = Q(\chi_s)$, $F(\chi_r) < F(\chi_s)$ and, moreover, no counter value exceeds $\pi_{M,N,K} + K$ within ρ'.

Having found such a prefix, we fix a counter C which increases between χ_r and χ_s and construct a new network $\mathcal{N}'$ which treats $\{C^+, C^-\}$ as input letters. Since $\mathcal{N}'$ has $N-1$ counters, the induction hypothesis yields a decomposition $u_2 v_2 u_3 v_3 \ldots u_y v_y u_{y+1}$ of the suffix of ρ after ρ'. We then set u_1 to be the segment from χ_0 to χ_r and v_1 to be the segment from χ_r to χ_s and argue that the resulting decomposition $u_1 v_1 u_2 v_2 \ldots u_y v_y u_{y+1}$ of ρ satisfies the conditions of the lemma. □

The Decomposition Lemma plays a major role in the proof of our main result.

5 Robustness and Regularity

The main technical result of this paper is the following.

Theorem 5.1. *Let $\mathcal{N}$ be a robust message-passing network. Then, there is a regular sequential language L_s over $\Sigma_{\mathcal{N}}$ which represents the tuple language $L(\mathcal{N})$.*

This means that at a global level, any robust asynchronous protocol can be substituted by an equivalent finite-state machine. To prove this result, we need some technical machinery.

Networks with bounded counters Let $\mathcal{N} = (\{\mathcal{A}_i\}_{i\in[1..n]}, Acc, Rej)$ be a message-passing network. For $K \in \mathbb{N}$, define $\mathcal{N}[K] = (Q[K], T[K], Q[K]_{\text{in}}, F[K])$ to be the finite-state automaton over the alphabet $\Sigma_{\mathcal{N}} \cup \Gamma^{\pm}_{\mathcal{N}}$ given by:

- $Q[K] = Q_{\mathcal{N}} \times \{f \mid f : \Gamma \longrightarrow [0..K]\}$, with $Q[K]_{\text{in}} = (q_{\text{in}}, \overline{0})$.
- $F[K] = Acc \times \{f \mid f : \Gamma \longrightarrow [0..K]\}$.
- If $(q, d, q') \in T_{\mathcal{N}}$, then $((q, f), d, (q', f')) \in T[K]$ where:
  - If $d \in \Sigma_{\mathcal{N}}$, $f' = f$.
  - If $d = C^+$, $f'(C') = f(C')$ for $C' \neq C$ and $f'(C) = \begin{cases} f(C)+1 & \text{if } f(C) < K \\ K & \text{otherwise.} \end{cases}$
  - If $d = C^-$, $f'(C') = f(C')$ for $C' \neq C$, $f(C) \geq 1$ and
  $$f'(C) = \begin{cases} f(C)-1 & \text{if } f(C) < K \\ K & \text{otherwise.} \end{cases}$$

Each transition $t = ((q, f), d, (q', f')) \in T[K]$ corresponds to a unique transition $(q, d, q') \in T_{\mathcal{N}}$, which we denote t^{-1}. For any sequence $t_1 t_2 \ldots t_m$ of transitions in $T[K]$, $\alpha(t_1 t_2 \ldots t_m) = \alpha(t_1^{-1} t_2^{-1} \ldots t_m^{-1})$. Moreover, if $(q_0, f_0') \stackrel{t_1 t_2 \ldots t_m}{\Longrightarrow} (q_m, f_m')$ and $(q_0, f_0) \stackrel{t_1^{-1} t_2^{-1} \ldots t_m^{-1}}{\Longrightarrow} \chi_m$, then $Q(\chi_m) = q_m$.

Thus, the finite-state automaton $\mathcal{N}[K]$ behaves like a message-passing network except that it deems any counter whose value attains a value K to be "full". Once a counter is declared to be full, it can be decremented as many times as desired. The following observations are immediate.

Proposition 5.2. *(i) If $(q_0, f_0') \xrightarrow{t_1'} \cdots \xrightarrow{t_m'} (q_n, f_m')$ is a computation of $\mathcal{N}$ then, $(q_0, f_0) \xrightarrow{t_1} \cdots \xrightarrow{t_m} (q_m, f_m)$ is a computation of $\mathcal{N}[K]$ where*

- *$t_1' \ldots t_m' = t_1^{-1} \ldots t_m^{-1}$.*
- *$\forall C \in \Gamma.\ \forall i \in [1..m].\ f_i(C) = \begin{cases} f_i'(C) & \text{if } f_j'(C) < K \text{ for all } j \leq i \\ K & \text{otherwise.} \end{cases}$*

(ii) Let $(q_0, f_0) \xrightarrow{t_1} \cdots \xrightarrow{t_m} (q_m, f_m)$ be a computation of $\mathcal{N}[K]$. There is a maximum prefix $t_1 \ldots t_\ell$ of $t_1 \ldots t_m$ such that $\mathcal{N}$ has a computation $(q_0, f_0') \xrightarrow{t_1^{-1}} \cdots \xrightarrow{t_\ell^{-1}} (q_\ell, f_\ell')$ with $f_0 = f_0'$. Moreover, if $\ell < m$, then for some counter C, $\alpha(t_{\ell+1}^{-1}) = C^-$, $f_\ell'(C) = 0$ and for some $j < \ell$, $f_j'(C) = K$.

(iii) Let $L_{seq}(\mathcal{N})$ denote the set of all words over $\Sigma_{\mathcal{N}}$ which arise in accepting runs of $\mathcal{N}$—in other words, $w \in L_{seq}(\mathcal{N})$ if there is an accepting run $\chi_0 \stackrel{t_1t_2\ldots t_m}{\Longrightarrow} \chi_m$ of $\mathcal{N}$ such that $w = \alpha(t_1t_2\ldots t_m)\restriction_{\Sigma_{\mathcal{N}}}$. Let $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K]) = \{w\restriction_{\Sigma_{\mathcal{N}}} \mid w \in L(\mathcal{N}[K])\}$. Then, $L_{seq}(\mathcal{N}) \subseteq L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$.

We can now prove our main result.

Proof Sketch: (of Theorem 5.1)
Let $\mathcal{N}$ be a robust network with M global states and N counters. By interchanging the accept and reject states, we obtain a robust network $\overline{\mathcal{N}}$ for $\overline{L(\mathcal{N})}$ with the same state-transition structure as $\mathcal{N}$. Let K denote $\Pi_{M,N,0}$, the strong pumping constant for $\mathcal{N}$ (and $\overline{\mathcal{N}}$). Consider the finite-state automaton $\mathcal{N}[K]$ generated from $\mathcal{N}$ by ignoring all counter values above K.

We know that $L_{seq}(\mathcal{N})$ is a subset of $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$. We claim that there is no word $w \in L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$ which represents an input $\langle w_1, w_2, \ldots, w_n\rangle$ from $L(\overline{\mathcal{N}})$.

Assuming the claim, it follows that for every word w in $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$, the $\mathcal{N}$-projection $\langle w\restriction_{\Sigma_1}, w\restriction_{\Sigma_2}, \ldots, w\restriction_{\Sigma_n}\rangle$ belongs to $L(\mathcal{N})$. On the other hand, since $L_{seq}(\mathcal{N})$ is a subset of $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$, every tuple $\langle w_1, w_2, \ldots, w_n\rangle$ in $L(\mathcal{N})$ is represented by some word in $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$. Thus, $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$ represents the language $L(\mathcal{N})$. Since $\mathcal{N}[K]$ is a finite-state automaton, the result follows.

To complete the proof, we must verify the claim. Suppose that there is a word w in $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$ which represents a tuple $\langle w_1, w_2, \ldots, w_n\rangle$ in $L(\overline{\mathcal{N}})$. There must be a run $\rho : \chi_0 \stackrel{t_1t_2\ldots t_m}{\Longrightarrow} \chi_m$ of $\mathcal{N}[K]$ on w which leads to a final state (q, f), where q is an accept state of $\mathcal{N}$ and hence a reject state of $\overline{\mathcal{N}}$.

Since $\overline{\mathcal{N}}$ has the same structure as $\mathcal{N}$, by Proposition 5.2 it is possible to mimic ρ in $\overline{\mathcal{N}}$. However, since $\langle w_1, w_2, \ldots, w_n\rangle \in L(\overline{\mathcal{N}})$ and $\overline{\mathcal{N}}$ is robust, it is not possible to mimic all of ρ in $\overline{\mathcal{N}}$—otherwise, $\overline{\mathcal{N}}$ would admit both accepting and rejecting runs on $\langle w_1, w_2, \ldots, w_n\rangle$. So, $\overline{\mathcal{N}}$ must get "stuck" after some prefix $\rho_1 : \chi_0 \stackrel{t_1t_2\ldots t_\ell}{\Longrightarrow} \chi_\ell$ of ρ—for some counter C, $t_{\ell+1}$ is a C^- move with $C(\chi_\ell) = 0$.

We call the residual computation $\rho_2 : \chi_\ell \stackrel{t_{\ell+1}t_{\ell+2}\ldots t_m}{\Longrightarrow} \chi_m$ the *stuck suffix* of ρ. Without loss of generality, assume that ρ has a stuck suffix of minimum length among all accepting runs over words in $L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$ representing tuples in $L(\overline{\mathcal{N}})$.

Let u be the prefix of w which has been read in ρ_1 and v be the suffix of w which is yet to be read. Let $\langle u_1, u_2, \ldots, u_n\rangle$ and $\langle v_1, v_2, \ldots, v_n\rangle$ be the $\mathcal{N}$-projections of u and v, respectively. Clearly, u_i is a prefix of $w_i\#_i$ for each $i \in [1..n]$. Since $\overline{\mathcal{N}}$ is robust, there must be an extension of ρ_1 to a run over $\langle w_1, w_2, \ldots, w_n\rangle$—this run must be accepting since $\langle w_1, w_2, \ldots, w_n\rangle \in L(\overline{\mathcal{N}})$.

Since $\mathcal{N}[K]$ can decrement C after ρ_1 while $\overline{\mathcal{N}}$ cannot, C must have attained the value K along ρ_1. By Lemmas 4.1 and 4.2, we can transform ρ_1 into a computation ρ_1' of $\overline{\mathcal{N}}$ which reaches a configuration $\chi_{\rho_1'}$ with $Q(\chi_{\rho_1'}) = Q(\chi_\ell)$, $C(\chi_{\rho_1'}) > 0$ and $F(\chi_{\rho_1'}) \geq F(\chi_\ell)$.

Let $\langle u_1', u_2', \ldots, u_n'\rangle$ be the input read along ρ_1'. Since ρ_1 could be extended to an accepting run of $\overline{\mathcal{N}}$ over $\langle u_1v_1, \ldots, u_nv_n\rangle$, ρ_1' can be extended to an accepting run of $\overline{\mathcal{N}}$ over $\langle u_1'v_1, \ldots, u_n'v_n\rangle$. On the other hand, we can extend ρ_1' in $\mathcal{N}[K]$ to a successful run over $u'v$, where u' is the sequentialization of $\langle u_1, u_2, \ldots, u_n\rangle$ along ρ_1'. This means that $u'v \in L_{\Sigma_{\mathcal{N}}}(\mathcal{N}[K])$ represents a tuple in $L(\overline{\mathcal{N}})$.

Since the counter C has a non-zero value after ρ'_1, we can extend ρ'_1 in $\overline{\mathcal{N}}$ to execute some portion of the stuck suffix ρ_2 of ρ. If this extension of ρ'_1 gets stuck, we have found a run which has a shorter stuck suffix than ρ, contradicting our assumption that ρ had a stuck suffix of minimum length.

On the other hand, if we can extend ρ'_1 to mimic the rest of ρ, we find that $\overline{\mathcal{N}}$ has an accepting run on $\langle u'_1 v_1, \ldots, u'_n v_n \rangle$ as well as a rejecting run on $\langle u'_1 v_1, \ldots, u'_n v_n \rangle$. This contradicts the robustness of $\overline{\mathcal{N}}$.

Thus it must be the case that there is no word w which is accepted by $\mathcal{N}[K]$ but which represents a tuple in $L(\overline{\mathcal{N}})$.

□

Example 5.3. Consider a network with two processes where $\Sigma_1 = \{a, c\}$ and $\Sigma_2 = \{b, d\}$. Let $L = \{\langle a^i c^k, b^j d^\ell \rangle \mid i \geq \ell \text{ and } j \geq k\}$. We can modify the network in Example 2.1 to accept L—we use two counters A and B to store the number of a's and b's read, respectively. We then decrement B each time c is read and A each time d is read to verify if the input is in L.

However, there is no robust protocol for L. If L has a robust protocol, there must be a regular language L_s which represents L. Let $\mathcal{A}$ be a finite-state automaton for L_s with n states. Choose a string $w \in L_s$ which has at least n occurrences each of c and d. It is easy to see that either all d's in w occur after all a's or all c's in w occur after all b's. We can then pump a suffix of w so that either the number of d's exceeds the number of a's or the number of c's exceeds the number of b's, thereby generating a word $u \in L_s$ with $\langle u\restriction_{\Sigma_1}, u\restriction_{\Sigma_2} \rangle \notin L$.

6 Discussion

Other models for asynchronous commmunication Many earlier attempts to model asynchronous systems focus on the infinite-state case—for instance, the port automaton model of Panangaden and Stark [12] and the I/O automaton model of Lynch and Tuttle [8]. Also, earlier work has looked at issues far removed from those which are traditionally considered in the study of finite-state systems.

Recently, Abdulla and Jonsson have studied decision problems for distributed systems with asynchronous communication [1]. However, they model channels as unbounded, fifo buffers, a framework in which most interesting questions become undecidable. The results of [1] show that the fifo model becomes tractable if messages may be lost in transit: questions such as reachability of configurations become decidable. While their results are, in general, incomparable to ours, we remark that their positive results hold for our model as well.

Petri net languages Our model is closely related to Petri nets [3, 6]. We can go back and forth between labelled Petri nets and message-passing networks while maintaining a bijection between the firing sequences of a net N and the computations of the corresponding network $\mathcal{N}$.

There are several ways to associate a language with a Petri net [4, 6, 13]. The first is to examine all firing sequences of the net. The second is to look at firing sequences which lead to a set of *final markings*. A third possibility is to identify firing sequences which reach markings which *dominate* some final marking. The third class corresponds to message-passing recognizable languages.

A number of positive results have been established for the first class of languages—for instance, regularity is decidable [2, 14]. On the other hand, a number of negative results have been established for the second class of languages—for instance, it is undecidable whether such a language contains *all* strings [14]. However, none of these results, positive or negative, carry over to the third class—ours is one of the few tangible results for this class of Petri net languages.

Directions for future work A challenging problem is to synthesize a distributed protocol from a description in terms of global states. In systems with synchronous communication, this is possible using an algorithm whereby each process maintains the latest information about the rest of the system [11, 15]. This algorithm has been extended to message-passing systems and could help in solving the synthesis problem [9]. Another important question is to be able to decide whether a given protocol is robust. We believe the problem is decidable.

References

1. P.A. Abdulla and B. Jonsson: Verifying programs with unreliable channels, in *Proc. 8th IEEE Symp. Logic in Computer Science*, Montreal, Canada (1993).
2. A. Ginzburg and M. Yoeli: Vector addition systems and regular languages, *J. Comput. System. Sci.* **20** (1980) 277–284
3. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines, *Theoret. Comput. Sci* **7** (1978) 311–324.
4. M. Hack: Petri Net Languages, *C.S.G. Memo 124*, Project MAC, MIT (1975).
5. G.J. Holzmann: *Design and validation of computer protocols*, Prentice Hall (1991).
6. M. Jantzen: Language theory of Petri nets, in W. Brauer, W. Reisig, G. Rozenberg (eds.), *Advances in Petri Nets, 1986, Vol 1*, Springer LNCS **254** (1986) 397–412.
7. R.M. Karp and R.E. Miller: Parallel program schemata, *J. Comput. System Sci.*, **3** (4) (1969) 167–195.
8. N.A. Lynch and M. Tuttle: Hierarchical correctness proofs for distributed algorithms, *MIT/LCS/TR-387*, Laboratory for Computer Science, MIT (1987).
9. M. Mukund, K. Narayan Kumar and M. Sohoni: Keeping track of the latest gossip in message-passing systems, *Proc. Structures in Concurrency Theory (STRICT)*, Berlin 1995, Workshops in Computing Series, Springer-Verlag (1995) 249–263.
10. M. Mukund, K. Narayan Kumar, J. Radhakrishnan and M. Sohoni: Counter automata and asynchronous communication, *Report TCS-97-4*, SPIC Mathematical Institute, Madras, India (1997).
11. M. Mukund and M. Sohoni: Gossiping, asynchronous automata and Zielonka's theorem, *Report TCS-94-2*, School of Mathematics, SPIC Science Foundation, Madras, India (1994).
12. P. Panangaden and E.W. Stark: Computations, residuals, and the power of indeterminacy, *Proc. ICALP '88*, Springer LNCS **317** (1988) 439–454.
13. J.L. Peterson: *Petri net theory and the modelling of systems*, Prentice Hall (1981).
14. R. Valk and G. Vidal-Naquet: Petri nets and regular languages, *J. Comput. System. Sci.* **23** (3) (1981) 299–325.
15. W. Zielonka: Notes on finite asynchronous automata, *R.A.I.R.O.—Inf. Théor. et Appl.*, **21** (1987) 99–135.