Computing Anonymously with Arbitrary Knowledge

Paolo Boldi Sebastiano Vigna
Dipartimento di Scienze dell'Informazione, Università di Milano, Italy
vigna@acm.org boldi@dsi.unimi.it

Abstract

We provide characterizations of the relations that can be computed with *arbitrary* knowledge on networks where all processors use the same algorithm and start from the same state (in particular, we do not assume that a bound on the network size is known). Three activation models are considered (synchronous, asynchronous, interleaved).

1 Introduction

The question concerning which problems can be solved by a distributed system when all processors use the same algorithm and start from the same state has a long story: it was firstly formulated by Angluin [1], who investigated the problem of establishing a "center". She was the first to realize the connection with the theory of *graph coverings*, which was going to provide, in particular with the work of Yamashita and Kameda [11], several characterization for problems that are solvable under certain topological constraints. Further investigation led to the classification of computable functions [11, 10, 3, 9], and allowed to eliminate several restrictions (such as bidirectionality, distinguished links, synchronicity, ...) [4, 5].

There is however a pitfall: all existing characterizations are based on the assumption that the processors know a bound N on the number of nodes in the network. This is an incredibly strong assumption; indeed, it implies that all algorithms can be simulated by constructing certain trees (the *views* of the system, which resume all information a processor can gather [11]) for $2N-1$ steps, and then terminating. In a sense, the whole issue becomes trivial, as one of the main problems—termination—is factored out *a priori*.

In this paper we characterize for the first time the relations that can be computed with *arbitrary* knowledge (i.e., by an arbitrary class of networks). Our networks are just directed graphs coloured on their arcs (information such as processor identity, communication models etc. can be easily encoded on the arc colouring [4]), and each processor changes its state depending on its previous state and on the state of its in-neighbours. Our characterizations do not assume *any* topological knowledge. The problem specification is given by a certain relation between the inputs and the outputs of the processors, and we allow the relation to depend on the network, so that problems like topology reconstruction can be specified. Note that in general it is not true that only trivial relations can be computed; for instance, one can certainly compute relations that depend only on the inputs of a bounded neighbourhood [8]. Moreover, one could use some *a priori* knowledge about the network to deduce some *global* properties (such as the number of processors) only from *local* data (such as the number of in-neighbours). Similar knowledge could be obtainable if the set of inputs we are considering is restricted.

We consider three known activation models: asynchronous (at each step, we can activate any subset of enabled processors), synchronous (all processors) and interleaved (a.k.a. *central daemon*—exactly one processor), and we give two characterization theorems of the following form: a certain problem is solvable on a certain class of networks iff there is a monotonic function f between two partial orders, satisfying a condition telling us that it is "sufficiently defined". Unfortunately, the notation to give our results is rather formidable, so it is difficult to explain the exact content of our theorems out of con-

text. Nonetheless, the subtleties of the matter require a completely formal development. (Analogous results have been obtained recently [2] for *anonymous shared memory systems*, in which all processors can read and write a set of shared registers, and activation is interleaved; the shared memory model is of course more powerful, as information can travel in constant time between any two processors, so the authors can show, for instance, that consensus is solvable even without knowing a bound on the number of processes.)

Intuitively, the domain of the function f is the set of partial views of the system (in the bounded-size case views are complete trees, but this does not fit when termination must be taken into account); note that there is no known notion of view for interleaved systems, so we introduce one, and we prove it to possess the known useful properties of standard views for synchronous systems (in particular, the state of a processor depends only on its current view). The function f maps views to output values (or $\perp$, for nonterminated processors); it tells us at the same time *when* to terminate and *what* to output.

In a sense, our monotonic functions play the a rôle similar to the program given to a universal Turing machine in classical computability: every computable relation can be computed by a universal distributed algorithm (the view construction) that accepts f as input, and uses it as a test for termination (and as an output function as well). On the other hand, suitable functions f induce a computed relation, the "problem solved by f", just like a program for the universal Turing machine computes a partial recursive function (its input/output relation). The analogy should be taken with care, however, as f cannot be described finitely in general.

In the last section, we outline an example of application of our characterization. The result proved is very simple, but the reader can easily check that finding a direct proof is by no means a simple task. We also sketch out some implications for self-stabilizing systems.

As a final note, we remark that in this work we are only concerned with the distributed aspects of the computability issue: we assume that processors have unlimited local computational power, and that they never fault.

2 Basic definitions

A *(directed) (multi)graph* G is defined by a set of nodes $N_G = \{1, 2, \ldots, n\}$ and a finite set A_G of arcs, and by two functions $s_G, t_G : A_G \to N_G$ that specify the source and the target of each arc. An *(arc-)coloured graph* (with set of colours C) is a graph endowed with a colouring function $\gamma : A_G \to C$. The subscripts will be dropped whenever no confusion is possible. We write $i \xrightarrow{a} j$ when the arc a has source i and target j, and $i \to j$ when $i \xrightarrow{a} j$ for some $a \in A_G$.

In the following, by a *network* we shall always mean a strongly connected coloured graph with at least one arc. The nodes of such a graph are called *processors*.

Computations of a network are defined by a state space and a transition function specifying how a processor must change its state when it is activated. The new state must depend, of course, on the previous state, and on the states of the in-neighbours; the latter are marked with the colour of the corresponding incoming arc (thus, e.g., if all incoming arcs have different colours a processor can distinguish its in-neighbours). We also need a function mapping input values to initial states and final states to output values. Formally, a *protocol* P is given by a set X_P of local states, a distinguished subset $F_P \subseteq X_P$ of final states, an input set Υ, an input function $\mathrm{in}_P : \Upsilon \to X_P$, an output set Ω, an output function $\mathrm{out}_P : F_P \to \Omega$ and a transition function $\delta_P : X_P \times (C \times X_P)^{\oplus} \to X_P$ (where $(C \times X_P)^{\oplus}$ is the set of finite multisets[1] over $C \times X_P$) satisfying the constraint that for every $x \in F_P$, $\delta_P(x, -) = x$. This condition simply states that when a processor enters a final state, it cannot change its own state thereafter, whatever input it receives from its in-neighbours.

The subscript P is often omitted, when understood from the context. Moreover, the input and output functions will be silently extended componentwise to vectors. We also consider the sets of C, Υ and Ω to be fixed.

A *global state* (for n processors, with respect to the protocol P) is a vector $\boldsymbol{x} \in X^n$. Such a state is *final* if

[1] For our purposes, finite multisets over S can be formalized as the elements of the abelian monoid free over S; we use bold braces to denote them.

$x_i \in F$ for every i. If a network G is in global state $\boldsymbol{x}$, every processor i has an input multiset $\left\{ \langle \gamma(a), x_j \rangle \right\}_{j \xrightarrow{a} i}$.

Given a network G and a state $\boldsymbol{x}$, a processor i is *enabled in* $\boldsymbol{x}$ iff

$$\delta\left(x_i, \left\{ \langle \gamma(a), x_j \rangle \right\}_{j \xrightarrow{a} i}\right) \neq x_i,$$

that is, if processor i is in a position to change its state. An *activation* for G in state $\boldsymbol{x}$ is a nonempty set of enabled processors; the activation is called

- *synchronous* if it contains every enabled processor;
- *interleaved* if it contains but a single enabled processor.

When we need to emphasize that no constraint is required on the set of activated processors, we use the term *asynchronous*.

A *computation* of the protocol P on the network G (with n processors) with input $\boldsymbol{v} \in \Upsilon^n$ is given by a (possibly infinite) sequence of global states $\boldsymbol{x}^0, \boldsymbol{x}^1, \ldots, \boldsymbol{x}^T(, \ldots)$ and a sequence of sets of processors $A^0, A^1, \ldots, A^{T-1}(, \ldots)$ (called the *activation sequence*) such that:

1. $\boldsymbol{x}^0 = \mathrm{in}(\boldsymbol{v})$;
2. for all t, A^t is an activation for G in state $\boldsymbol{x}^t$, and $\boldsymbol{x}^{t+1}$ satisfies

$$x_i^{t+1} = \begin{cases} x_i^t & \text{if } i \notin A^t \\ \delta\left(x_i^t, \left\{ \langle \gamma(a), x_j^t \rangle \right\}_{j \xrightarrow{a} i}\right) & \text{otherwise.} \end{cases}$$

Sometimes we shall use the word "computation" to denote only the sequence $\boldsymbol{x}^0, \ldots, \boldsymbol{x}^T(, \ldots)$.

A computation is called synchronous, interleaved or asynchronous if every activation is such. It is *terminating* if it is finite and its last state is final; in this case, its output is defined as the vector $\mathrm{out}\left(\boldsymbol{x}^T\right)$.

Given a class of networks $\mathcal{C}$, a *($\mathcal{C}$-indexed) relation* R is a function associating to each $G \in \mathcal{C}$ with n processors a set $R_G \subseteq \Upsilon^n \times \Omega^n$. Relations are used to define problems: for instance, if $\Upsilon = \{*\}$ and $\Omega = \{0, 1\}$, the relation containing all pairs $\langle * * \cdots *, b_1 b_2 \cdots b_n \rangle$, where exactly one of the b_i's is nonzero, defines the well-known *anonymous election problem*. Function computations, topology reconstruction etc. can be easily described in a similar way. Note that classes specify knowledge: the larger the class, the smaller the knowledge (the class of all networks corresponds to no knowledge at all; a singleton to maximum knowledge).

Let $\mathcal{C}$ be a class of networks, R be a relation and P a protocol. We say that P *computes* (in a synchronous, interleaved, asynchronous manner resp.) R on $\mathcal{C}$ if for all $G \in \mathcal{C}$ with n processors it happens that

- for all $\boldsymbol{v} \in \mathrm{dom}(R_G)$, every maximal (synchronous, interleaved, asynchronous resp.) computation of P on G with input $\boldsymbol{v}$ is terminating, and its output $\boldsymbol{\omega}$ is such that $\boldsymbol{v}\ R_G\ \boldsymbol{\omega}$
- for all $\boldsymbol{v} \notin \mathrm{dom}(R_G)$, no processor ever reaches a final state in every (synchronous, interleaved, asynchronous resp.) computation of P on G with input $\boldsymbol{v}$.

The requirement on inputs that are not in the domain of R_G, that is, those $\boldsymbol{v}$ for which there is no $\boldsymbol{\omega}$ such that $\boldsymbol{v}\ R_G\ \boldsymbol{\omega}$, is imposed in analogy with the theory of recursive functions (but note that, besides "truly" divergent, i.e., infinite, computations we have also the possibility of deadlocks). We remark that in other literature (e.g., [2]) the behaviour on inputs not in the domain of the relation to be computed is considered irrelevant; however, our choice allows to analyze properly semidefined relations, as shown in Section 5.

3 Synchronous and asynchronous computations

The title of this section is due to the fact that the two types of computation above, as we are going to show, give rise to the same computable relations.

Definition 1 A protocol P is *well terminating* iff for all $x \in X$ and $\boldsymbol{I} \in (C \times F)^{\oplus}$ it holds that $\delta(x, \boldsymbol{I}) \in F$.

The previous condition says that a processor changes its state to a final one whenever its in-neighbours are all in a final state. The first observation we need is that if a

protocol computes a relation in a synchronous manner, then it is possible to find a well-terminating protocol that computes the same relation asynchronously.

Theorem 1 Let P be protocol computing synchronously the relation R on the class $\mathcal{C}$. Then there exists a well-terminating protocol P' computing asynchronously R on $\mathcal{C}$.

Proof. See the Appendix. ∎

The previous theorem shows in particular that relations computable synchronously can also be computed asynchronously. Since the opposite inclusion is trivial (just by using the very same protocol), we have the following

Corollary 1 A relation can be computed synchronously on a class iff it can be computed asynchronously. Moreover, any such relation can be computed by a well-terminating protocol.

Thus, we can limit our considerations to well-terminating protocols computing synchronously.

In order to state our characterization theorem for the synchronous case, we shall need to manage trees ordered by *truncation prefix*: essentially, we will write $T \leq T'$ when truncating T' at the height of T gives exactly T. More formally, we define $\mathcal{T}$ (the set of finite trees with nodes coloured on Υ and arcs coloured on C) as the least solution [7] to the equation

$$\mathcal{T} = \Upsilon \times (C \times \mathcal{T})^{\oplus},$$

and we write $\upsilon\{c_1T_1, \ldots, c_kT_k\}$ instead of

$$\langle \upsilon, \{\langle c_1, T_1\rangle, \ldots, \langle c_k, T_k\rangle\}\rangle$$

to denote the tree having root coloured by υ, with k subtrees $T_1, \ldots, T_k$, where the arc leading from the root to the subtree T_i is coloured by c_i. (The shorthand υ is used for trees having no internal node.) We write $h(T)$ for the *height* of a tree T, defined in the obvious recursive way from $h(\upsilon) = 0$.

Now we recursively define the truncation prefix ordering $\leq$ as follows:

- $\upsilon \leq \upsilon\{c_1T_1, \ldots, c_kT_k\}$;
- $T = \upsilon\{c_1T_1, \ldots, c_kT_k\} \leq \upsilon\{c_1T_1', \ldots, c_kT_k'\}$ if $T_i \leq T_i'$ for all i and $T_i = T_i'$ for all T_i of nonmaximal height (i.e., $h(T_i) < h(T) - 1$).

Said otherwise, $T \leq T'$ if T' is obtained by "growing" the deepest leaves of T.

We are now in the position of defining the view of processor i after t synchronous steps when a function $f : \mathcal{T} \to \Omega_\perp$ is used as termination function ($\Omega_\perp$ is the *flat* domain of outputs, obtained adding an element $\perp$ to Ω, and setting $\perp \leq \omega$ for every $\omega \in \Omega$; $\perp$ means intuitively "still undefined"). Of course, the view will depend on the network G and on the particular given input $\boldsymbol{\upsilon}$, so we shall use the following notation

$$G^i_{\boldsymbol{\upsilon}}|^f t$$

which will appear often, so we suggest our reader to take a moment to digest it.

Definition 2 Let $f : \mathcal{T} \to \Omega_\perp$ be a monotonic function. Then, for all networks G with n processors, $\boldsymbol{\upsilon} \in \Upsilon^n$, processors i of G and $t \in \mathbf{N}$ we define recursively

$$G^i_{\boldsymbol{\upsilon}}|^f 0 = \upsilon_i$$

$$G^i_{\boldsymbol{\upsilon}}|^f t+1 = \begin{cases} G^i_{\boldsymbol{\upsilon}}|^f t & \text{if } f\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) \neq \perp; \\ \upsilon_i\left\{\gamma(a)\ G^j_{\boldsymbol{\upsilon}}|^f t\right\}_{j \xrightarrow{a} i} & \text{otherwise.} \end{cases}$$

It is worth noticing that there is a protocol P_f (with local state $\mathcal{T}$) whose synchronous execution on the network G with input $\boldsymbol{\upsilon}$ assigns to each processor i at step t the state $G^i_{\boldsymbol{\upsilon}}|^f t$. At each step, a processor gathers the current views of its in-neighbours, and combines them in its own new view (this corresponds to the "otherwise" case in Definition 2). Moreover, each processor uses f in order to know when to terminate (all states in $f^{-1}(\Omega)$ are final, and we use f as output function). The main difference with the "classical" view construction is that we use the termination information contained in f.

We now prove some natural properties of views: they have height t if termination has not been reached, and they form an ascending chain w.r.t. the ordering of $\mathcal{T}$.

Lemma 1 With the notation of Definition 2, assume that $f(\upsilon\{c_1T_1,\ldots,c_kT_k\}) = \bot$ implies $f(T_i) = \bot$ for some i. Then,

- if $f\left(G^i_{\boldsymbol{v}}|^f t\right) = \bot$, then $h\left(G^i_{\boldsymbol{v}}|^f t+1\right) = t+1$;
- $G^i_{\boldsymbol{v}}|^f t \leq G^i_{\boldsymbol{v}}|^f t+1$.

Proof. See the Appendix. ∎

Given a tree T, we denote with prune(T) the tree obtained deleting the deepest leaves of T. It is easy to show (observing that if $T \leq T'$ and $h(T) = h(T') - 1$, then prune$\left(T'\right) = T$) the following consequence of the previous lemma:

Corollary 2 Under the same conditions as in Lemma 1, if $f\left(G^i_{\boldsymbol{v}}|^f t\right) = \bot$ then

$$\text{prune}\left(G^i_{\boldsymbol{v}}|^f t+1\right) = G^i_{\boldsymbol{v}}|^f t.$$

We can now prove that the state of a processor in every well-terminating protocol depends only on its view, for a suitable f: Corollary 2 will play a fundamental rôle, as it means that every processor can recover its previous state by pruning its current view.

Theorem 2 Let P be a well-terminating protocol. Then, there exists a function $\varphi : \mathcal{T} \to X$ with the following property:

> for every network G with n processors, every processor i, every input $\boldsymbol{v} \in \Upsilon^n$ and every $t \in \mathbf{N}$, the state $\boldsymbol{x}^t$ reached by protocol P on the network G with input $\boldsymbol{v}$ after t synchronous steps satisfies
>
> $$x^t_i = \varphi\left(G^i_{\boldsymbol{v}}|^f t\right),$$
>
> where $f = \text{out} \circ \varphi : \mathcal{T} \to \Omega_\bot$, and out$(-)$ is defined as $\bot$ on nonfinal states.

Proof. See the Appendix. ∎

We can now state our characterization theorem for the (a)synchronous case:

Theorem 3 Given a class $\mathcal{C}$, a relation R is synchronously computable on $\mathcal{C}$ iff there is a monotonic function $f : \mathcal{T} \to \Omega_\bot$ such that for all $G \in \mathcal{C}$ and $\boldsymbol{v} \in \text{dom}(R_G)$

$$\boldsymbol{v}\ R_G \left\langle \bigvee_t f\left(G^1_{\boldsymbol{v}}|^f t\right), \ldots, \bigvee_t f\left(G^n_{\boldsymbol{v}}|^f t\right)\right\rangle$$

(in particular, when this happens the last vector is entirely nonbottom), and for all $\boldsymbol{v} \notin \text{dom}(R_G)$

$$\left\langle \bigvee_t f\left(G^1_{\boldsymbol{v}}|^f t\right), \ldots, \bigvee_t f\left(G^n_{\boldsymbol{v}}|^f t\right)\right\rangle = \langle \bot, \bot, \ldots, \bot\rangle.$$

Proof. For the left-to-right implication, let P be a well-terminating protocol computing R; define φ and f as in Theorem 2, and let $\boldsymbol{x}^0, \ldots, \boldsymbol{x}^T$ be the maximal synchronous computation of P on the network G with input $\boldsymbol{v} \in \text{dom}(R_G)$. Then, we already know that $x^T_i = \varphi\left(G^i_{\boldsymbol{v}}|^f T\right)$, so out$\left(x^T_i\right)$ is equal to $f\left(G^i_{\boldsymbol{v}}|^f T\right)$, hence the thesis (if $\boldsymbol{v} \notin \text{dom}(R_G)$ the computation is infinite and f is always bottom).

For the right-to-left implication, consider the synchronous protocol P_f that lets each processor i build $G^i_{\boldsymbol{v}}|^f t$ at step t, and use f as output function. ∎

4 Interleaved computations

It is known that interleaved activation is more powerful than synchronous activation: for instance, it is easy to give a graph-colouring protocol for the class of bidirectional networks (pick up a colour your neighbours still did not pick and stop), while the problem is unsolvable in the synchronous case, even restricting the class to a single ring, as all processors are constrained to remain in the same state.

The first problem to solve in order to give the interleaved counterpart of Theorem 3 is that there is no notion of view in the literature for interleaved computations. The main obstruction is the impossibility of duplicating Corollary 2, which is a cornerstone in the proof of Theorem 2: in other words, pruning the view of a processor in an interleaved activation does *not* give the view before the last activation.

To work around this limitation, we must use a richer domain $\mathcal{H}$, that encodes not only the present view, but also the previous ones, recursively. More precisely, we define $\mathcal{H}$ as the least solution to the equation

$$\mathcal{H} = \Upsilon \times \left((C \times \mathcal{H})^{\oplus}\right)^*,$$

and we write $\upsilon \boldsymbol{H}_1 \cdots \boldsymbol{H}_k$ for a generic element of $\mathcal{H}$, where $\boldsymbol{H}_i = \left\{c_1 H_{i1}, \ldots, c_{l_i} H_{il_i}\right\}$. We consider $\mathcal{H}$ ordered as follows: $\upsilon \boldsymbol{H}_1 \cdots \boldsymbol{H}_k \leq \upsilon \boldsymbol{H}_1 \cdots \boldsymbol{H}_{k+l}$, with $l \in \mathbf{N}$.

We denote with $\sigma, \tau, \ldots$ (finite or infinite) sequences of processors, with the convention that $\sigma = \sigma^0\sigma^1\cdots$, and so on. We also use them as interleaved activation sequences (in place of $\{\sigma^0\}\{\sigma^1\}\cdots$). Clearly, the view of a processor is different depending on the activation sequence, so we shall now use the notation $G^i_{\boldsymbol{\upsilon}}|^f\sigma$ to denote the view (in $\mathcal{H}$) of processor i after the activation sequence σ. As in the synchronous case, there will be a straightforward protocol with state space $\mathcal{H}$ such that the state of processor i after σ is exactly $G^i_{\boldsymbol{\upsilon}}|^f\sigma$: each processor will gather the current views from its in-neighbours, combine them and *concatenate* the result to its current view (rather than replacing it).

Definition 3 Let $f : \mathcal{H} \to \Omega_\perp$ be a monotonic function. Then, for all networks G with n processors, $\boldsymbol{\upsilon} \in \Upsilon^n$, processor i of G and $\sigma \in \{1, 2, \ldots, n\}^*$ we define recursively

$$G^i_{\boldsymbol{\upsilon}}|^f\varepsilon = \upsilon_i\varepsilon$$

$$G^i_{\boldsymbol{\upsilon}}|^f\sigma j = \begin{cases} G^i_{\boldsymbol{\upsilon}}|^f\sigma & \text{if } f\left(G^i_{\boldsymbol{\upsilon}}|^f\sigma\right) \neq \perp \text{ or } i \neq j; \\ (G^i_{\boldsymbol{\upsilon}}|^f\sigma)\ \left\{\gamma(a)\ G^j_{\boldsymbol{\upsilon}}|^f\sigma\right\}_{j \xrightarrow{a} i} & \text{otherwise.} \end{cases}$$

Before proving the version of Theorem 2 for the interleaved case, we need give a definition. If P, G and $\boldsymbol{\upsilon}$ are understood, we write $\boldsymbol{x}^\sigma$ for the global state of G after the activation sequence σ.

Definition 4 Let G be a network with n processors, P a protocol, and $\boldsymbol{\upsilon} \in \Upsilon^n$. The *restriction* $\bar{\sigma}$ of $\sigma \in \{1, 2, \ldots, n\}^*$ to G, P and $\boldsymbol{\upsilon}$ is defined recursively by $\bar{\varepsilon} = \varepsilon$ and

$$\overline{\sigma i} = \begin{cases} \bar{\sigma} i & \text{if } i \text{ is enabled in } \boldsymbol{x}^{\bar{\sigma}}; \\ \bar{\sigma} & \text{otherwise.} \end{cases}$$

Note that the above definition is sensible, as $\bar{\sigma}$ is always an activation sequence for P on the network G with input $\boldsymbol{\upsilon}$. Moreover, the definition naturally extends to infinite sequences (by taking the limit of the restricted prefixes; note however that the restriction of an infinite sequence may be finite).

Said otherwise, the restriction of a sequence σ is just the activation sequence obtained by deleting from σ the occurrences of processors that are not enabled (either because they are in a final state, or because they are simply not allowed to take a step).

Theorem 4 Let P be a protocol. Then, there exists a function $\varphi : \mathcal{H} \to X$ with the following property:

> for every network G with n processors, every processor i, every input $\boldsymbol{\upsilon} \in \Upsilon^n$ and every sequence $\sigma \in \{1, \ldots, n\}^*$, the state $\boldsymbol{x}^{\bar{\sigma}}$ satisfies
>
> $$x_i^{\bar{\sigma}} = \varphi\left(G^i_{\boldsymbol{\upsilon}}|^f\sigma\right),$$
>
> where $f = \text{out} \circ \varphi : \mathcal{H} \to \Omega_\perp$, and out$(-)$ is defined as $\perp$ on nonfinal states.

Proof. See the Appendix. ∎

To state and prove the characterization theorem for the interleaved case we just need an additional concept:

Definition 5 An infinite sequence $\sigma \in \{1, 2, \ldots, n\}^\infty$ is *fair* iff each $i \in \{1, 2, \ldots, n\}$ appears in σ infinitely often.

Theorem 5 Given a class $\mathcal{C}$, a relation R is computable in an interleaved way on $\mathcal{C}$ iff there is a monotonic function $f : \mathcal{H} \to \Omega_\perp$ such that for all fair sequences $\sigma \in \{1, 2, \ldots, n\}^\infty$, all $G \in \mathcal{C}$ and $\boldsymbol{\upsilon} \in \text{dom}(R_G)$

$$\boldsymbol{\upsilon}\ R_G \left\langle \bigvee_t f\left(G^1_{\boldsymbol{\upsilon}}|^f\sigma^0\cdots\sigma^t\right), \ldots, \bigvee_t f\left(G^n_{\boldsymbol{\upsilon}}|^f\sigma^0\cdots\sigma^t\right)\right\rangle$$

(in particular, when this happens the last vector is entirely nonbottom), and for all $\boldsymbol{\upsilon} \notin \text{dom}(R_G)$

$$\left\langle \bigvee_t f\left(G^1_{\boldsymbol{\upsilon}}|^f\sigma^0\cdots\sigma^t\right), \ldots, \bigvee_t f\left(G^n_{\boldsymbol{\upsilon}}|^f\sigma^0\cdots\sigma^t\right)\right\rangle = \langle\perp, \perp, \ldots, \perp\rangle.$$

Proof. For the left-to-right implication, consider a protocol P computing R, and an arbitrary fair sequence σ. Let φ and f be the functions defined as in the statement of Theorem 4, and recall that $x_i^{\overline{\sigma^0 \cdots \sigma^t}} = \varphi(G^i_{\boldsymbol{v}}|^f \sigma^0 \cdots \sigma^t)$ is the state of the network after the activation sequence $\overline{\sigma^0 \cdots \sigma^t}$. Thus, we have

$$\bigvee_t f\left(G^i_{\boldsymbol{v}}|^f \sigma^0 \cdots \sigma^t\right)$$
$$= \bigvee_t \mathrm{out}\left(\varphi\left(G^i_{\boldsymbol{v}}|^f \sigma^0 \cdots \sigma^t\right)\right) = \bigvee_t \mathrm{out}\left(x_i^{\overline{\sigma^0 \cdots \sigma^t}}\right),$$

but since $\bar{\sigma}$ is maximal (as σ is fair), the right (hence the left) side is defined iff $\boldsymbol{v} \in \mathrm{dom}(R_G)$, and the thesis follows immediately.

For the right-to-left implication, consider the protocol P that lets each processor i build $G^i_{\boldsymbol{v}}|^f \sigma$, and modify its enabling rule so that processor i is enabled in global state $\boldsymbol{x}$ only if $|x_i| \leq \min\{|x_j| \mid j \to i \wedge x_j \notin F\}$ (with a slight abuse of notation, by the *length* of a state we mean the length of its second component, which is a string). In other words, a processor is enabled only if its state is not longer than the state of all its nonterminated in-neighbours: we are in fact using the state length as a clock, which keeps track of the number of steps taken by each processor, and guarantees that processors do not drift too much.

If P terminates on $G \in \mathcal{C}$ with input $\boldsymbol{v} \in \mathrm{dom}(R_G)$ for every maximal activation sequence, it clearly computes R (note that there are no maximal finite sequences with nonterminated processors, as the processor with shortest state length among the nonterminated ones is always enabled, and that clearly P never terminates on inputs not in $\mathrm{dom}(R_G)$). Assume by contradiction that there is an infinite activation sequence σ of P. Of course, σ cannot be fair (or all processors would eventually terminate, by the property of f). So there is a smallest $T \in \mathbf{N}$ such that all processors appearing in $\sigma^T \sigma^{T+1} \cdots$ have infinite occurrences, and the set U of remaining processors (i.e., processors appearing only in $\sigma^0 \cdots \sigma^{T-1}$) is nonempty. Note that the in-neighbours of a processor in U^c are in U^c or terminated after $\sigma^0 \cdots \sigma^{T-1}$, as the length of the state of every processor in U^c grows without bound.

Now, we pad the sequence σ in such a way not to disturb the state of processors in U^c. More precisely, we define a sequence of finite strings τ_k as follows: $\tau_0 = \sigma^0 \sigma^1 \cdots \sigma^{T-1}$, and given τ_t, we build τ_{t+1} by concatenating all processors terminated after $\overline{\tau_t}$ in arbitrary order, a processor in U of shortest state length amongst the enabled ones, if any, and finally σ^{T+t}. We obtain the infinite sequence τ as the limit of the sequences τ_t above.

Note that the state of a processor in U^c after $\overline{\tau_t}$ is exactly its state after $\sigma^0 \cdots \sigma^{t+T}$. This is clearly true for $t = 0$; assume it is true for τ_t: the sequence τ_{t+1} is obtained by concatenating terminated processors, which will be deleted by restriction, possibly a processor $i \in U$, and the processor $\sigma^{T+t} \in U^c$. But, as we remarked, no processor in U is an in-neighbour of a processor in U^c, so the change of state of i does not influence the change of state of σ^{T+t}.

Moreover, the sequence τ is fair: suppose by contradiction there is a nonempty set of processors $V \subseteq U$ that does not appear in τ after a certain step. After a finite number of additional activations, the processors in V of shortest state length will have also shortest state length amongst processors in U. Because of this, moreover, they will be all enabled (possibly after some other steps, so that also the state length of their in-neighbours in U^c can grow suitably). Thus, one of them would be necessarily inserted in σ, a contradiction.

We remark that all processors deleted by restriction on a τ_t are terminated. A straightforward induction proof shows that in this case $G^i_{\boldsymbol{v}}|^f \tau_t = G^i_{\boldsymbol{v}}|^f \overline{\tau_t}$, so we have

$$G^i_{\boldsymbol{v}}|^f \tau_t = G^i_{\boldsymbol{v}}|^f \overline{\tau_t} = G^i_{\boldsymbol{v}}|^f \sigma^0 \sigma^1 \cdots \sigma^{T+t}$$

when $i \in U^c$. This is a contradiction, because for a sufficiently large t, $f\left(G^i_{\boldsymbol{v}}|^f \tau_t\right) \neq \perp$, and thus processor i, being terminated, should not appear in the sequence σ any more. ∎

5 A nontrivial example

In this section we show that in the class of prime rings it is possible to compute with interleaved activation the relation (with $\Upsilon = \Omega = \{*\}$) that is empty exactly for all rings of less than k notes. In other words, we require that

each processor terminates iff the network contains k processors or more: such a relation is clear not computable using synchronous activation, as at every step all processors are in the same state.

Given a view $H \in \mathcal{H}$, we can consider the set of all its subviews, that is, if $H = \upsilon \boldsymbol{H}_1 \cdots \boldsymbol{H}_k$ and

$$\boldsymbol{H}_i = \left\{ c_1 H_{i1}, \ldots, c_{l_i} H_{il_i} \right\}$$

we postulate that H is a subview of H, and every subview of H_{ij} is also a subview of H. The set of subviews of H has an order induced by that of $\mathcal{H}$, and we can consider the cardinality of the *maximum antichain*[2] contained there. The leading idea is that if we are running the view construction algorithm on a prime ring with at least k nodes, the cardinality of the maximum antichain of the view of each processor is to become at least k.

Let $f : \mathcal{H} \to \{*\}_\perp$ be the monotonic function which is nonbottom exactly on views with an antichain of at least k elements. First note that the subviews of $G^i|^f\sigma$ are all of the form $G^j|^f\tau$, with τ a prefix of σ. So, if G has less than k processors no antichain of at least k elements will ever arise in $G^i|^f\sigma$, and f will always evaluate to $\perp$.

On the other hand, by the characterization given in [5], we know that on an anonymous ring of prime size it is possible to assign a unique identifier to each processor. So Theorem 5 gives us a corresponding monotonic function $e : \mathcal{H} \to \mathbf{N}_\perp$, and we know that for every fair sequence σ there is a T such that the values $e\left(G^i|^e\sigma^0 \cdots \sigma^T\right)$ are distinct and nonbottom for all i. This in particular implies that the views $G^i|^e\sigma^0 \cdots \sigma^T$ are all incomparable.

Assume by contradiction that $f\left(G^i|^f\sigma^0 \cdots \sigma^t\right) = \perp$ for all t and i; then, in particular, there exist i and j such that $G^i|^f\sigma^0 \cdots \sigma^T \leq G^j|^f\sigma^0 \cdots \sigma^T$ (otherwise, ultimately all views would contain an antichain of size k). But our assumption on f implies that $G^i|^f\sigma^0 \cdots \sigma^T = G^i|^\perp\sigma^0 \cdots \sigma^T$. A straightforward induction proof shows that there is a monotonic function $E : \mathcal{H} \to \mathcal{H}$ such that $E\left(G^i|^\perp\sigma\right) = G^i|^e\sigma$: but then

$$\begin{aligned} G^i|^e\sigma^0 \cdots \sigma^T &= E\left(G^i|^\perp\sigma^0 \cdots \sigma^T\right) \\ &= E\left(G^i|^f\sigma^0 \cdots \sigma^T\right) \leq E\left(G^j|^f\sigma^0 \cdots \sigma^T\right) \\ &= E\left(G^j|^\perp\sigma^0 \cdots \sigma^T\right) = G^j|^e\sigma^0 \cdots \sigma^T, \end{aligned}$$

a contradiction. Thus, $f\left(G^i|^f\sigma^0 \cdots \sigma^T\right) \neq \perp$ for some i and T, and by strong connectivity for all i ultimately.

The discussion above shows how $\mathcal{H}$ extends the notion of view to the interleaved case. The authors have shown in previous work [6] that self-stabilizing and anonymous systems are strictly related in the synchronous case (the behaviours that are realizable anonymously are the same that can be realized in a self-stabilizing way), and the notion of view introduced here suggests that these results can be extended to the interleaved case. For instance, a simple modification to the protocol induced by f will give a self-stabilizing protocol computing the size of the network (of course, one needs a self-stabilizing view construction, but this is essentially the construction given in [6]). As a final note, we remark that most techniques used in the proof sketched above are general; for instance, for every monotonic function f there is a monotonic function $F : \mathcal{H} \to \mathcal{H}$ such that $F\left(G^i|^\perp\sigma\right) = G^i|^f\sigma$.

[2] An *antichain* in a partially ordered set is a subset whose elements are pairwise incomparable.

References

[1] ANGLUIN, D. Global and local properties in networks of processors. In *Proc. 12th Symposium on the Theory of Computing* (1980), pp. 82–93.

[2] ATTIYA, H., GORBACH, A., AND MORAN, S. Computing in totally anonymous asynchronous shared memory systems. In *12th international symposium on DIStributed Computing* (1998), vol. 1499 of *Lecture Notes in Computer Science*, p. 49.

[3] ATTIYA, H., SNIR, M., AND WARMUTH, M. K. Computing on an anonymous ring. *J. Assoc. Comput. Mach. 35*, 4 (1988), 845–875.

[4] BOLDI, P., CODENOTTI, B., GEMMELL, P., SHAMMAH, S., SIMON, J., AND VIGNA, S. Symmetry breaking in anonymous networks: Characterizations. In *Proc. 4th Israeli Symposium on Theory of Computing and Systems* (1996), IEEE Press, pp. 16–26.

[5] BOLDI, P., AND VIGNA, S. Computing vector functions on anonymous networks. In *SIROCCO '97. Proc. 4th International Colloquium on Structural Information and Communication Complexity* (1997), D. Krizanc and P. Widmayer, Eds., vol. 1 of *Proceedings in Informatics*, Carleton Scientific, pp. 201–214. An abstract appeared also as a Brief Announcement in *Proc. PODC '97*, ACM Press.

[6] BOLDI, P., AND VIGNA, S. Self-stabilizing universal algorithms. In *Self–Stabilizing Systems (Proc. of the 3rd Workshop on Self–Stabilizing Systems, Santa Barbara, California, 1997)* (1997), S. Ghosh and T. Herman, Eds., vol. 7 of *International Informatics Series*, Carleton University Press, pp. 141–156.

[7] MONK, J. D. *Introduction to Set Theory*. McGraw–Hill, New York, 1969.

[8] NAOR, M., AND STOCKMEYER, L. What can be computed locally? *SIAM J. Comput. 24*, 6 (1995), 1259–1277.

[9] NORRIS, N. Computing functions on partially wireless networks. In *Structure, Information and Communication Complexity. Proc. 2nd Colloquium SIROCCO '95* (1996), L. M. Kirousis and E. Kranakis, Eds., vol. 2 of *International Informatics Series*, Carleton University Press, pp. 53–64.

[10] YAMASHITA, M., AND KAMEDA, T. Computing functions on asynchronous anonymous networks. *Math. Systems Theory 29*, 4 (1996), 331–356. See also [12].

[11] YAMASHITA, M., AND KAMEDA, T. Computing on anonymous networks: Part I—characterizing the solvable cases. *IEEE Trans. Parallel and Distributed Systems 7*, 1 (1996), 69–89.

[12] YAMASHITA, M., AND KAMEDA, T. Erratum to computing functions on asynchronous anonymous networks. *Theory of Computing Systems (formerly Math. Systems Theory) 31*, 1 (1998).

Appendix

Proof of Theorem 1. The new protocol P' has state space $X' = X^+$ (finite nonempty sequences of states in X), final states given by $F' = X^*F$ (sequences with the last element in F), the same input function, and output function given by $\mathrm{out}'(x_1x_2\cdots x_p) = \mathrm{out}(x_p)$. The basic idea behind the construction of the new transition function δ' is that the q-th character of the state of processor i records the state that i would be in at the q-th step of a synchronous computation of P. Thus, processors must read from their neighbours as many "records" as possible to extend their own record list. Of course, the lists cannot be extended after adding a final state, and in this case we consider such lists silently extended by repeating their last element.

Formally,

$$\delta'(v, \{\!\{\langle c_1, v_1\rangle, \ldots, \langle c_h, v_h\rangle\}\!\}) = \begin{cases} v & \text{if } v \in F' \text{ or } q < |v| \\ vw & \text{otherwise,} \end{cases}$$

where $q = \min\{|v_k| \mid v_k \notin F'\}$ and w is defined as follows (we use $[-]_i$ to denote the i-th character of a string, or the last one if i exceeds the length of the string): consider the sequence $y_0, y_1, \ldots$ where $y_0 = [v]_{|v|}$ and

$$y_{k+1} = \delta\left(y_k, \{\!\{\langle c_1, [v_1]_{|v|+k}\rangle, \ldots, \langle c_h, [v_h]_{|v|+k}\rangle\}\!\}\right).$$

We set

$$w = \begin{cases} y_1 \cdots y_{q-|v|+1} & \text{if } q < \infty \\ y_1 \cdots y_{\min\{k \mid y_k = y_{k+1}\}} & \text{if } q = \infty \text{ and } \exists k\ y_k = y_{k+1} \\ \varepsilon & \text{otherwise.} \end{cases}$$

Some comments are in order. If all neighbours of a processor are in a final state, $q = \infty$ and the second clause of the definition of w implies that the new protocol emulates iteratively the old one until termination (otherwise no simulation at all is performed; this can happen only for unreachable states). Otherwise, $q - |v| + 1$ is the number of elements that we can add by emulation to our current list.

It should be clear that, by the very definition of δ', the q-th character of the state of a processor during a computation of P' on some network of $\mathcal{C}$ is just the state reached by the same processor after q steps of the synchronous computation of P on the same network with the same input. Moreover, if P is terminating on that input and the global state is not final, then there is a processor that is enabled, and its activation would extend its own local state (just take a processor whose local state has minimum length among those that are not in a final state). This proves that P' computes the same relation as P.

Finally, the fact that P' is well terminating is an immediate consequence of the previous considerations: one can treat unreachable states by suitably redefining the transition function on them. ∎

Proof of Lemma 1. We work by induction on t, proving both statements at the same time. The base case is straightforward: in the first case, if $f(\upsilon_i) = \bot$ then, by Definition 2, $G^i_{\boldsymbol{\upsilon}}|^f 1$ has height 1 (recall that every node has at least an incoming arc), and the second case is trivial.

Assume now $t > 0$. For the first statement, if $f\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) = \bot$, by definition $G^i_{\boldsymbol{\upsilon}}|^f t + 1 = \upsilon_i \left\{ \gamma(a) G^j_{\boldsymbol{\upsilon}}|^f t \right\}_{j \xrightarrow{a} i}$ and thus $h\left(G^i_{\boldsymbol{\upsilon}}|^f t + 1\right) = 1 + \max_{j \xrightarrow{a} i} h(G^j_{\boldsymbol{\upsilon}}|^f t)$. Since (assuming inductively the second statement) $f\left(G^i_{\boldsymbol{\upsilon}}|^f t - 1\right) \leq f\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) = \bot$, there is certainly a $j \to i$ s.t. $f\left(G^j_{\boldsymbol{\upsilon}}|^f t - 1\right) = \bot$, so by inductive hypothesis $h\left(G^j_{\boldsymbol{\upsilon}}|^f t\right) = t$, and $h\left(G^i_{\boldsymbol{\upsilon}}|^f t + 1\right) = t + 1$ (note that the height of a generic tree $G^i_{\boldsymbol{\upsilon}}|^f t$ is t at most).

For the second statement, the only relevant case is when $f\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) = \bot$; in this case, by induction $G^i_{\boldsymbol{\upsilon}}|^f t - 1 \leq G^i_{\boldsymbol{\upsilon}}|^f t$ and so

$$G^i_{\boldsymbol{\upsilon}}|^f t + 1 = \upsilon_i \left\{ \gamma(a) G^j_{\boldsymbol{\upsilon}}|^f t \right\}_{j \xrightarrow{a} i}$$
$$G^i_{\boldsymbol{\upsilon}}|^f t = \upsilon_i \left\{ \gamma(a) G^j_{\boldsymbol{\upsilon}}|^f t - 1 \right\}_{j \xrightarrow{a} i},$$

We just have to prove that if $h(G^j_{\boldsymbol{\upsilon}}|^f t - 1) < h(G^i_{\boldsymbol{\upsilon}}|^f t) - 1$, then equality holds. In the stated case, since $h(G^i_{\boldsymbol{\upsilon}}|^f t) = t$, we have $t > 1$ and $h(G^j_{\boldsymbol{\upsilon}}|^f t - 1) < t - 1$; thus, using the inductive hypothesis of the first statement, $f\left(G^j_{\boldsymbol{\upsilon}}|^f (t-2)\right) \neq \bot$, which finally implies $G^j_{\boldsymbol{\upsilon}}|^f t - 1 = G^j_{\boldsymbol{\upsilon}}|^f t$. ∎

Proof of Theorem 2. Firstly we define φ by recursion, letting $\varphi(\upsilon) = \mathrm{in}(\upsilon)$, and, for $T = \upsilon\{c_1 T_1, \ldots, c_k T_k\}$,

$$\varphi(T) = \delta(\varphi(\mathrm{prune}\,(T)), \{\langle c_1, \varphi(T_1)\rangle, \ldots, \langle c_k, \varphi(T_k)\rangle\}).$$

Note that $f = \mathrm{out} \circ \varphi$ is trivially monotonic and satisfies the additional hypothesis of Lemma 1: indeed, if $\varphi(T) \notin F$ then there is at least a T_i such that $\varphi(T_i) \notin F$ (because P is well terminating).

We work by induction on t, the case $t = 0$ being obvious. If $x^t_i \in F$, then $f\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) \neq \bot$, so $G^i_{\boldsymbol{\upsilon}}|^f t + 1 = G^i_{\boldsymbol{\upsilon}}|^f t$ and

$$x^{t+1}_i = x^t_i = \varphi\left(G^i_{\boldsymbol{\upsilon}}|^f t\right) = \varphi\left(G^i_{\boldsymbol{\upsilon}}|^f t + 1\right).$$

If $x^t_i \notin F$, by the induction hypothesis and using Corollary 2 we have

$$\begin{aligned}
&\varphi\left(G^i_{\boldsymbol{\upsilon}}|^f t + 1\right) \\
&= \delta\left(\varphi\left(\mathrm{prune}\left(G^i_{\boldsymbol{\upsilon}}|^f t + 1\right)\right), \left\{\langle \gamma(a), \varphi\left(G^j_{\boldsymbol{\upsilon}}|^f t\right)\rangle\right\}_{j \xrightarrow{a} i}\right) \\
&= \delta\left(\varphi\left(G^i_{\boldsymbol{\upsilon}}|^f t\right), \left\{\langle \gamma(a), \varphi\left(G^j_{\boldsymbol{\upsilon}}|^f t\right)\rangle\right\}_{j \xrightarrow{a} i}\right) \\
&= \delta\left(x^t_i, \left\{\langle \gamma(a), x^t_j\rangle\right\}_{j \xrightarrow{a} i}\right) = x^{t+1}_i. \quad ∎
\end{aligned}$$

Proof of Theorem 4. Firstly we define φ by recursion, letting $\varphi(\upsilon\varepsilon) = \mathrm{in}(\upsilon)$, and, for $H = \upsilon_1 \boldsymbol{H}_1 \cdots \boldsymbol{H}_k$ $(k > 0)$ we let $\varphi(H)$ be

$$\delta\left(\varphi(\upsilon \boldsymbol{H}_1 \cdots \boldsymbol{H}_{k-1}), \left\{\langle c_{k1}, \varphi(H_{k1})\rangle, \ldots, \langle c_{kl_k}, \varphi(H_{kl_k})\rangle\right\}\right).$$

Note that $f = \mathrm{out} \circ \varphi$ is trivially monotonic.

We work by induction on σ, the case $\sigma = \varepsilon$ being obvious. Consider the sequence σj: if $i \neq j$, then $G^i_{\boldsymbol{\upsilon}}|^f \sigma j = G^i_{\boldsymbol{\upsilon}}|^f \sigma$, and $x^{\overline{\sigma j}}_i = x^{\bar\sigma}_i$ (regardless whether $\overline{\sigma j} = \bar\sigma$ or $\overline{\sigma j} = \bar\sigma j$), hence the thesis; if instead $i = j$, we can consider the following subcases:

1. if $x^{\bar\sigma}_i \in F$ (i.e., $\mathrm{out}\left(x^{\bar\sigma}_i\right) = f\left(G^i_{\boldsymbol{\upsilon}}|^f \sigma\right) \neq \bot$), then by definition $G^i_{\boldsymbol{\upsilon}}|^f \sigma i = G^i_{\boldsymbol{\upsilon}}|^f \sigma$. But $\overline{\sigma i} = \bar\sigma$ in this case, so the thesis follows;

2. if $x^{\bar\sigma}_i \notin F$, assume that i is not enabled in the state $\boldsymbol{x}^{\bar\sigma}$, that is, $\delta\left(x^{\bar\sigma}_i, \left\{\langle \gamma(a), x^{\bar\sigma}_k\rangle\right\}_{k \xrightarrow{a} i}\right) = x^{\bar\sigma}_i$; in such case $\overline{\sigma i} = \bar\sigma$ and

$$\begin{aligned}
\varphi\left(G^i_{\boldsymbol{\upsilon}}|^f \sigma i\right) &= \varphi\left(\left(G^i_{\boldsymbol{\upsilon}}|^f \sigma\right)\left\{\langle \gamma(a), G^k_{\boldsymbol{\upsilon}}|^f \sigma\rangle\right\}_{k \xrightarrow{a} i}\right) \\
&= \delta\left(x^{\bar\sigma}_i, \left\{\langle \gamma(a), x^{\bar\sigma}_k\rangle\right\}_{k \xrightarrow{a} i}\right) = x^{\bar\sigma}_i = x^{\overline{\sigma i}}_i.
\end{aligned}$$

3. otherwise, $\overline{\sigma i} = \bar\sigma i$ and the thesis follows in the same way as in the last case. ∎