A Knowledge-Based Analysis of Global Function Computation*

Joseph Y. Halpern
Cornell University
Ithaca, NY 14853
halpern@cs.cornell.edu

Sabina Petride
Cornell University
Ithaca, NY 14853
petride@cs.cornell.edu

Abstract

Consider a distributed system N in which each agent has an input value and each communication link has a weight. Given a global function, that is, a function f whose value depends on the whole network, the goal is for every agent to eventually compute the value $f(N)$. We call this problem *global function computation*. Various solutions for instances of this problem, such as Boolean function computation, leader election, (minimum) spanning tree construction, and network determination, have been proposed, each under particular assumptions about what processors know about the system and how this knowledge can be acquired. We give a necessary and sufficient condition for the problem to be solvable that generalizes a number of well-known results [Attyia, Snir, and Warmuth 1988; Yamashita and Kameda 1996; Yamashita and Kameda 1999]. We then provide a *knowledge-based (kb) program* (like those of Fagin, Halpern, Moses, and Vardi [1995, 1997]) that solves global function computation whenever possible. Finally, we improve the message overhead inherent in our initial kb program by giving a *counterfactual belief-based program* [Halpern and Moses 2004] that also solves the global function computation whenever possible, but where agents send messages only when they believe it is necessary to do so. The latter program is shown to be implemented by a number of well-known algorithms for solving leader election.

1 Introduction

Consider a distributed system N in which each agent has an input value and each communication link has a weight. Given a global function, that is, a function f whose value depends on the whole network, the goal is for every agent to eventually compute the value $f(N)$. We call this problem *global function computation*. Many distributed protocols involve computing some global function of the network. This problem is typically straightforward if the network is known. For example, if the goal is to compute the spanning tree of the network, one can simply apply one of the well-known algorithms proposed by Kruskal or Prim. However, in a distributed setting, agents may have only local information, which makes the problem more difficult. For example, the algorithm proposed by Gallager, Humblet and Spira [1983] is known for its complexity.[1] Moreover, the algorithm does not work for all networks, although

*Work supported in part by NSF under grants CTC-0208535, ITR-0325453, and IIS-0534064, by ONR under grant N00014-02-1-0455, by the DoD Multidisciplinary University Research Initiative (MURI) program administered by the ONR under grants N00014-01-1-0795 and N00014-04-1-0725, and by AFOSR under grants F49620-02-1-0101 and FA9550-05-1-0055.

[1]Gallager, Humblet, and Spira's algorithm does not actually solve the minimum spanning tree as we have defined it, since agents do not compute the minimum spanning tree, but only learn relevant information about it, such as which of its edges lead in the direction of the root.

it is guaranteed to work correctly when agents have distinct inputs and no two edges have identical weights.

Computing shortest paths between nodes in a network is another instance of global function computation that has been studied extensively [Ford and Fulkerson 1962; Bellman 1958]. The well-known *leader election problem* [Lynch 1997] can also be viewed as an instance of global computation in all systems where agents have distinct inputs: the leader is the agent with the largest (or smallest) input. The difficulty in solving global function computation depends on what processors know. For example, when processors know their identifiers (names) and all ids are unique, several solutions for the leader election problem have been proposed, both in the synchronous and asynchronous settings [Chang and Roberts 1979; Le Lann 1977; Peterson 1982]. On the other hand, Angluin [1980], and Johnson and Schneider [1985] proved that it is impossible to deterministically elect a leader if agents may share names. In a similar vein, Attiya, Snir and Warmuth [1988] prove that there is no deterministic algorithm that computes a non-constant Boolean global function in a ring of unknown and arbitrarily large size if agents' names are not necessarily unique. Attiya, Gorbach, and Moran [2002] characterize what can be computed in what they call *totally anonymous shared memory systems*, where access to shared memory is anonymous.

We aim to better understand what agents need to know to compute a global function. We do this using the framework of *knowledge-based (kb) programs*, proposed by Fagin, Halpern, Moses and Vardi [1995, 1997]. Intuitively, in a kb program, an agent's actions may depend on his knowledge. To say that the agent with identity i knows some fact φ we simply write $K_i\varphi$. For example, if agent i sends a message msg to agent j only if he does not know that j already has the message, then the agent is following a kb program that can be written as

$$\mathbf{if}\ K_i(has_j(msg))\ \mathbf{then}\ \mathsf{skip}\ \mathbf{else}\ send(msg).$$

Knowledge-based programs abstract away from particular details of implementation and generalize classes of standard programs. They provide a high-level framework for the design and specification of distributed protocols. They have been applied to a number of problems, such as *atomic commitment* [Hadzilacos 1987], *distributed commitment* [Mazer and Lochovsky 1990], Byzantine agreement [Dwork and Moses 1990; Halpern, Moses, and Waarts 2001], sequence transmission [Halpern and Zuck 1992], and analyzing the TCP protocol [Stulp and Verbrugge 2002].

We first characterize when global function computation is solvable, i.e., for which networks N and global functions f agents can eventually learn $f(N)$. As we said earlier, whether or not agents can learn $f(N)$ depends on what they initially know about N. We model what agents initially know as a set $\mathcal{N}$ of networks; the intuition is that $\mathcal{N}$ is the set of all networks such that it is common knowledge that N belongs to $\mathcal{N}$. For example, if it is commonly known that the network is a ring, $\mathcal{N}$ is the set of all rings; this corresponds to the setting considered by Attiya, Snir and Warmuth [1988]. If, in addition, the size n of N is common knowledge, then $\mathcal{N}$ is the (smaller) set of all rings of size n. Yamashita and Kameda [1996] focus on three different types of sets $\mathcal{N}$: (1) for a given n, the set of all networks of size n, (2) for a fixed d, the set of all networks of diameter at most d, and (3) for a graph G, the set of networks whose underlying graph is G, for all possible labelings of nodes and edges. In general, the more that is initially known, the smaller $\mathcal{N}$ is. Our problem can be rephrased as follows: given N and f, for which sets $\mathcal{N}$ is it possible for all agents in N to eventually learn $f(N)$?

For simplicity, we assume that the network is finite and connected, that communication is reliable, and that no agent fails. Consider the following simple protocol, run by each agent in the network:

agents start by sending what they initially know to all of their neighbors; agents wait until they receive information from all their neighbors; and then agents transmit all they know on all outgoing links. This is a *full-information protocol*, since agents send to their neighbors everything they know. Clearly with the full-information protocol all agents will eventually know all available information about the network. Intuitively, if $f(N)$ can be computed at all, then it can be computed when agents run this full-information protocol. However, there are cases when this protocol fails; no matter how long agents run the protocol, they will never learn $f(N)$. This can happen because

1. although the agents actually have all the information they could possibly get, and this information suffices to compute the value of f, the agents do not know this;

2. although the agents have all the information they could possibly get (and perhaps even know this), the information does not suffice to compute the function value.

In Section 2, we illustrate these situations with simple examples. We show that there is a natural way of capturing what agents know in terms of *bisimilarity relations* [Milner 1989], and use bisimilarity to characterize exactly when global function computation is solvable. We show that this characterization provides a significant generalization of results of Attiya, Snir, and Warmuth [1988] and Yamashita and Kameda [1999].

We then show that the simple program where each agent just forwards all the new information it obtains about the network solves the global function computation problem whenever possible. It is perhaps obvious that, if anything works at all, this program works. We show that the program terminates with each agent knowing the global function value iff the condition that we have identified holds.

Our program, while correct, is typically not optimal in terms of the number of messages sent. Generally speaking, the problem is that agents may send information to agents who already know it or will get it via another route. For example, consider an oriented ring. A simple strategy of always sending information to the right is just as effective as sending information in both directions. Thus, roughly speaking, we want to change the program so that an agent sends whatever information he learns to a neighbor only if he does not know that the neighbor will eventually learn it anyway.

Since agents decide which actions to perform based on what they know, this will be a kb program. While the intuition behind this kb program is quite straightforward, there are subtleties involved in formalizing it. One problem is that, in describing kb programs, it has been assumed that names are commonly known. However, if the network size is unknown, then the names of all the agents in the network cannot be commonly known. Things get even more complicated if we assume that identifiers are not unique. For example, if identifiers are not unique, it does not make sense to write "agent i knows φ"; $K_i\varphi$ is not well defined if more than one agent can have the id i.

We deal with these problems using techniques introduced by Grove and Halpern [1995, 1993]. Observe that it makes perfect sense to talk about each agent acting based on his own knowledge by saying "if *I* know φ, then ...". I here represents the name each agent uses to refer to himself. This deals with self-reference; by using relative names appropriately, we can also handle the problem of how an agent refers to other agents.

A second problem arises in expressing the fact that an agent should send information to a neighbor only if the neighbor will not eventually learn it anyway. As shown by Halpern and Moses [2004] the most obvious way of expressing it does not work; to capture this intuition correctly we must use *counterfactuals*. These are statements of the form $\varphi > \psi$, which are read "if φ then ψ", but the "if

... then" is not treated as a standard material implication. In particular, the formula is not necessarily true if φ is false. In Section 3.1, we provide a kb program that uses counterfactuals which solves the global function computation problem whenever possible, while considerably reducing communication overhead.

As a reality check, for the special case of leader election in networks with distinct ids, we show in Section 5 that the kb program is essentially implemented by the protocols of Lann, Chang and Roberts [Le Lann 1977; Chang and Roberts 1979], and Peterson [1982], which all work in rings (under slightly different assumptions), and by the optimal flooding protocol [Lynch 1997] in networks of bounded diameter. Thus, the kb program with counterfactuals shows the underlying commonality of all these programs and captures the key intuition behind their design.

The rest of this paper is organized as follows. In Section 2, we give our characterization of when global function computation is possible. In Section 3 we describe the kb program for global function computation, and show how to optimize it so as to minimize messages. In Section 5, we show that the program essentially implements some standard solutions to leader election in a ring. We remark that to define kb programs with counterfactuals requires a lot of technical machinery, which can sometimes obscure the essential simplicity of the ideas. Thus, we defer the detailed formal definitions and the proofs of results to the appendix, giving only the essential ideas in the main part of the paper.

2 Characterizing when global function computation is solvable

We model a network as a directed, simple (no self-loops), connected, finite graph, where both nodes and edges are labeled. Each node represents an agent; its label is the agent's input, possibly together with the agent's name (identifier). Edges represent communication links; edge labels usually denote the cost of message transmission along links. Communication is reliable, meaning that every message sent is eventually delivered and no messages are duplicated or corrupted.

We assume that initially agents know their *local information*, i.e., their own input value, the number of outgoing links, and the weights associated with these links. However, agents do not necessarily know the weights on non-local edges, or any topological characteristics of the network, such as size, upper bound on the diameter, or the underlying graph. Additionally, agents may not know the identity of the agents they can directly communicate with, or if they share their names with other agents. In order to uniquely identify agents in a network N of size n, we label agents with "external names" $1, \ldots, n$. Agents do not necessarily know these external names; we use them for our convenience when reasoning about the system. In particular, we assume that the global function f does not depend on these external names; $f(N) = f(N')$ for any two networks N and N' that differ only in the way that nodes are labeled.

Throughout the paper we use the following notation: We write $V(N)$ for the set of agents in N and $E(N)$ for the set of edges. For each $i \in V(N)$, let $Out_N(i)$ be the set of i's neighbors on outgoing links, so that $Out_N(i) = \{j \in V(N) \mid (i,j) \in E(N)\}$; let $In_N(i)$ be the set of i's neighbors on incoming links, so that $In_N(i) = \{j \in V(N) \mid (j,i) \in E(N))\}$; let $in_N(i)$ denote i's input value. Finally, if e is an edge in $E(N)$, let $w_N(e)$ denote e's label.

We want to understand, for a given network N and global function f, when it is possible for agents to eventually know $f(N)$. This depends on what agents know about N. As mentioned in the introduction, the general (and unstated) assumption in the literature is that, besides their local information, whatever agents know initially about the network is *common knowledge*. We start our analysis by making the same assumption, and characterize the initial common knowledge as a set $\mathcal{N}$ of networks.

In this section, we assume that agents are following a full-information protocol. We think of the protocol as proceeding in *rounds*: in each round agents send to all neighbors messages describing all the information they have; messages are stamped with the round number; round k for agent i starts after he has received all round $k-1$ messages from his neighbors (since message delivery is reliable, this is guaranteed to happen). The round-based version of the full-information protocol makes sense both in synchronous and asynchronous settings, and for any assumptions about the order in which messages are delivered.

Intuitively, the full-information protocol reduces uncertainty. For example, suppose that $\mathcal{N}$ consists of all unidirectional 3-node rings, and let N be a three node ring in which agents have inputs a, b, and c, and all edges have the same weight w. Let i be the external name of the agent with input a. Initially, i considers possible all 3-nodes rings in which the weight on his outgoing edge is w and his input is a. After the first round, i learns from his incoming neighbor, who has external name j, that j's incoming edge also has weight w, and that j has input c. Agent j learns in the first round that his incoming neighbor has input b and that his incoming edge also has weight w. Agent j communicates this information to i in round 2. At the end of round 2, i knows everything about the network N, as do the other two agents. Moreover, he knows exactly what the network is. But this depends on the fact that i knows that the ring has size 3.

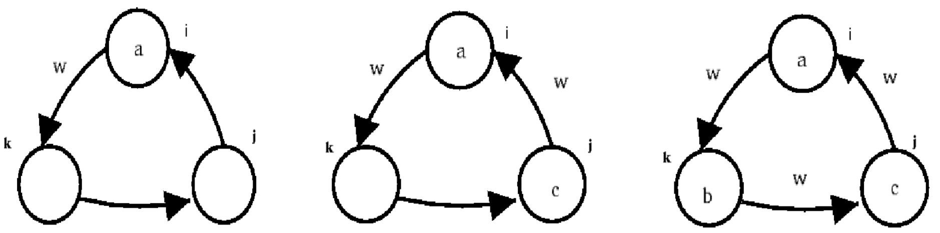

Figure 1: How i's information changes with the full-information protocol.

Now consider the same network N, but suppose that agents do not know the ring size, i.e., $\mathcal{N}$ is the set of all unidirectional rings, of all possible sizes and for all input and weight distributions. Again, at the end of round 2, agent i has all the information that he could possibly get, as do the other two agents. However, at no point are agents able to distinguish the network N from a 6-node ring N' in which agents look just like the agents on the 3-node ring (see Figure 2). Consider the pair of agents i in N and i' in N'. It is easy to check that these agents get exactly the same messages in every round of the full-information protocol. Thus, they have no way of distinguishing which is the true situation. If the function f has different values on N and N', then the agents cannot compute $f(N)$. On the other hand, if $\mathcal{N}$ consists only of networks where inputs are distinct, then i realizes at the end of round 2 that he must be k's neighbor, and then he knows the network configuration.

We want to characterize when agent i in network N thinks he could be agent i' in network N'. Intuitively, at round k, i thinks it possible that he could be i' if there is a bijection μ that maps i's incoming neighbors to i''s incoming neighbors such that, at the previous round $k-1$, each incoming neighbor j of i thought that he could be $\mu(j)$.

Definition 2.1: Given networks N and N' and agents $i \in V(N)$ and $i' \in V(N')$, i and i' are 0-*bisimilar*,

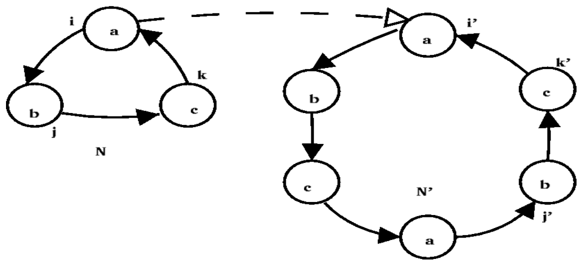

Figure 2: Two indistinguishable networks.

written $(N, i) \sim_0 (N', i')$, iff

- $in_N(i) = in_{N'}(i')$;
- there is a bijection $f^{out} : Out_N(i) \longrightarrow Out_{N'}(i')$ that preserves edge-labels; that is, for all $j \in Out_N(i)$, we have $w_N(i, j) = w_{N'}(i', f^{out}(j))$.

For $k > 0$, i and i' are *k-bisimilar*, written $(N, i) \sim_k (N', i')$, iff

- $(N, i) \sim_0 (N', i')$, and
- there is a bijection $f^{in} : In_N(i) \longrightarrow In_{N'}(i')$ such that for all $j \in In_N(i)$
  - $w_N(j, i) = w_{N'}(f^{in}(j), i')$,
  - the (j, i) edge is bidirectional iff the $(f^{in}(j), i')$ edge is bidirectional, and
  - $(N, j) \sim_{k-1} (N', f^{in}(j))$.

Note that $\sim_k$ is an equivalence relation on the set of pairs (N, i) with $i \in V(N)$, and that $\sim_{k+1}$ is a refinement of $\sim_k$.

The following lemma relates bisimilarity and the full-information protocol:

Lemma 2.2: *The following are equivalent:*

(a) $(N, i) \sim_k (N', i')$.

(b) Agents $i \in V(N)$ and $i' \in V(N')$ have the same initial local information and receive the same messages in each of the first k rounds of the full-information protocol.

(c) If the system is synchronous, then i and i' have the same initial local information and receive the same messages in each of the first k rounds of every deterministic protocol.

Proof: We first prove that (a) implies (c). Let P be an arbitrary deterministic protocol. The proof proceeds by induction, with the base case following from the definition of $\sim_0$. Suppose that, if $(N, i) \sim_k (N', i')$, then i and i' start with the same local information and receive same information in each of the first k rounds of protocol P and that $(N, i) \sim_{k+1} (N', i')$. Then $(N, i) \sim_k (N', i')$, and there exists a bijection $f^{in} : In_N(i) \longrightarrow In_{N'}(i)$ such that $(N, j) \sim_k (N', f^{in}(j))$ for all $j \in In_N(i)$. From the inductive hypothesis, it follows that i and i' have the same initial information and receive the same messages in the first k rounds of P; similarly, for each j incoming neighbor of i, j and $f^{in}(j)$ have same initial information and receive same messages in each of the first k rounds of P. Hence, j and $f^{in}(j)$ have the same local state at time k and, since P is deterministic, j sends i the same messages as $f^{in}(j)$ sends to i'. Thus, i and i' receive same messages in round $k+1$ of protocol P.

To prove that (c) implies (b), it suffices to notice that the full-information protocol is a special case of a deterministic protocol and that, given how we have defined rounds in an asynchronous setting, i receives the same messages in round k of the full-information protocol in both the synchronous and asynchronous case.

Finally, we prove that (b) implies (a) by induction on k. For $k = 0$, it is clear from Definition 2.1 that $(N, i) \sim_0 (N', i')$ exactly when i and i' have the same initial local information. For the inductive step, suppose that i and i' have the same initial local information and receive the same messages at each round $k' \leq k+1$. We can then construct a mapping, say f^{in}, from $In_N(i)$ to $In_{N'}(i')$ such that for all $j \in In_N(i)$, the information that i receives from j is the same as the information that i' receives from $f^{in}(j)$ in each of the first $k+1$ rounds. Since j is following a full-information protocol, it follows that j must have the same initial local information as j' and that j and j' receive the same messages in each of the first k rounds. By the induction hypothesis, $(N, j) \sim_k (N', f^{in}(j))$. Since part of i's information from j is also the weight of edge (j, i), f^{in} must preserve edge-weights. Thus, $(N, i) \sim_{k+1} (N', i')$. ∎

Intuitively, if the function f can be computed on N, then it can be computed using a full-information protocol. The value of f can be computed when f takes on the same value at all networks that the agents consider possible. The round at which this happens may depend on the network N, the function f, and what it is initially known. Moreover, if it does not happen, then f is not computable. Using Lemma 2.2, we can characterize if and when it happens.

Theorem 2.3: *The global function f can be computed on networks in $\mathcal{N}$ iff, for all networks $N \in \mathcal{N}$, there exists a constant $k_{\mathcal{N},N,f}$, such that, for all networks $N' \in \mathcal{N}$, all $i \in V(N)$, and all $i' \in V(N')$, if $(N, i) \sim_{k_{\mathcal{N},N,f}} (N', i')$ then $f(N') = f(N)$.*

Proof: First suppose that the condition in the statement of the theorem holds. At the beginning of each round k, each agent i in the network proceeds as follows. If i received the value of f in the previous round, then i forwards the value to all of its neighbors and terminates; otherwise, i computes f's value on all the networks N' such that there exists an i' such that agent i' would have received the same messages in the first $k-1$ rounds in network N' as i actually received. (By Lemma 2.2, these are just the pairs (N', i') such that $(N', i') \sim_{k-1} (N, i)$.) If all the values are equal, then i sends the value to all his neighbors and terminates; otherwise, i sends whatever new information he has received about the network to all his neighbors.

Let k_i be the first round with the property that for all $N' \in \mathcal{N}$ and i' in N', if $(N, i) \sim_{k_i} (N', i')$, then $f(N') = f(N)$. (By assumption, such a k_i exists and it is at most $k_{\mathcal{N},N,f}$.) It is easy to see that, by round k_i, i learns the value of $f(N)$, since either i gets the same messages that it gets in the full-information protocol up to round k_i or it gets the function value. Thus, i terminates by the end of round $k_i + 1$ at the latest, after sending the value of f, and the protocol terminates in at most $k_{\mathcal{N},N,f} + 1$ rounds. Clearly all agents learn $f(N)$ according to this protocol.

Now suppose that the condition in the theorem does not hold and, by way of contradiction, that the value of f can be computed by some protocol P on all the networks in $\mathcal{N}$. There must exist some network N for which the condition in the theorem fails. Consider a run where all messages are delivered synchronously. There must be some round k such that all agents in N have computed the function value by round k. Since the condition fails, there must exist a network $N' \in \mathcal{N}$ and agents $i \in V(N)$ and $i' \in V(N')$ such that $(N, i) \sim_k (N', i')$ and $f(N) \neq f(N')$. By Lemma 2.2, i and i' have the same initial information and receive the same messages in the first k rounds of protocol P. Thus, they must output the same value for the function at round k. But since $f(N) \neq f(N')$, one of these answers must be wrong, contradicting our assumption that P computes the value of f in all networks in $\mathcal{N}$. ∎

Intuitively, $k_{\mathcal{N},N,f}$ is a round at which each agent i knows that f takes on the same value at all the networks i considers possible at that round. Since we are implicitly assuming that agents do not forget, the set of networks that agent i considers possible never grows. Thus, if the function f takes on the same value at all the networks that agent i considers possible at round k, then f will take on the same value at all networks that i considers possible at round $k' > k$, so every agent knows the value of $f(N)$ in round $k_{\mathcal{N},N,f}$. In some cases, we can provide a useful upper bound on $k_{\mathcal{N},N,f}$. For example, if $\mathcal{N}$ consists only of networks with distinct identifiers, or, more generally, of networks in which no two agents are *locally* the same, i.e., $(N, i) \not\sim_0 (N, j)$ for all $i \neq j$, then we can take $k_{\mathcal{N},N,f} = diam(N) + 1$, where $diam(N)$ is the diameter of N.

Theorem 2.4: *If initially it is common knowledge that no two agents are locally the same, then all global functions can be computed; indeed, we can take $k_{\mathcal{N},N,f} = diam(N) + 1$.*

Proof: Since $f(N) = f(N')$ if N and N' are isomorphic, it suffices to show that $(N, i) \sim_{diam(N)+1} (N', i')$ implies that N and N' are isomorphic for all $N, N' \in \mathcal{N}$. First observe that, by an easy induction on k, if there is a path of length $k \leq diam(N)$ from i to j in N, then there must exist a node $j' \in V(N')$ such that there is a path from i' to j' of length k and $(N, j) \sim_{diam(N)+1-k} (N', j')$. Moreover, note that j' must be unique, since if $(N, j) \sim_{diam(N)+1-k} (N', j'')$, then j, j', and j'' must be locally the same and, by assumption, no distinct agents in N' are locally the same. Define a map h from N to N' by taking $h(j) = j'$. This map is 1-1, since if $h(j_1) = h(j_2)$, then j_1 and j_2 must be locally the same, and hence identical.

Let N'' be the subgraph of N' consisting of all nodes of distance at most $diam(N)$ from i'. An identical argument shows that there is a 1-1 map h' from N'' to N such that j' and $h'(j')$ are locally the same for all $j' \in V(N'')$. The function h' is the inverse of h, since $h(h'(j'))$ and j' are locally the same, and hence identical, for all $j' \in V(N)$. Finally, we must have that h is a graph isomorphism from N to N'', since the fact j and $h(j)$ are locally the same guarantee that they have the same labels, and if $(j_1, j_2) \in E(N)$, then $(h(j), h(j')) \in E(N'')$ and the two edges have the same label.

It remains to show that $N' = N''$. Suppose not. Then there is a node $j_1 \in V(N')$ of distance $diam(N) + 1$ from i'. Let $j_2 \in V(N)$ be such that j_1 is an outgoing neighbor of j_2 and the distance from i' to j_2 is $diam(N)$. By construction, $j_2 \in V(N'')$; by our previous argument, there is a node $j_3 \in V(N)$ such that $(N, j_3) \sim_1 (N', j_2)$. Since j_2 and j_3 are locally the same, they must have the same number of outgoing links, say m. That means that there are m nodes in N that have j_3 as an incoming neighbor, say $i_1, \ldots, i_m$. Thus, each of $h(i_1), \ldots, h(i_m)$, all of which are in N'', must have j_3 as an incoming neighbor. But j_3 has only m outgoing edges, and one of them goes to j_2, which is not in N''. This is a contradiction. ∎

Attiya, Snir, and Warmuth [1988] prove an analogue of Lemma 2.2 in their setting (where all networks are rings) and use it to prove a number of impossibility results. In our language, these impossibility results all show that there does not exist a k such that $(N, i) \sim_k (N', i')$ implies $f(N) = f(N')$ for the functions f of interest, and thus are instances of Theorem 2.3.[2]

Yamashita and Kameda characterize when global functions can be computed in undirected networks (which have no weights associated with the edges), assuming that an upper bound on the size of the network is known. They define a notion of *view* and show that two agents have the same information whenever their views are *similar* in a precise technical sense; $f(N)$ is computable iff for all networks N' such that agents in N and N' have similar views, $f(N') = f(N)$. Their notion of similarity is essentially our notion of bisimilarity restricted to undirected networks with no edge labels. Thus, their result is a special case of Theorem 2.3 for the case that $\mathcal{N}$ consists of undirected networks with no edge labels of size at most n^* for some fixed constant n^*; they show that $k_{\mathcal{N},N,f}$ can be taken to be n^* in that case. Not only does our result generalize theirs, but our characterization is arguably much cleaner.

Theorem 2.4 sheds light on why the well-known protocol for minimum spanning tree construction proposed by Gallager, Humblet, and Spira [1983] can deal both with systems with distinct ids (provided that there is a commonly-known ordering on ids) and for networks with identical ids but distinct edge-weights. These are just instances of situations where it is common knowledge that no two agents are locally the same.

3 A standard program for global function computation

3.1 Standard programs with shared names

A standard *program* Pg has the form

$$\begin{array}{l} \textbf{if } t_1 \textbf{ then } \mathsf{act}_1 \\ \textbf{if } t_2 \textbf{ then } \mathsf{act}_2 \\ \ldots, \end{array}$$

where the t_js are standard tests (possibly involving temporal operators such as $\Diamond$), and the act_js are actions. The intended interpretation is that agent i runs this program forever. At each point in time, i nondeterministically executes one of the actions act_j such that the test t_j is satisfied; if no such action exists, i does nothing. We sometime use obvious abbreviations like **if** $\ldots$ **then** $\ldots$ **else**.

Following Grove and Halpern [Grove 1995; Grove and Halpern 1993] (GH from now on), we distinguish between agents and their names. We assume that programs mention only names, not agents (since in general the programmer will have access only to the names, which can be viewed as denoting roles). We use $\mathbf{N}$ to denote the set of all possible names and assume that one of the names is I. In the semantics, we associate with each name the agent who has that name. We assume that each agent has a way of naming his neighbors, and gives each of his neighbors different names. However, two different agents may use the same name for different neighbors. For example, in a ring, each agent may name his neighbors L and R; in an arbitrary network, an agent whose outdegree is d may refer to his outgoing neighbors as 1, 2, ..., d. We allow actions in a program to depend on names, so the meaning of an action may depend on which agent is running it. For example, in our program for global function computation, if i uses name $\mathbf{n}$ to refer to his neighbor j, we write i's action of sending message msg to j as $send_{\mathbf{n}}(msg)$. Similarly, if A is a set of names, then we take $send_A(msg)$ to be the action of

[2]We remark that Attiya, Snir, and Warmuth allow their global functions to depend on external names given to agents in the network. This essentially amounts to assuming that the agent's names are part of their input.

sending msg to each of the agents in A (and not sending anything to any other agents). Let $\mathbf{Nbr}$ denote the neighbors of an agent, so that $send_{\mathbf{Nbr}}(msg)$ is the action of sending msg to all of an agent's neighbors.

We assume that message delivery is handled by the channel (and is not under the control of the agents). In the program, we use a primitive proposition $some\_new\_info$ that we interpret as true for agent i iff i has received some new information; in our setting, that means that i has learned about another agent in the network and his input, has learned the weight labeling some edges, or has learned that there are no further agents in the network. (Note that in the latter case, i can also compute the function value. For example, in doing leader election on a unidirectional ring, if i gets its id back after sending it around the network, then i knows that it has heard from all agents in the network, and can then compute which agent has the highest id.) Note that $some\_new\_info$ is a proposition whose truth is relative to an agent. As already pointed out by GH, once we work in a setting with relative names, then both propositions and names need to be interpreted relative to an agent; we make this more precise in the next section. In the program, the action $send_{\mathbf{n}}(new\_info)$ has the effect of i sending $\mathbf{n}$ whatever new information i learned.

With this background, we can describe the program for global function computation, which we call Pg^{GC}; each agent runs the program

$$\mathbf{if}\ some\_new\_info\ \mathbf{then}\ send_{\mathbf{Nbr}}(new\_info); receive,$$

where the $receive$ action updates the agent's state by receiving any messages that are waiting to be delivered. As written, Pg^{GC} does not terminate; however, we can easily modify it so that it terminates if agents learn the function value. (They will send at most one message after learning the function value.)

We would like to prove that Pg^{GC} solves the global function computation problem. To do this, we need to give precise semantics to programs; that is the subject of the next section.

3.2 Protocols, systems, and contexts

We interpret programs in the *runs and systems* framework of Fagin et al. [1995], adapted to allow for names. We start with a possibly infinite set $\mathcal{A}$ of agents. At each point in time, only finitely many agents are present. Each of these agents i is in some local state l_i. The *global state* of the system at a particular point is a tuple s consisting of the local states of the agents that exist at that point. Besides the agents, it is also convenient to assume that there is an *environment state*, which keeps track of everything relevant to the system not included in the agents' states. In our setting, the environment state simply describes the network.

A *run* is a function from time (which we take here to range over the natural numbers) to global states. Intuitively, a run describes the evolution of the system over time. With each run, we associate the set of agents that exist in that run. For simplicity, we assume that the set of agents is constant over the run; that is, we are not allowing agents to enter the system or leave the system. However, different sets of agent may be associated with different runs. (While this is appropriate in our setting, it is clearly not appropriate in general. We can easily extend the framework presented here to allow agents to enter or leave the system.) Let $\mathcal{A}(r)$ denote the agents present in run r. A pair (r, m) consisting of a run r and time m is called a *point*. If $i \in \mathcal{A}(r)$, we use $r_i(m)$ to denote agent i's local state at the point (r, m). A *system* $\mathcal{R}$ consists of a set of runs.

In a *system for global function computation*, each agent's initial local information is encoded in the agent's local state; it must be consistent with the environment. For example, if according to the environment the network is a bidirectional ring, each agent must have two outgoing edges according to its local state. We assume that agents have *perfect recall*, so that they keep track in their local states of everything that they have heard and when they heard it. This means that, in particular, the local state of an agent encodes whether the agent has obtained new information about the network in a given round k.

We are particularly interested in systems generated by protocols. A protocol P_i for agent i is a function from i's local states to nonempty sets of actions that i may perform. If the protocol is deterministic, then $P_i(\ell)$ is a singleton for each local state ℓ. A *joint protocol* is a tuple $P = \{P_i : i \in \mathcal{A}\}$, which consists of one protocol for each agent.

We can associate with each joint protocol P a system, given a *context*. A context describes the environment's protocol, the initial states, the effect of actions, and the association of names with agents. Since names are relative to agents, we do the association using a *naming function* $\mu : \mathcal{G} \times \mathcal{A} \times \mathbf{N} \to \mathcal{A}$, where $\mathcal{G}$ is the set of global states. Intuitively, $\mu(g, i, \mathbf{n}) = j$ if agent i assigns name $\mathbf{n}$ to agent j at the global state g. Thus, we take a context γ to be a tuple $(P_e, \mathcal{G}_0, \tau, \mu)$, where P_e is a protocol for the environment, $\mathcal{G}_0$ is a set of initial global states, τ is a *transition function*, and μ is a naming function.[3] The environment is viewed as running a protocol just like the agents; its protocol is used to capture, for example, when messages are delivered in an asynchronous system. The transition function τ and naming function μ determine a mapping denoted τ_μ associating with each *joint action* (a tuple consisting of an action for the environment and one for each of the agents) a *global state transformer*, that is, a mapping from global states to global states. Note that we need the naming function since actions may involve names. For the simple programs considered in this paper, the transition function will be almost immediate from the description of the global states.

We focus in this paper on a family of contexts that we call *contexts for global function computation*. Intuitively, the systems that represent programs in a context for global function computation are systems for global function computation. A context $\gamma^{GC} = (P_e, \mathcal{G}_0, \tau, \mu)$ for global function computation has the following features:

- The environment's protocol P_e controls message delivery and is such that all messages are eventually delivered, and no messages are duplicated or corrupted.
- The initial global states are such that the environment's state records the network N and agent i's local state records agent i's initial local information; we use N_r to denote the network in a run r (as encoded by the initial global state in r).
- The transition function τ_μ is such that the agents keep track of all messages sent and delivered and the set of agents does not change over time. That is, if s is a global state, act is a joint action, and $s' = \tau_\mu(\mathsf{act})(s)$, then $\mathcal{A}(s) = \mathcal{A}(s')$ and agent i's local state in s' is the result of appending all messages that i sent and received as a result of action act to i's local state in s. We assume that τ_μ is such that the action $send_{\mathbf{n}}(new\_info)$ has the appropriate effect, i.e., if $send_{\mathbf{n}}(new\_info)$ is agent i's component of a joint action act and agent i gives agent j name $\mathbf{n}$ in the global state s (note here we need the assumption that the naming function μ depends only on the global state)

[3]Fagin et al. [1995] also have a component of the context that describes the set of "allowable" runs. This plays a role when considering issues like fairness, but does not play a role in this paper, so we omit it for simplicity. Since they do not consider names, they do not have a component μ in their contexts.

and $s' = \tau_\mu(\mathsf{act})(s)$, then in s', j's local state records the fact that j has received the information from i.

In the following, we will denote the set of all networks encoded in the initial global states of a context γ^{GC} for global function computation as $\mathcal{N}(\gamma^{GC})$.

A run r is consistent with a joint protocol P if it could have been generated when running P. Formally, run r is *consistent with joint protocol P in context γ* if its initial global state $r(0)$ is one of the initial global states $\mathcal{G}_0$ given in γ, and for all m, the transition from global state $r(m)$ to $r(m+1)$ is the result of performing one of the joint actions specified by P according to the agents in r, and the environment protocol P_e (given in γ) in the global state $r(m)$. That is, if $P = \{P_i : i \in \mathcal{A}\}$ and P_e is the environment's protocol in context γ, then $r(0) \in \mathcal{G}_0$, and if $r(m) = (\ell_e, \{\ell_i : i \in \mathcal{A}(r)\})$, then there must be a joint action $(\mathsf{act}_e, \{\mathsf{act}_i : i \in r(\mathcal{A})\})$ such that $\mathsf{act}_e \in P_e(\ell_e)$, $\mathsf{act}_i \in P_i(\ell_i)$ for $i \in r(\mathcal{A})$, and $r(m+1) = \tau_\mu(\mathsf{act}_e, \{\mathsf{act}_i : i \in r(\mathcal{A})\})(r(m))$ (so that $r(m+1)$ is the result of applying the joint action $(\mathsf{act}_e, \{\mathsf{act}_i : i \in \mathcal{A}\})$ to $r(m)$. For future reference, we will say that a run r is *consistent with* γ if r is consistent with *some* joint protocol P in γ. A system $\mathcal{R}$ *represents* a joint protocol P in a context γ if it consists of all runs consistent with P in γ. We use $\mathbf{R}(P, \gamma)$ to denote the system representing P in context γ.

We want to associate with a program a protocol. To do this, we need to interpret the tests in the program. In doing so, we need to consider the fact that tests in the programs we consider here may contain names. This is the case for example of leader election programs in a ring network, where an agent may send a message only if his identifier is larger than his left neightbor's. We can write this as $id_I > id_L$, and clearly this test holds for the agent with maximum id, but does not hold for the agent with minimum id. This is why we need to interpret the tests in a program relative to an agent and with respect to a naming function μ that resolves names relative to the agent. Given a set Φ of primitive propositions, let an *interpretation* π be a mapping that associates with each naming function μ a function $\pi_\mu : \mathcal{G} \times \mathcal{A} \times \Phi \rightarrow \{\mathit{true}, \mathit{false}\}$. Intuitively, $\pi_\mu(g, i, p) = \mathit{true}$ if p is true at the global state g relative to agent i. Furthermore, we need to ensure that the interpretation is consistent, in the sense that if $id_I > id_L$ is interpreted as true in a global state g with respect to agent i, and i's left neighbor refers to i as his right neighbor, then $id_R > id_I$ is taken as true in same global state, this time when interpreted relative to i's left neighbor. To formalize this, we take Φ' to be the set of all propositions in Φ with relative names replaced by "external names" 1, …, n, and take functions $\pi' : \mathcal{G} \times \Phi' \rightarrow \{\mathit{true}, \mathit{false}\}$ to be *objective interpretation functions*. We say that π_μ is *consistent* if there exists an objective interpretation π' such that, for all global states g, agents i and tests p in Φ, $\pi_\mu(g, i, p) = \mathit{true}$ if and only if $\pi'(g, p') = \mathit{true}$, where p' is just like p, except that all names $\mathbf{n}$ are replaced by the external name $\mu(g, i, \mathbf{n})$. In the following, we will focus only on contexts γ and interpretations π such that π_μ (for μ the naming function in γ) is consistent. Of course, we can extend π_μ to arbitrary propositional formulas, in the standard way; for example, we take $\pi_\mu(g, i, \neg\varphi) = \mathit{true}$ iff $\pi_\mu(g, i, \varphi) = \mathit{false}$, $\pi_\mu(g, i, \varphi \wedge \psi) = \mathit{true}$ iff $\pi_\mu(g, i, \varphi) = \mathit{true}$ and $\pi_\mu(g, i, \psi) = \mathit{true}$, etc.

An interpretation is *local* (for program Pg and in context γ) if the tests φ in Pg depend only on the local state, in the sense that if ℓ is agent i's local state in the global state g and also agent j's local state in the global state g', then $\pi_\mu(g, i, \varphi) = \mathit{true}$ iff $\pi_\mu(g', j, \varphi) = \mathit{true}$. In this case, we write $\pi_\mu(\ell, \varphi) = \mathit{true}$. Given an interpretation π that is local, we can associate with a program Pg for agent i a protocol Pg^{π_μ}. We define $Pg^{\pi_\mu}(\ell) = \{\mathsf{act}_j \mid \pi_\mu(\ell, t_j) = \mathit{true}\}$ if there exist tests t_j such that $\pi_\mu(\ell, t_j) = \mathit{true}$, and take $Pg^{\pi_\mu}(\ell) = \mathsf{skip}$ otherwise. Define $\mathbf{I}(Pg, \gamma, \pi) = \mathbf{R}(Pg^{\pi_\mu}, \gamma)$, for μ the naming function in context γ.

An *interpreted context for global function computation* is a pair (γ, π), where γ is a context for global function computation and π_μ interprets $some\_new\_info$ appropriately (so that $\pi_\mu(g, i, some\_new\_info)$ $=$ *true* if i received some new information about the network in g and has not sent a message since receiving that information).

For the purpose of global function computation, we often talk about agents *knowing* a fact about the network, some piece of information, or the function value, and how this knowledge changes during a run of a protocol like $(Pg^{GC})^{\tau_\mu}$. Intuitively, this says that, regradless of the agent's uncertainity about the network, and in general about the global state he is in, φ holds. i's uncertainity about the global world comes from two sources: i's uncertainty about the local states of other agents, and i's uncertainity about his own identity and the identities of the other agents he can refer to by certain names. More precesily, when in some local state $\ell = r_i(m)$, i cannot distinguish between the global world $r(m)$ and any global world $r'(m')$ such that there exists an agent i' with same local state as i, i.e., $r'_{i'}(m') = \ell$. In the following, we will a tuple (r, m, i) a *situation*, and we will say that situations (r, m, i) and (r', m', i') are indistinguishable to agent i if i thinks possible he is i' in $r'(m')$, i.e., $r_i(m) = r'_{i'}(m')$. We define an *extended interpreted system* to be a tuple $\mathcal{I} = (\mathcal{R}, \pi, \mu)$, where $\mathcal{R}$ is a system, π is an interpretation, and μ is a naming function. We say that fact φ holds at situation (r, m, i) and with respect to interpreted system $\mathcal{I}$, denoted as $(\mathcal{I}, r, m, i) \models \varphi$, precisely when $\pi_\mu(r(m), i, \varphi) =$ *true*. We can now formalize the fact that i knows φ at point (r, m) as the condition that φ holds at all situations intistinguishable to i from (r, m, i), i.e., $(\mathcal{I}, r', m', i') \models \varphi$ for all situations (r', m', i') in $\mathcal{I}$ with $r'_{i'}(m') = r_i(m)$.

Program Pg solves the global function computation problem for function f in the interpreted context (γ^{GC}, π) if and only if, in all runs r of $\mathbf{I}(Pg, \gamma^{GC}, \pi)$, eventually all agents in $\mathcal{A}(r)$ know the value $f(N_r)$. That is, for all such runs r, there exists a time m such that, for all agents i in $\mathcal{A}(r)$, f takes the same value $f(N_r)$ on all networks i thinks possible when in local state $r_i(m)$, i.e., on all networks in runs r' such that there exists a time m' and an agent i' with $r'_{i'}(m') = r_i(m)$.

3.3 Proving the correctness of Pg^{GC}

Theorem 3.1: *If f and $\mathcal{N}(\gamma^{GC})$ satisfy the condition in Theorem 2.3, then Pg^{GC} solves the global function computation problem for f in all interpreted contexts (γ^{GC}, π) for global function computation.*

Proof: Let f be a global function and let (γ^{GC}, π) be an interpreted system for global function computation such that f and $\mathcal{N}(\gamma^{GC})$ satisfy the condition in Theorem 2.3. Let r be a run in the system $\mathbf{I}(Pg^{GC}, \gamma^{GC}, \pi)$.

We first show that at some point in r, some agent knows $f(N_r)$. Suppose not. Let r' be the unique run of the full-information protocol starting with the same initial global state as r. We show by induction on k that there is a time m_k such that, at time (r, m_k), all the agents in $\mathcal{A}(r)$ have at least as much information about the network as they do at the beginning of round k in r'. That is, for all agents i in $\mathcal{A}(r)$, the set of networks i considers possible at time m_k in r (i.e., the set of all networks $N_{r''}$ for r'' run in $\mathbf{I}(Pg^{GC}, \gamma^{GC}, \pi)$ such that there exists a situation (r'', m'', i'') with $r''_{i''}(m'') = r_i(m_k)$) is a subset of the set of networks i considers possible at the beginning of round k in r' (i.e., if m'_k is the time in r' when round k begins, the set of networks $N_{r''}$ for r'' run of the full-information protocol such that there exists a situation (r'', m'', i'') with $r''_{i''}(m'') = r'_i(m'_k)$).

The base case is immediate: we can take $m_1 = 0$ since, by assumption, agents in r and r' start with the same initial states. For the inductive step, suppose that i learns some new information from j in round k of r'. That means j knew this information at the beginning of round k in r' so, by the induction hypothesis, j must have known this information by time m_k in r. Thus, there is a time $m'_k \leq m_k$ such that j first learns this information in run r (where we take $m'_k = 0$ if $k = 1$). It follows from the semantics of Pg^{GC} that j sends this information to i at time m'_k in r. Since we have assumed that communication is reliable, i learns it by some time m''_k. Since i has only finitely many neighbors and there are only finitely many pieces of information about the network, there must be a time in r by which i learns all the information that it learns by the beginning of round $k+1$ in r'. And since there are only finitely many agents in $\mathcal{A}(r)$, there must be a time m_{k+1} by which all the agents in $\mathcal{A}(r)$ learn all the information about the network that they know at the beginning of round $k+1$ in r'.

By Theorem 2.3, there exists a round $k_{\mathcal{N}(\gamma^{GC}),N_r,f}$ such that, running the full-information protocol, for all networks $N' \in \mathcal{N}(\gamma^{GC})$, all $i' \in V(N')$, and all $i \in V(N_r)$, we have that $f(N_r) = f(N')$ if $(N_r, i) \sim_{k_{\mathcal{N}(\gamma^{GC}),N_r,f}} (N', i')$. Suppose that i is an agent in N_r, r' is a run in $\mathbf{I}(Pg^{GC}, \gamma^{GC}, \pi)$, and i' is an agent in $N_{r'}$ such that $r_i(m_{k_{\mathcal{N}(\gamma^{GC}),N_r,f}}) = r'_{i'}(m')$. A straightforward argument now shows that $(N_r, i) \sim_{k_{\mathcal{N}(\gamma^{GC}),N_r,f}} (N_{r'}, i')$. (Formally, we show by induction on k with a subinduction on k' that if $k \leq k_{\mathcal{N}(\gamma^{GC}),N_r,f}$, $k' \leq k$, and j is an agent at distance k' from i in N_r, then there exists an agent j' of distance k' from i' in $N_{r'}$ such that $(N_r, i) \sim_{k-k'} (N_{r'}, i')$, and similarly switching the roles of i, i', N_r, and $N_{r'}$.) It follows that i knows $f(N_r)$ by time $m_{k_{\mathcal{N}(\gamma^{GC}),N_r,f}}$ in r, contradicting the assumption that no agent learns $f(N_r)$.

Suppose that i is the first agent to learn the function value in r, and does so at time m (or one of the first, if there are several agents that learn the function value at time m). We can now use the same argument as above to show that eventually all agents learn the function value. A formal proof proceeds by induction on the distance of agent j from i in N_r; we omit details here. ∎

4 Improving message overhead

While sending only the new information that an agent learns at each step reduces the size of messages, it does not preclude sending unnecessary messages. One way of reducing communication is to have agent i not send information to the agent he names $\mathbf{n}$ if he *knows* that $\mathbf{n}$ already *knows* the information. Since agent i is acting based on what he knows, this is a *knowledge-based (kb) program*. We now formalize this notion.

4.1 Knowledge-based programs with shared names

Consider a language with a modal operator $K_{\mathbf{n}}$ for each name $\mathbf{n} \in \mathbf{N}$. When interpreted relative to agent i, $K_{\mathbf{n}}\varphi$ is read as "the agent i names $\mathbf{n}$ knows φ". A knowledge-based program Pg_{kb} has the form

$$\begin{array}{l}\textbf{if } t_1 \wedge k_1 \textbf{ do } \mathsf{act}_1 \\ \textbf{if } t_2 \wedge k_2 \textbf{ do } \mathsf{act}_2 \\ \quad \ldots\end{array}$$

where t_j and act_j are as for standard programs, and k_j are knowledge tests (possibly involving belief and counterfactual tests, as we will see later in the section).

Let $cont(new\_info)$ be a primitive proposition that characterizes the content of the message new_info. For example, suppose that N is a unidirectional ring, and new_info says that i's left neighbor has input value v_1. Then $cont(new\_info)$ is true at all points where i's left neighbor has input value v_1. (Note that $cont(new\_info)$ is a proposition whose truth is relative to an agent.) Thus, it seems that the following kb program should solve the global function computation problem, while decreasing the number of messages:

$$\begin{array}{l}\textbf{if } some\_new\_info \textbf{ then}\\ \quad \textbf{for each } nonempty\ subset \text{ A } of\ agents \textbf{ do}\\ \quad \textbf{if } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg K_I K_{\mathbf{n}}(cont(new\_info))\} \textbf{ then } send_A(new\_info); receive.\end{array} \tag{1}$$

There are, however, some subtleties involved giving semantics to this program; we consider these in the next section. In the process, we will see that there are number of ways that the message complexity of the program can be further improved.

4.2 Semantics of kb programs with shared names

We can use the machinery that we have developed to give semantics to formulas such as $K_{\mathbf{n}}\varphi$. The statement $K_{\mathbf{n}}\varphi$ holds with respect to a situation (r, m, i) and an interpreted system $\mathcal{I}$ precisely when the agent $j = \mu(r(m), i, \mathbf{n})$ i names $\mathbf{n}$ knows φ when in local state $r_j(m)$, i.e., when φ holds in all situations (r', m', j') in $\mathcal{I}$ agent j cannot distinguish from (r, m, j). We can then define

$(\mathcal{I}, r, m, i) \models K_{\mathbf{n}}\varphi$ iff, for all j, j' and points (r', m') such that $\mu(r(m), i, \mathbf{n}) = j$ and $r_j(m) = r'_{j'}(m')$, we have $(\mathcal{I}, r', m', j') \models \varphi$.

As observed by GH, once we allow relative names, we must be careful about scoping. For example, suppose that, in an oriented ring, i's left neighbor is j and j's left neighbor is k. What does a formula such as $K_I K_L(\mathit{left\_input} = 3)$ mean when it is interpreted relative to agent i? Does it mean that i knows that j knows that k's input is 3, or does it mean that i knows that j knows that j's input is 3? That is, do we interpret the "left" in *left_input* relative to i or relative to i's left neighbor j? Similarly, to which agent does the second L in $K_I K_L K_L\varphi$ refer? That, of course, depends on the application. Using a first-order logic of naming, as in [Grove 1995], allows us to distinguish the two interpretations readily. In a propositional logic, we cannot do this. In the propositional logic, GH assumed *innermost scoping*, so that the *left* in *left_input* and the second L in $K_I K_L K_L\varphi$ are interpreted relative to the "current" agent considered when they are evaluated (which is j). For the purpose of this paper, in a formula such as $K_I K_{\mathbf{n}}$ $cont(new\_info)$, we want to interpret $cont(new\_info)$ relative to "I", the agent i that sends the message, not with respect to the agent j that is the interpretation of $\mathbf{n}$. To capture this, we add limited quantification over names to the language. In particular, we allow formulas of the form $\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge K_{\mathbf{n}}(\mathbf{n}'\text{'s}\varphi))$, which is interpreted as "there exists a name $\mathbf{n}'$ such that the agent I names $\mathbf{n}$ gives name $\mathbf{n}'$ to the agent that currently has name I and $\mathbf{n}$ knows that φ interpreted relative to $\mathbf{n}'$ holds". Thus, to emphasize the scoping, instead of writing $K_I K_{\mathbf{n}} cont(new\_info)$, we write $K_I(\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge K_{\mathbf{n}}(\mathbf{n}'\text{'s} cont(new\_info))))$.

We can now give semantics to kb programs. We can associate with a kb program Pg_{kb} and an extended interpreted system $\mathcal{I} = (\mathcal{R}, \pi, \mu)$ a protocol for agent i denoted $(\mathsf{Pg}_{kb})_i^{\mathcal{I}}$. Intuitively, we evaluate the standard tests in Pg_{kb} according to π and μ and evaluate the knowledge tests according to $\mathcal{I}$. Formally, for each local state ℓ of agent i, we define $(\mathsf{Pg}_{kb})_i^{\mathcal{I}}(\ell)$ to consist of all actions act_j such that the test $t_j \wedge k_j$ holds with respect to a tuple (r, m, i') in $\mathcal{I}$ such that $r_{i'}(m) = \ell$ (recall that protocols can be nondeterministic); if there is no point in $\mathcal{I}$ where some agent has local state ℓ, then $(\mathsf{Pg}_{kb})_i^{\mathcal{I}}(\ell)$ performs the null action (which leaves the state unchanged).

A joint protocol P is said to *implement* Pg_{kb} in interpreted context (γ, π) if, by interpreting Pg_{kb} with respect to $\mathbf{I}(P, \gamma, \pi)$, we get back protocol P; i.e., if, for each agent i, we have $P_i = (\mathsf{Pg}_{kb})_i^{\mathcal{I}(P,\gamma,\pi)}$. Here we seem to be implicitly assuming that all agents run the same kb program. This is certainly true for the programs we give for global function computation, and actually does not result in any loss of generality. For example, if names are commonly known, the actions performed by agents can depend on tests of the form "if your name is $\mathbf{n}$ then ...". Similarly, if we have a system where some agents are senders and others are receivers, the roles of agents can be encoded in their local states, and tests in the program can ensure that all agents act appropriately, despite using the same program.

In certain cases we are interested in joint protocols P that satisfy a condition slightly weaker than implementation, first defined by Halpern and Moses [2004] (HM from now on). Joint protocols P and P' are *equivalent in context* γ, denoted $P \approx_\gamma P'$, if $P_i(\ell) = P'_i(\ell)$ for every local state $\ell = r_i(m)$ with $r \in \mathbf{R}(P, \gamma)$. We remark that if $P \approx_\gamma P'$, then it easily follows that $\mathbf{R}(P, \gamma) = \mathbf{R}(P', \gamma)$: we simply show by induction on m that every prefix of a run in $\mathbf{R}(P, \gamma)$ is a prefix of a run in $\mathbf{R}(P', \gamma)$, and vice versa. P *de facto implements* Pg_{kb} in context γ if $P \approx_\gamma \mathsf{Pg}_{kb}^{\mathcal{I}(P,\gamma,\pi)}$. Arguably, de facto implementation suffices for most purposes, since all we care about are the runs generated by the protocol. We do not care about the behavior of the protocol on local states that never arise when we run the protocol.

The kb program Pg_{kb} solves the global function computation problem for f in the interpreted context (γ^{GC}, π) if, for all protocols P that de facto implement Pg_{kb} in γ^{GC} and all runs r in $\mathcal{R}(P, \gamma)$, eventually all agents in $\mathcal{A}(r)$ know the value $f(N_r)$.

We can now show that the kb program (1) solves the global function computation problem for all functions f and interpreted contexts (γ^{GC}, π) for global function computation such that f and $\mathcal{N}(\gamma^{GC})$ satisfy the condition in Theorem 2.3. Rather than proving this result, we focus on further improving the message complexity of the kb program, and give a formal analysis of correctness only for the improved program.

4.3 Avoiding redundant communication with counterfactual tests

We can further reduce message complexity by not sending information not only if the recipient of the message already knows the information, but also if he will *eventually* know the information. It seems relatively straightforward to capture this: we simply add a $\Diamond$ operator to the kb program (1 to get

$$
\begin{array}{l}
\textbf{if } \mathit{some\_new\_info} \textbf{ then} \\
\quad \textbf{for each } \mathit{nonempty\ subset}\ \mathrm{A}\ \mathit{of\ agents} \textbf{ do} \\
\quad \textbf{if } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg K_I \Diamond (\exists \mathbf{n}' (\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge K_{\mathbf{n}}(\mathbf{n}'\text{'s}\,\mathit{cont}(\mathit{new\_info}))))\} \\
\qquad \textbf{then } \mathit{send}_A(\mathit{new\_info}); \mathit{receive}.
\end{array}
$$

Unfortunately, this modification will not work: as observed by HM, once we add the $\Diamond$ operator, the resulting program has no implementation in the context γ^{GC}. For suppose there exists a protocol P that implements it, and let $\mathcal{I} = \mathcal{I}(P, \gamma^{GC}, \pi)$, that is, by interpreting the above program w.r.t. $\mathcal{I}$, we get back the protocol P. Does i (the agent represented by I) send $\mathit{new\_info}$ to $\mathbf{n}$ in $\mathcal{I}$? If i sends its new information to $\mathbf{n}$ at time m in a run r of $\mathcal{I}$, then, as communication is reliable, eventually $\mathbf{n}$ will know i's new information and i knows that this is the case, i.e., $(\mathcal{I}, r, m, i) \models K_I \Diamond (\exists \mathbf{n}' (\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge K_{\mathbf{n}}(\mathbf{n}'\text{'s}\,\mathit{cont}(\mathit{new\_info}))))$. As P implements the above kb program and $\mathcal{I} = \mathcal{I}(P, \gamma^{GC}, \pi)$, it follows that i does not send its new information to $\mathbf{n}$. On the other hand, if no one sends $\mathit{new\_info}$ to $\mathbf{n}$, then

$\mathbf{n}$ will not know it, and i should send it. Roughly speaking, i should send the information iff i does not send the information.

HM suggest the use of counterfactuals to deal with this problem. As we said in the introduction, a counterfactual has the form $\varphi > \psi$, which is read as "if φ were the case then ψ". As is standard in the philosophy literature (see, for example, [Lewis 1973; Stalnaker 1968]), to give semantics to counterfactual statements, we assume that there is a notion of *closeness* defined on situations. This allows us to consider the situations closest to a given situation that have certain properties. For example, if in a situation (r, m, i) agent i sends its new information to neighbor $\mathbf{n}$, we would expect that the closest situations (r', m, i) to (r, m, i) where i does *not* send its new information to $\mathbf{n}$ are such that, in r', all agents use the same protocol in r' as in r, excpet that, at time m in r', i sends its new information to all agents to which it sends its new information at the point (r, m) with the exception of $\mathbf{n}$. The counterfactual formula $\varphi > \psi$ is taken to be true if, in the closest situations to the current situation where φ is true, ψ is also true.

Once we have counterfactuals, we must consider systems with runs that are not runs of the program. These are runs where, for example, counter to fact, the agent does not send a message (although the program says it should). Following HM, we can make these executions less likely relative to those generated by running the program by associating to each run a *rank*; the higher the rank, the less likely the run. We then require that the runs of the program be the only ones of minimal rank. Once we work with a system that includes runs other than those generated by the program, agents may no longer *know* that, for example, when the program says they should send a message to their neighbor, they actually do so (since there could be an run in the system not generated by the program, in which at some point the agent has the same local state as in a run of the program, but it does not send a message). Agents do know, however, that they send the message to their neighbor in all runs of minimal rank, that is, in all the runs consistent with the program. By associating a rank with each run, we can talk about formulas φ that hold at all situations in runs of minimal rank among those an agent i cannot distinguish from the current situation. If φ holds at all points in runs of minimal rank that i considers possible then we say that i *believes* φ (although i may not *know* φ. We write $B_{\mathbf{n}}\varphi$ to denote that the agent named $\mathbf{n}$ believes φ, although this is perhaps better read as "the agent named $\mathbf{n}$ knows that φ is (almost certainly) true". We provide the formal semantics of belief and counterfactuals, which is somewhat technical, in Appendix A; we hope that the intuitions we have provided will suffice for understanding what follows.

Using counterfactuals, we can modify the program to say that agent i should send the information only if i does not believe "if I do not send the information, then $\mathbf{n}$ will eventually learn it anyway". To capture this, we use the proposition $do(send_{\mathbf{n}}(new\_info))$, which is true if i is about to send new_info to $\mathbf{n}$. If there are only finitely many possible values of f, say $v_1, \ldots, v_k$, then the formula $B_{\mathbf{n}}(f = v_1) \vee \ldots \vee B_{\mathbf{n}}(f = v_k)$ captures the fact that the agent with name $\mathbf{n}$ knows the value of f. However, in general, we want to allow an unbounded number of function values. For example, if agents have distinct numerical ids, we are trying to elect as leader the agent with the highest id, and there is no bound on the size of the network, then the set of possible values of f is unbounded. We deal with this problem by allowing limited quantification over values. In particular, we use formulas of the form $\exists v B_{\mathbf{n}}(f = v)$, which intuitively say that the agent with name $\mathbf{n}$ knows the value of f. Let Pg^{GC}_{cb} denote the following modification of Pg^{GC}:

$$\begin{array}{l} \textbf{if } \mathit{some\_new\_info} \textbf{ then} \\ \quad \textbf{for each } \mathit{nonempty\ subset}\ \mathrm{A}\ \mathit{of\ agents}\ \textbf{do} \\ \quad \textbf{if } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg B_I[\neg do(send_{\mathbf{n}}(new\_info)) > \\ \qquad\qquad \Diamond(\exists \mathbf{n}'(Calls(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}'\text{'s} cont(new\_info))) \vee \exists v B_{\mathbf{n}}(f = v))]\} \\ \quad \textbf{then } send_A(new\_info); receive. \end{array}$$

In this program, the agent i representing I sends $\mathbf{n}$ the new information if i does not believe that $\mathbf{n}$ will eventually learn the new information or the function value in any case. As shown in Appendix B, this improved program still solves the global function computation problem whenever possible.

Theorem 4.1: *If f and $\mathcal{N}(\gamma^{GC})$ satisfy the condition in Theorem 2.3, then Pg^{GC}_{cb} solves the global function computation problem for f in all interpreted contexts (γ^{GC}, π) for global function computation.*

5 Case study: leader election

In this section we focus on leader election. If we take the function f to describe a method for computing a leader, and require that all agents eventually know who is chosen as leader, this problem becomes an instance of global function computation. We assume that agents have distinct identifiers (which is the context in which leader election has been studied in the literature). It follows from Corollary 2.4 that leader election is solvable in this context; the only question is what the complexity is. Although leader election is only one instance of the global function computation problem, it is of particular interest, since it has been studied so intensively in the literature. We show that a number of well-known protocols for leader election in the literature essentially implement the program Pg^{GC}_{cb}. In particular, we consider a protocol combining ideas of Lann [1977] and Chang and Roberts [1979] (LCR from now on) presented by Lynch [1997], which works in unidirectional rings, and Peterson's [1982] protocol P1 for unidirectional rings and P2 for bidirectional rings. We briefly sketch the LCR protocol and Peterson's protocols P1 and P2, closely following Lynch's [1997] treatment.

The LCR protocol works in unidirectional rings, and does not assume a bound on their size. Each agent starts by sending its id along the ring; whenever it receives a value, if the value is larger than the maximum value seen so far, then the agent forwards it; if not, it does nothing, except when it receives its own id. If this id is M, the agent then sends the message "the agent with id M is the leader" to its neighbor. Each agent who receives such a message forwards it until it reaches the agent with id M again. The LCR protocol is correct because it ensures that the maximum id travels along the ring and is forwarded by each agent until some agent receives its own id back. That agent then knows that its id is larger than that of any other agent, and thus becomes the leader.

Peterson's protocol P2 for bidirectional rings operates in phases. In each phase, agents are designated as either *active* or *passive*. Intuitively, the active agents are those still competing in the election. Once an agent becomes passive, it remains passive, but continues to forward messages. Initially all agents are active. In each phase, an active agent compares its id with the ids of the closest active agent to its right and the closest active agent to its left. If its id is the largest of the three, it continues to be active; otherwise, it becomes passive. Just as with the LCR protocol, when an agent receives back its own id, it declares itself leader. Then if its id is M, it sends the message "the agent with id M is the leader", which is forwarded around the ring until everyone knows who the leader is.

Peterson shows that, at each phase, the number of active agents is at most half that of the previous phase, and always includes the agent with the largest id. It follows that, eventually, the only active agent is the one with the largest id. Peterson's protocol terminates when the agent that has the maximum id discovers that it has the maximum id by receiving its own id back. The message complexity of Peterson's protocol is thus $O(n \log n)$, where n is the number of agents.

Peterson's protocol P1 for unidirectional rings is similar. Again, passive agents forward all messages they receive, at each round at most half of the agents remain active, and the agent with the largest value becomes leader. There are, however, a number of differences. Agents now have "temporary" ids as well as their own ids. It is perhaps better to think of an agent's id as being active if it has an "active temporary id". (In the bidirectional case, we can identify the temporary id with the actual id, so an agent is active iff its id is active.) We take a temporary id to be active at phase $p+1$ if it is larger than the temporary ids that precede or follow it in phase p. But since messages can only be sent in one direction, the way to discover this is for an active agent to forward its temporary id to the following two active agents. An active agent can then tell if the preceding active agent's temporary id was greater than the following and preceding active temporary id's. If so, it remains active, and takes as its temporary id what was the temporary id of the preceding active agent. Otherwise, the agent becomes passive. It is not hard to check that an agent is active in the bidirectional protocol iff its id is active in the unidirectional protocol (i.e., iff its id is the temporary id of an active agent in the unidirectional protocol). When an agent receives its original value, then it declares itself leader and sends a message describing the result of the election around the ring.

We remark that although they all work for rings, the LCR protocol is quite different from P1 and P2. In the LCR protocol, agents forward their values along their unique outgoing link. Eventually, the agent with the maximum input receives its own value and realizes that it has the maximum value. In P1 and P2, agents are either *active* or *passive*; in each round, the number of active agents is reduced, and eventually only the agent with the maximum value remains active.

Despite their differences, LCR, P1, and P2 all essentially implement Pg_{cb}^{GC}. There are two reasons we write "essentially" here. The first, rather trivial reason is that, when agents send information, they do not send all the information they learn (even if the agent they are sending it to will never learn this information). For example, in the LCR protocol, if agent i learns that its left neighbor has value v and this is the largest value that it has seen, it passes along v without passing along the fact that its left neighbor has this value. We can easily deal with this by modifying the protocols so that all the agents send new_info rather than whatever message they were supposed to send. However, this modification does not suffice. The reason is that the modified protocols send some "unnecessary" messages. This is easiest to see in the case of LCR. Suppose that j is the processor with highest id. When j receives the message with its id back and sends it around the ring again (this is essentially the message saying that j is the leader), in a full-information protocol, j's second message will include the id j' of the processor just before j. Thus, when j' receives j's second message, it will not need to forward it to j. If LCR$'$ is the modification of LCR where each process sends new_info rather than the maximum id seen so far, and the last message in LCR is not sent, then we can show that LCR$'$ indeed de facto implements Pg_{cb}^{GC}. The modifications to P2 that are needed to get a protocol P2$'$ that de facto implements Pg_{cb}^{GC} are somewhat more complicated. Each processor i running P2$'$ acts as it does in P2 (modulo sending new_info) until the point where it first gets a complete picture of who is in the ring (and hence who the leader is). What happens next depends on whether i is the first to find out who the leader is or not and whether i is active or not. We leave details to the Appendix C.

Theorem 5.1: *The following all hold:*

(a) Given parameter d*, the optimal flooding protocol [Lynch 1997] de facto implements* Pg_{cb}^{GC} *in contexts where (i) all networks have diameter at most* d *and (ii) all agents have distinct identifiers.*

(b) LCR′ de facto implements Pg_{cb}^{GC} *in all contexts where (i) all networks are unidirectional rings and (ii) agents have distinct identifiers.*

(c) There exists a protocol P1′ that agrees with P1 up to the last phase (except that it sends new_info*) and implements* Pg_{cb}^{GC} *in all contexts where (i) all networks are unidirectional rings and (ii) agents have distinct identifiers.*

(d) There exists a protocol P2′ that agrees with P2 up to the last phase (except that it sends new_info*) and de facto implements* Pg_{cb}^{GC} *in all contexts where (i) all networks are bidirectional rings and (ii) agents have distinct identifiers.*

Theorem 5.1 brings out the underlying commonality of all these protocols. Moreover, it emphasizes the connection between counterfactual reasoning and message optimality. Finally, it shows that reasoning at the kb level can be a useful tool for improving the message complexity of protocols. For example, although P2′ has the same order of magnitude message complexity as P2 ($O(n \log n)$), it typically sends $O(n)$ fewer messages. While this improvement comes at the price of possibly longer messages, it does suggest that this approach can result in nontrivial improvements. Moreover, it suggests that starting with a high-level kb program and then trying to implement it using a standard program can be a useful design methodology. Indeed, our hope is that we will be able to synthesize standard programs by starting with high-level kb specifications, synthesizing a kb program that satisfies the specification, and then instantiating the kb program as a standard program. We have some preliminary results along these lines that give us confidence in the general approach [Bickford, Constable, Halpern, and Petride 2005]; we hope that further work will lend further credence to this approach.

A Counterfactual belief-based programs with names

The standard approach to giving semantics to counterfactuals [Lewis 1973; Stalnaker 1968] is that $\varphi > \psi$ is true at a point (r, m) if ψ is true at all the points "closest to" or "most like" (r, m) where φ is true. For example, suppose that we have a wet match and we make a statement such as "if the match were dry then it would light". Using $\Rightarrow$ this statement is trivially true, since the antecedent is false. However, with $>$, we must consider the worlds most like the actual world where the match is in fact dry and decide whether it would light in those worlds. If we think the match is defective for some reason, then even if it were dry, it would not light.

To capture this intuition in the context of systems, we extend HM's approach so as to deal with names. We just briefly review the relevant details here; we encourage the reader to consult [Halpern and Moses 2004] for more details and intuition. Define an *order assignment* for an extended interpreted system $\mathcal{I} = (\mathcal{R}, \pi, \mu)$ to be a function $\lessdot$ that associates with every situation (r, m, i) a partial order relation $\lessdot_{(r,m,i)}$ over situations. The partial orders must satisfy the constraint that (r, m, i) is a minimal element of $\lessdot_{(r,m,i)}$, so that there is no situation (r', m', i') such that $(r', m', i') \lessdot_{(r,m,i)} (r, m, i)$. Intuitively, $(r_1, m_1, i_1) \lessdot_{(r,m,i)} (r_2, m_2, i_2)$ if (r_1, m_1, i_1) is "closer" to the true situation (r, m, i) than

(r_2, m_2, i_2). A *counterfactual system* is a pair of the form $\mathcal{J} = (\mathcal{I}, \lessdot)$, where $\mathcal{I}$ is an extended interpreted system and $\lessdot$ is an order assignment for the situations in $\mathcal{I}$.

Given a counterfactual system $\mathcal{J} = (\mathcal{I}, \lessdot)$, a set A of situations, and a situation (r, m, i), we define the situations in A that are closest to (r, m, i), denoted $\mathtt{closest}(A, r, m, i)$, by taking

$$\begin{array}{l}\mathtt{closest}(A, r, m, i) = \\ \quad \{(r', m', i') \in A : \text{ there is no situation } (r'', m'', i'') \in A \\ \quad \text{such that } (r'', m'', i'') \lessdot_{(r,m,i)} (r', m', i')\}.\end{array}$$

A counterfactual formula is assigned meaning with respect to a counterfactual system $\mathcal{J}$ by interpreting all formulas not involving $>$ with respect to $\mathcal{I}$ using the earlier definitions, and defining

$$(\mathcal{J}, r, m, i) \models \varphi > \psi \text{ iff for all } (r', m', i') \in \mathtt{closest}([\![\varphi]\!]_{\mathcal{J}}, r, m, i),\ (\mathcal{J}, r', m', i') \models \psi,$$

where $[\![\varphi]\!]_{\mathcal{J}} = \{(r, m, i) : (\mathcal{J}, r, m, i) \models \varphi\}$; that is, $[\![\varphi]\!]_{\mathcal{J}}$ consists of all situations in $\mathcal{J}$ satisfying φ.

All earlier analyses of (epistemic) properties of a protocol P in a context γ used the runs in $\mathbf{R}(P, \gamma)$, that is, the runs consistent with P in context γ. However, counterfactual reasoning involves events that occur on runs that are not consistent with P (for example, we may need to counterfactually consider the run where a certain message is not sent, although P may say that it should be sent). To support such reasoning, we need to consider runs not in $\mathbf{R}(P, \gamma)$. The runs that must be added can, in general, depend on the type of counterfactual statements allowed in the logical language. Thus, for example, if we allow formulas of the form $do(i, \mathsf{act}) > \psi$ for process i and action act, then we must allow, at every point of the system, a possible future in which i's next action is act. Following [Halpern and Moses 2004], we do reasoning with respect to the system $\mathcal{R}^+(\gamma)$ consisting of *all* runs compatible with γ, that is, all runs consistent with some protocol P' in context γ.

We want to define an order assignment in the system $\mathcal{R}^+(\gamma)$ that ensures that the counterfactual tests in Pg_{cb}^{GC}, which have an antecedent $\neg do(send_{\mathbf{n}}(msg)$, get interpreted appropriately. HM defined a way of doing so for counterfactual tests whose antecedent has the form $do(i, \mathsf{act})$. We modify their construction here. Given a context γ, situation (r, m, i) in $\mathcal{R}^+(\gamma)$, action act, and a deterministic protocol P,[4] we define the closest set of situations to (r, m, i) where i does *not* perform action $send_{\mathbf{n}}(msg)$, $\mathtt{close}(\overline{send_{\mathbf{n}}(msg)}, P, \gamma, r, m, i)$, as $\{(r', m, i') :$ (a) $r' \in \mathcal{R}^+(\gamma)$, (b) $r'(m') = r(m')$ for all $m' \leq m$, (c) if agent i performs some action $send_A(msg')$ according to P in local state $r_i(m)$ and $\mathbf{n} \notin A$ or $msg' \neq msg$, or if i does not perform action $send_A(msg')$ for any set A of agents and message msg', then $r' = r$ and $i = i'$, (d) if agent i performs $send_A(msg)$ according to P in local state $r_i(m)$ and $\mathbf{n} \in A$, then i performs $send_{A-\{\mathbf{n}\}}(msg)$ in local state $r_i(m)$ in run r', and follows P in all other local states in run r', (e) all agents other than i' follow P at all points of $r'\}$. That is, $\mathtt{close}(\overline{send_{\mathbf{n}}(msg)}, P, \gamma, r, m, i)$ is $\{r, m, i\}$ if i does not send msg to $\mathbf{n}$ at the local state $r_i(m)$; otherwise $\mathtt{close}(\overline{send_{\mathbf{n}}(msg)}, P, \gamma, r, m, i)$ is the set consisting of situations (r', m, i') such that r' is identical to r up to time m and all the agents act according to P at later times, except that at the local state $r'_{i'}(m) = r_i(m)$ in r', agent i' who is indistinguishable from i does not send msg to $\mathbf{n}$, but does send it to all other agents to which it sent msg in $r_i(m)$.

Define an *order generator* o to be a function that associates with every protocol P an order assignment $\lessdot^P = o(P)$ on the situations of $\mathcal{R}^+(\gamma)$. We are interested in order generators that prefer runs in

[4] We restrict in this paper to deterministic protocols. We can generalize this definition to randomized protocols in a straightforward way, but we do not need this generalization for the purposes of this paper.

which agents follow their protocols as closely as possible. An order generator o for γ *respects protocols* if, for every (deterministic) protocol P, interpreted context $\zeta = (\gamma, \pi)$ for global computation, situation (r, m, i) in $\mathbf{R}(P, \gamma)$, and action act, $\mathtt{closest}([\![\neg send_A(msg)]\!]_{\mathbf{I}(P,\zeta)}, r, m, i)$ is a nonempty subset of $\mathtt{close}(\overline{send_{\mathbf{n}}(msg)}, P, \gamma, r, m, i)$ that includes (r, m, i) if $(r, m, i) \in \mathtt{close}(\overline{send_A(msg)}, P, \gamma, r, m, i)$. Perhaps the most obvious order generator that respects protocols just sets $\mathtt{closest}([\![\neg send_{\mathbf{n}}(msg)]\!]_{\mathbf{I}(P,\zeta)}, r, m, i) = \mathtt{close}(\overline{send_{\mathbf{n}}(msg)}, P, \gamma, r, m, i)$, although our results hold if $=$ is replaced by $\subseteq$.

Reasoning in terms of the large set of runs $\mathcal{R}^+(\gamma)$ as opposed to $\mathbf{R}(P, \gamma)$ leads to agents not knowing properties of P. For example, even if, according to P, some agent i always performs action act when in local state l_i, in $\mathcal{R}^+(\gamma)$ there are bound to be runs r and times m such that $r_i(m) = l_i$, but i does not perform action act at the point (r, m). Thus, when we evaluate knowledge with respect to $\mathcal{R}^+(\gamma)$, i no longer knows that, according to P, he performs act in state l_i. Following HM, we deal with this by adding extra information to the models that allows us to capture the agents' beliefs. Although the agents will not *know* they are running protocol P, they will *believe* that they are. We do this by associating with each run $r \in \mathcal{R}^+(\gamma)$ a *rank* $\kappa(r)$, which is either a natural number or ∞, such that $\min_{r \in \mathcal{R}^+(\gamma)} \kappa(r) = 0$. Intuitively, the rank of a run defines the likelihood of the run. Runs of rank 0 are most likely; runs of rank 1 are somewhat less likely, those of rank 2 are even more unlikely, and so on. Very roughly speaking, if $\epsilon > 0$ is small, we can think of the runs of rank k as having probability $O(\epsilon^k)$. We can use ranks to define a notion of belief (cf. [Friedman and Halpern 1997]).

Intuitively, of all the points considered possible by a given agent in a situation (r, m, i), the ones believed to have occurred are the ones appearing in runs of minimal rank. More formally, for a point (r, m) define

$$\mathsf{min}_i^{\kappa}(r, m) \;=\; \min\{\kappa(r') \mid r' \in \mathcal{R}^+(\gamma) \text{ and } r'_{i'}(m') = r_i(m) \text{ for some } m' \geq 0 \text{ and } i' \in \mathcal{A}(r')\}.$$

Thus, $\mathsf{min}_i^{\kappa}(r, m)$ is the minimal κ-rank of runs r' in which $r_i(m)$ appears as a local state at the point (r', m).

A *counterfactual belief system* (or just *cb* system for short) is a triple of the form $\mathcal{J} = (\mathcal{I}, \lessdot, \kappa)$, where $(\mathcal{I}, \lessdot)$ is a counterfactual system, and κ is a ranking function on the runs of $\mathcal{I}$. In cb systems we can define a notion of belief. We add the modal operator $B_{\mathbf{n}}$ to the language for each $\mathbf{n} \in \mathbf{N}$, and define

$$\begin{aligned}(\mathcal{I}, \lessdot, \kappa, r, m, i) \models B_{\mathbf{n}}\varphi \text{ iff, for all } j, j' \text{ and points } (r', m') \text{ such that } \mu(r, m, i, \mathbf{n}) = j, \\ r_j(m) = r'_{j'}(m'), \text{ and } \kappa(r') = \mathsf{min}_j^{\kappa}(r, m), \text{ we have } (\mathcal{I}, r', m', j') \models \varphi.\end{aligned}$$

The following lemma illustrates a key feature of the definition of belief. What distinguishes knowledge from belief is that knowledge satisfies the *knowledge axiom*: $K_i\varphi \Rightarrow \varphi$ is valid. While $B_i\varphi \Rightarrow \varphi$ is not valid, it is true in runs of rank 0.

Lemma A.1: [Halpern and Moses 2004] *Suppose that $\mathcal{J} = (\mathcal{R}, \pi, \mu, \lessdot, \kappa)$ is a cb system, $r \in \mathcal{R}$, and $\kappa(r) = 0$. Then for every formula φ and all times m, we have $(\mathcal{J}, r, m, i) \models B_I\varphi \Rightarrow \varphi$.*

By analogy with order generators, we want a uniform way of associating with each protocol P a ranking function. Intuitively, we want to do this in a way that lets us recover P. We say that a ranking function κ is *P-compatible* (for γ) if $\kappa(r) = 0$ if and only if $r \in \mathbf{R}(P, \gamma)$. A *ranking generator* for a context γ is a function σ ascribing to every protocol P a ranking $\sigma(P)$ on the runs of $\mathcal{R}^+(\gamma)$. A ranking generator σ is *deviation compatible* if $\sigma(P)$ is P-compatible for every protocol P. An obvious example of a deviation-compatible ranking generator is the *characteristic* ranking generator σ_ξ, where $\sigma_\xi(P)$ is the ranking that assigns rank 0 to every run in $\mathbf{R}(P,\gamma)$ and rank 1 to all other runs. This captures the assumption that runs of P are likely and all other runs are unlikely, without attempting to distinguish among them. Another deviation-compatible ranking generator is σ^*, where $\sigma^*(P)$ is the ranking that assigns to a run r the total number of times that agents deviate from P in r. Obviously, $\sigma^*(P)$ assigns r the rank 0 exactly if $r \in \mathbf{R}(P,\gamma)$, as desired. Intuitively, σ^* captures the assumption that not only are deviations unlikely, but they are independent.

It remains to give semantics to the formulas $\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}'\text{'s}\varphi))$ and $\exists v B_{\mathbf{n}}(f = v)$. Recall that we want $\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}'\text{'s}\varphi))$ to be true at a situation (r, m, i) if there exists a name $\mathbf{n}'$ such that the agent j that agent i names $\mathbf{n}$ calls i $\mathbf{n}'$, and j knows that φ interpreted relative to $\mathbf{n}'$ (i.e., i) holds. More formally,

> $(\mathcal{I}, \lessdot, \kappa, r, m, i) \models \exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}'\text{'s}\varphi))$ iff, for all j, j' and points (r', m') such that $\mu(r(m), i, \mathbf{n}) = j$, $r_j(m) = r'_{j'}(m')$, and $\kappa(r') = \mathsf{min}_j^\kappa(r, m)$, we have $(\mathcal{I}, r', m', i) \models \varphi$.

Note that the semantics for $\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}'\text{'s}\varphi))$ is almost the same as that for $B_{\mathbf{n}}\varphi$. The difference is that we evaluate φ at (r', m') with respect to i (the interpretation of I at the situation (r, m, i)), not j'. We could give semantics to a much richer logic that allows arbitrary quantification over names, and give separate semantics to formulas of the form $\mathit{Calls}(\mathbf{n}, I, \mathbf{n}')$ and $\mathbf{n}'\text{'s}\varphi$, but what we have done suffices for our intended application.

The semantics of $\exists v B_{\mathbf{n}}(f = v)$ is straightforward. Recall that the value of f in run r is $f(N_r)$. We can then take $\exists v B_{\mathbf{n}}(f = v)$ to be true at a point (r, m) according so some agent i if all runs $\mathbf{n}$ believes possible are associated with the same function value:

> $(\mathcal{I}, \lessdot, \kappa, r, m, i) \models \exists v B_{\mathbf{n}}(f = v)$ iff, for all j, j' and points (r', m') such that $\mu(r(m), i, \mathbf{n}) = j$, $r_j(m) = r'_{j'}(m')$, and $\kappa(r') = \mathsf{min}_j^\kappa(r, m)$, we have $f(N_r) = f(N_{r'})$.

With all these definitions in hand, we can define the semantics of counterfactual belief-based programs such as Pg_{cb}^{GC}. A *counterfactual belief-based program* (or *cbb program*, for short) Pg_{cb} is similar to a kb program, except that the knowledge modalities $K_{\mathbf{n}}$ are replaced by the belief modalities $B_{\mathbf{n}}$. We allow counterfactuals in belief tests but, for simplicity, do not allow counterfactuals in the standard tests.

As with kb programs, we are interested in when a protocol P *implements* a cbb program Pg_{cb}. Again, the idea is that the protocol should act according to the high-level program, when the tests are evaluated in the cb system corresponding to P. To make this precise, given a cb system $\mathcal{J} = (\mathcal{I}, \lessdot, \kappa)$, an agent i, and a cbb program Pg_{cb}, let $(\mathsf{Pg}_{cb})_i^{\mathcal{J}}$ denote the protocol derived from Pg_{cb} by using $\mathcal{J}$ to evaluate the belief tests. That is, a test in Pg_{cb} such as $B_{\mathbf{n}}\varphi$ holds at a situation (r, m, i) in $\mathcal{J}$ if φ holds at all situations (r', m', j') in $\mathcal{J}$ such that $\mu(r(m), i, \mathbf{n}) = j$, $r'_{j'}(m') = r_j(m)$, and $\kappa(r') = \mathsf{min}_j^\kappa(r, m)$. Define a *cb context* to be a tuple (γ, π, o, σ), where (γ, π) is an interpreted context with naming function μ_γ (for simplicity, we use μ_γ to refer to the naming function in context γ), o is an order generator for $\mathcal{R}^+(\gamma)$ that respects protocols, and σ is a deviation-compatible ranking generator for γ. A cb system $\mathcal{J} = (\mathcal{I}, \lessdot, \kappa)$ *represents* the cbb program Pg_{cb} in cb context (γ, π, o, σ) if (a) $\mathcal{I} = (\mathcal{R}^+(\gamma), \pi, \mu_\gamma)$, (b) $\lessdot = o(\mathsf{Pg}_{cb}^{\mathcal{J}})$, and (c) $\kappa = \sigma(\mathsf{Pg}_{cb}^{\mathcal{J}})$. A protocol P *implements* Pg_{cb} in cb context $\chi = (\gamma, \pi, o, \sigma)$ if $P = \mathsf{Pg}_{cb}^{(\mathcal{I}, o(P), \sigma(P))}$. Protocol P *de facto implements* Pg_{cb} in χ if $P \approx_\gamma \mathsf{Pg}_{cb}^{(\mathcal{I}, o(P), \sigma(P))}$.

B Proof of correctness for Pg_{cb}^{GC}

Theorem 4.1: *If f and $\mathcal{N}(\gamma^{GC})$ satisfy the condition in Theorem 2.3, then Pg_{cb}^{GC} solves the global function computation problem for f in all interpreted contexts (γ^{GC}, π) for global function computation.*

Proof: Let f and $\mathcal{N}$ be such that the condition in Theorem 2.3 is satisfied. Suppose that o is an order generator that respects protocols, σ is a deviation-compatible ranking generator, γ^{GC} is a context for global computation such that in all initial states the network encoded in the environment state is in $\mathcal{N}$, χ^{GC} is the cb context $(\gamma^{GC}, \pi, o, \sigma)$, P is a protocol that de facto implements Pg_{cb}^{GC} in χ^{GC}, $\mathcal{J} = (\mathcal{R}^+(\gamma), \pi, \mu_\gamma, o(P), \sigma(P))$, and $r \in \mathbf{R}(P, \gamma^{GC})$. We prove that at some point in run r all agents in N_r know $f(N_r)$.

We proceed much as in the proof of Theorem 3.1; we just highlight the differences here. Again, we first show that some agent in r learns $f(N_r)$. Suppose not. Let r' be the unique run of the full-information protocol in a synchronous context starting with the same initial global state as r. Again, we show by induction on k that there is a time m_k such that, at the point (r, m_k), all the agents in $\mathcal{A}(r)$ have at least as much information about the network as they do at the beginning of round k in r'. The base case is immediate, as before. For the inductive step, suppose that i learns some information about the network from j during round k. Again, there must exist a time $m'_k \leq m$ where j first learns this information in run r. It follows that $(\mathcal{J}, r, m'_k, j) \models \mathit{some\_new\_info}$.

Suppose that j names i $\mathbf{n}$ in r; that is $\mu_\gamma(r(m_k), j, \mathbf{n}) = i$. Now either (a) j believes at time m'_k that, if he does not perform a $\mathit{send}_{\mathsf{A}}(\mathit{new\_info})$ action with $\mathbf{n} \in A$, i will eventually learn its new information or the function value anyway, or (b) j does not believe this. In case (b), it follows that

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models \;& \neg B_I[\neg \mathit{do}(\mathit{send}_{\mathbf{n}}(\mathit{new\_info})) > \Diamond((\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}')\wedge \\ & B_{\mathbf{n}}(\mathbf{n}' \text{ 's } \mathit{cont}(\mathit{new\_info}))) \vee \exists v B_{\mathbf{n}}(f = v))].\end{aligned}$$

Since P implements Pg_{cb}^{GC} in χ^{GC}, in case (b), j sends i $\mathit{new\_info}$ at time m'_k, so there is some round m''_k by which i learns this information. On the other hand, in case (a), it must be the case that

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models \;& B_I[\neg \mathit{do}(\mathit{send}_{\mathbf{n}}(\mathit{new\_info})) > \Diamond(\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}')\wedge \\ & B_{\mathbf{n}}(\mathbf{n}' \text{ 's } \mathit{cont}(\mathit{new\_info}))) \vee \exists v B_{\mathbf{n}}(f = v))].\end{aligned}$$

Since σ is deviation compatible by assumption, and r is a run of P, it follows that $\kappa(r) = 0$. Thus by Lemma A.1,

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models \;& \neg \mathit{do}(\mathit{send}_{\mathbf{n}}(\mathit{new\_info})) > \Diamond(\exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}')\wedge \\ & B_{\mathbf{n}}(\mathbf{n}' \text{ 's } \mathit{cont}(\mathit{new\_info}))) \vee \exists v B_{\mathbf{n}}(f = v)).\end{aligned}$$

Since P implements Pg_{cb}^{GC} in χ^{GC}, in case (a), j does not send $\mathit{new\_info}$ to i in round m'_k. Thus, $(\mathcal{J}, r, m'_k, j) \models \neg \mathit{do}(\mathit{send}_{\mathbf{n}}(\mathit{new\_info}))$. It follows that

$$(\mathcal{J}, r, m'_k, j) \models \exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}' \text{ 's } \mathit{cont}(\mathit{new\_info}))) \vee \exists v B_{\mathbf{n}}(f = v)).$$

Since, by assumption, no one learns the function value in r, we have that

$$(\mathcal{J}, r, m'_k, j) \models \exists \mathbf{n}'(\mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \wedge B_{\mathbf{n}}(\mathbf{n}' \text{ 's } \mathit{cont}(\mathit{new\_info}))).$$

Thus, it follows that i must eventually learn j's information in this case too.

It now follows, just as in the proof of Theorem 3.1, that some agent learns $f(N_r)$ in r, and that eventually all agents learn it. We omit details here. ∎

```
status := nonleader; maxid := id; val_R :=⊥; done := 0
send_L(id)
do until done = 1
   receive
   if RQ ≠ ⊥ then
      val_R := dequeue(RQ)
      if (val_R = id) then
         status := leader; send_L("id is the leader"); done := 1
      else if (val_R > maxid) then
         maxid := val_R; send_L(maxid)
      else if (val_R is a leader message) then
         send_L(val_R); done := 1
```

Figure 3: The LCR protocol.

```
do until (id ∈ val_R) ∧ (sent leader message ∨ maxid = id_L)
   receive
   if some_new_info then
      if ((id ∉ val_R ∧ max(val_R) > maxid) ∨ (id ∈ val_R) then send_L(new_info)
```

Figure 4: The LCR$'$ protocol.

C Proof of Theorem 5.1

In this section we prove Theorem 5.1, which says that LCR$'$, P1$'$, and P2$'$ de facto implement Pg_{cb}^{GC}. We start by sketching the proof for LCR$'$, and then provide a detailed proof for P2$'$. The proof for P1$'$ is similar and is omitted here.

C.1 The argument for LCR$'$

The pseudocode for LCR and LCR$'$ is given in Figures 3 and 4 respectively. In the code for LCR, we use id to denote the agent's initial id. We assume that each agent has one queue, denoted RQ, which holds messages received from the right. The placing of messages in the queue is controlled by the channel, not the agent. We use $RQ = \perp$ to denote that the right queue is empty. We write $val_R := dequeue(RQ)$ to denote the operation of removing the top message from the right queue and assigning it to the variable val_R. If $RQ = \perp$ when a $dequeue$ operation is performed, then the agent waits until it is nonempty. Each agent has a local variable $status$ that is initially set to $nonleader$ and is changed to $leader$ only by the agent with the maximum id in the ring when it discovers it is the leader. We take $done$ to be a binary variable that is initialized to 0 and changed to 1 after the maximum id has been computed. Agents keep track of the maximum id seen so far in the variable $maxid$. We call a message of the form "M is the leader" a *leader message*. Note that in our version of LCR, after the leader finds out that it is the leader, it informs all the other agents of this fact. This is not the case for the original LCR protocol. We include it here for compatibility with our global function computation protocol. (Similar remarks hold for P2.)

In the code for LCR$'$, val_R encodes all the new information that the sender sends (and thus is not just a single id). Let $max(val_R)$ be the maximum id encoded in val_R. Since an agent sends all the new information it has, there is no need for special messages of the form "M is the leader". The leader can be computed from val_R if the message has gone around the ring, which will be the case if $id \in val_R$. Moreover, if $id \in val_R$, an agent can also compute whether the leader is its left neighbor, and whether it has earlier essentially sent an "M is the leader message" (more precisely, an agent can tell if it has earlier been in a state where $id \in val_R$ and it sent a message). We take the test $id_L = maxid$ to be true if an agent knows that the leader is its left neighbor (which means that a necessary condition for $id_L = maxid$ to be true is that $id \in val_R$); we take *sent leader message* to be true if $id \in val_R$ and the agent earlier sent a message when $i \in val_R$ was true. Notice that in LCR$'$ we do not explicitly set val_R; val_R can be computed from the agent's state, by looking at the new information received.

The basic idea of the proof is simple: we must show that Pg_{cb}^{GC} and LCR$'$ act the same at all points in a system that represent LCR$'$. That means showing that an agent sends a message iff it believes that, without the message, its neighbor will not eventually learn the information that it has or the function value. Since LCR$'$ solves the leader election problem, when processors do not send a message, they believe (correctly) that their neighbor will indeed learn the function value. So consider a situation where a processor i sends a message according to LCR$'$. That means that either it has gotten a message val_R such that $val_R > maxid$ or it has gotten a leader message. If it does not forward a leader message, then it is clear that all the processors between i and the leader (of which there must be at least one) will not learn who the leader is, because no further messages will be sent. If i has received a message with $val_R > maxid$, then consider $maxid$ is in fact the largest id. Then it is easy to see that i will never receive any further messages, and no processor will ever find out who the leader is. Since this ring is consistent with i's information, i does not believe that, if it does not forward the message, i's left neighbor will learn the information or learn who the leader is. Thus, according to Pg_{cb}^{GC}, i should forward the message. We omit the formal details of the proof here, since we do the proof for P2$'$ (which is harder) in detail.

C.2 The argument for P2$'$

We start by describing P2. Since P2 works in bidirectional rings, rather than just having one queue, as in LCR, in P2, each agent has two queues, denoted LQ and RQ, which hold messages received from the left and right, respectively. While an agent is active, it processes a message from RQ, then LQ, then RQ, and so on. The status of an agent, i.e., whether it is active, passive or the leader, is indicated by the variable $status$. Initially, $status$ is $active$. Finally, we take wl to be a binary variable that indicates whether the agent is waiting to receive a message from its left. When an active agent receives val_R, it compares val_R to its id. If $val_R = id$ (which can happen only if i is active) then, as in the LCR protocol, i declares itself to be the leader (by setting $status$ to $leader$), and it sends out a message to this effect. If i is active and $val_R > id$, then i becomes passive; if $val_R < id$, then i remains active and sends its id to the right. Finally, if i is passive, then i forwards val_R to the left. The situation is symmetric if i receives val_L. The pseudocode for P2 is given in Figure 5.

To understand in more detail how P2 and P2$'$ work, it is helpful to characterize the order in which agents following P2 send and process messages. Since P2 and P2$'$ are identical up to the point that an agent knows the leader, the characterization will apply equally well to P2$'$. We can get a complete characterization despite the fact that we do not assume synchrony, nor that messages are received in FIFO order. As usual, we use $(a_1, \ldots, a_k)^*$ to denote 0 or more repetitions of a sequence of actions

$status := active;\ val_L := \perp;\ val_R := \perp;\ done := 0;\ wl = 0$
$send_L(id);$
do until $done = 1$
  if $(RQ \neq \perp) \wedge (wl = 0)$ **then**
    $val_R := dequeue(RQ)$
    $wl := 1$
    if $(val_R = id)$ **then** $status := leader;\ send_R($"*id is the leader*"$);\ done := 1$
    if $status = active \wedge val_R > id$ **then** $status := passive$
    if $status = active \wedge val_R < id$ **then** $send_R(id)$
    if $status = passive$ **then** $send_L(id);$ **if** $(val_R$ is a leader message$)$ **then** $done := 1$
  if $(LQ \neq \perp) \wedge (wl = 1)$ **then**
    $val_L := dequeue(LQ)$
    $wl := 0$
    if $(val_L = id)$ **then** $status := leader;\ send_L($"*id is the leader*"$);\ done := 1$
    if $status = active \wedge val_L > id$ **then** $status := passive$
    if $status = active \wedge val_L < id$ **then** $send_L(id)$
    if $status = passive$ **then** $send_R(id);$ **if** $(val_L$ is a leader message$)$ **then** $done := 1$

Figure 5: Peterson's protocol P2.

$a_1, \ldots, a_k$. We denote the action of sending left (resp. right) as SL (resp. SR), and the action of processing from the left (resp. right) as PL (resp. PR).

Lemma C.1: *For all runs r of P2, times m, and agents i in N_r*

(a) if i is active at time m, then i's sequence of actions in the time interval $[0, m)$ is a prefix of the sequence (SL, PR, SR, PL);*

(b) if i is passive at time m, i does not yet know which agent has the maximum id, and i became passive at time $m' \leq m$ after processing a message from the right (resp., left), then i's history in the time interval $[m', m]$ is a prefix of the sequence (PL, SR, PR, SL) (resp., (PR, SL, PL, SR)*).*

Proof: We proceed by induction on the time m. The result is trivially true if $m = 0$, since no actions are performed in the interval $[0, 0]$. Suppose the result is true for time m; we show it for time $m + 1$. If i is active at time $m + 1$, then the result is immediate from the description of P2 (since it is immediate that, as long as i is active, it cycles through the sequence SL, PR, SR, PL). So suppose that i is passive at time $m + 1$. It is clear from the description of P2 that, while i is passive, PL is immediately followed by SR and PR is immediately followed by SL. Thus, it suffices to show that (i) if i was active when it performed its last action, and this action was PR, then i's next action is PL; (ii) if i was active when it performed its last action, and this action was PL, then i's next action is PR; (iii) if i was passive when it performed its last action, and this action was SR, then i's next action is PR; and (iv) if i was passive when it performed its last action, and this action was SL, then i's next action is PL. The proofs of (i)–(iv) are all essentially the same, so we just do (i) here.

Suppose that i's last action before time $m+1$ was PR, and then i became passive. It is clear from the description of P2 that i's next action is either PR or PL. Suppose, by way of contradiction, that i performs PR at time $m+1$. It follows from the induction hypothesis that there must exist some k such that i performed SR k times and PR $k+2$ times in the interval $[0, m+1]$. But then the agent R_i to i's right performed SL at least $k+2$ times and PL at most k in the interval $[0, m]$. This contradicts the induction hypothesis. ∎

Intuitively, P2 and P2$'$ act the same as long as agents do not know who the leader is. In P2$'$, they will know who the leader is once they know all the agents on the ring. To make this latter notion precise, define the sets $I_L(i, r, m)$ and $I_R(i, r, m)$ of agents as follows: $I_R(i, r, 0) = I_L(i, r, 0) = \{i\}$. If, at time $m+1$, i processes a message from its right, and this message was sent by R_i at time m', then

$$I_R(i,r,m+1) = I_R(i,r,m) \cup I_R(R_i,r,m') \text{ and } I_L(i,r,m+1) = I_L(i,r,m) \cup I_L(R_i,r,m') - \{R_i\}.$$

If, at time $m+1$, i processes a message from its left, and this message was sent by L_i at time m', then

$$I_L(i,r,m+1) = I_L(i,r,m) \cup I_L(L_i,r,m') \text{ and } I_R(i,r,m+1) = I_R(i,r,m) \cup I_R(L_i,r,m') - \{L_i\}.$$

Finally, if i does not process a message at time $m+1$, then

$$I_R(i,r,m+1) = I_R(i,r,m) \text{ and } I_L(i,r,m+1) = I_L(i,r,m).$$

$I_R(i, r, m)$ and $I_L(i, r, m)$ characterize the set of agents to i's right and left, respectively, that i knows about at the point (r, m). $I_L(i, r, m)$ and $I_R(i, r, m)$ are always intervals for agents running a full-information protocol (we prove this formally below). Thus, agent i has *heard from everybody in the ring*, denoted *heard_from_all*, if $I_L(i, r, m) \cup I_R(i, r, m)$ contains all agents in the ring. More formally, $(\mathcal{J}, r, m, i) \models$ *heard_from_all* if $I_L(i, r, m) \cup I_R(i, r, m)$ consists of all the agents in the network N encoded in the environment state in (r, m). Note that *heard_from_all* may hold relative to agent i without i knowing it; i may consider it possible that there are agents between the rightmost agent in $I_R(i, r, m)$ and the leftmost agent in $I_L(i, r, m)$. We define the primitive proposition *has_all_info* to be true at at the point (r, m) relative to i if $I_L(i, r, m) \cap I_R(i, r, m) - \{i\} \neq \emptyset$. It is not difficult to show that *has_all_info* is equivalent to K_I(*heard_from_all*); thus, we say that *i knows it has all the information* if *has_all_info* holds relative to i.

The pseudocode for P2$'$ while agents do not know that they have all the information is given in Figure 6. (We describe what agents do when they know all the information at the end of this section.) Note that the pseudocode does not describe what happens if an agent is active and $val_R \geq id$. Intuitively, at this point, the agent becomes passive, but with P2$'$ there is no action that changes an agent's status; rather, the status is inferred from the messages that have been received. (This is similar to the reason that the LCR$'$ protocol had so many fewer steps than the LCR protocol.) Since agents running P2 perform the same actions under essentially the same conditions as agents running P2$'$ up to the point that an agent knows that it has all the information, Lemma C.1 also applies to all runs r of P2$'$, times m, and agents i in N_r such that i did not know that it had all the information at time $m-1$ in r.

We now prove a number of properties of $I_L(i, r, m)$ and $I_R(i, r, m)$ that will be useful in our analysis of P2$'$.

Lemma C.2: *For all runs r of P2$'$ and times m the following hold:*

$send_L(new\_info)$;
do until has_all_info
  if $(RQ \neq \perp) \wedge (wl = 0)$ **then**
    if $status = active \wedge val_R < id$ **then** $send_R(new\_info)$
    if $status = passive$ **then** $send_L(new\_info)$;
  if $(LQ \neq \perp) \wedge (wl = 1)$ **then**
    if $status = active \wedge val_L < id$ **then** $send_L(new\_info)$
    if $status = passive$ **then** $send_R(new\_info)$;

Figure 6: The initial part of protocol P2$'$, run while agents do not know that they have all the information.

(a) *$I_R(i,r,m)$ is an interval of agents starting with i and going to the right of i, and $I_L(i,r,m)$ is an interval of agents starting with i and going to the left of i.*

(b) *If, at time m, i processes a message from the right sent by R_i at time m', and R_i did not know that it had all the information at time m', then*

  (i) *$I_R(R_i,r,m') \supset I_R(i,r,m-1) - \{i\}$, $I_R(i,r,m) \supset I_R(i,r,m-1)$, and $I_R(i,r,m) = \{i\} \cup I_R(R_i,r,m')$; and*

  (ii) *$I_L(i,r,m) = I_L(i,r,m-1)$.*

(c) *If, at time m, i processes a message from the left sent by L_i at time m', and L_i did not know that it had all the information at time m', then*

  (i) *$I_L(L_i,r,m') \supset I_L(i,r,m-1) - \{i\}$, $I_L(i,r,m) \supset I_L(i,r,m-1)$, and $I_L(i,r,m) = \{i\} \cup I_L(L_i,r,m')$; and*

  (ii) *$I_R(i,r,m) = I_R(i,r,m-1)$.*

(d) *If i processed a message from the right in the interval $[0,m]$, and R_i did not know that it had all the information when it last sent a message to i, then*

$$\max_{\{m' \leq m : val_R(i,r,m') \neq \perp\}} val_R(i,r,m')$$

is the maximum id of the agents in $I_R(i,r,m) - \{i\}$, where $val_R(i,r,m')$ is the value of agent i's variable val_R at the point (r,m'); if i processed a message from the left in the interval $[0,m]$, then

$$\max_{\{m' \leq m : val_L(i,r,m') \neq \perp\}} val_L(i,r,m')$$

is the maximum id in $I_L(i,r,m) - \{i\}$.

(e) *i is active at time m if and only if i has the largest id in $I_L(i,r,m) \cup I_R(i,r,m)$.*

Proof: We prove all parts of the lemma simultaneously by induction on m. The result is immediate if $m = 0$, since i is active at time 0, i does not process a message at time 0, and $I_L(i,r,0) = I_R(i,r,0) = \{i\}$. Suppose that parts (a)–(e) hold for all times $m' < m$. We show that they also hold at time m. They clearly hold if i does not process a message at time m, since in that case $I_L(i,r,m) = I_L(i,r,m-1)$

and $I_R(i,r,m) = I_R(i,r,m-1)$. So suppose that i processes a message msg from its right at time m, and msg was sent by R_i at time m'. (The proof is similar if i receives from the left, and is left to the reader.) If msg is the first message received by i from the right, then it follows from Lemma C.1 that i has sent no messages to the right, and R_i has sent only one message to i. Thus, $I_R(i,r,m-1) = \{i\}$. Parts (a)–(e) now follow easily from the induction hypothesis.

So suppose that msg is not the first message that i has received from R_i. Part (a) is immediate from the induction hypothesis. To prove part (b), let m_1 be the last time prior to m' that R_i sent a message, say msg', to its left. It easily follows from Lemma C.1 (which, as we observed, also applies to P2$'$ while agents do not know that they have all the information) that there are times m_2 and m_3, both in the interval (m_1, m'), such that i received msg' at time m_2 and R_i processed a message from its right at m_3; moreover, i did not process any messages from the right between time m_2 and m. By the induction hypothesis, $I_R(i,r,m_2) = \{i\} \cup I_R(R_i,r,m_1)$, $I_L(i,r,m_2) = I_L(i,r,m_2-1)$, and $I_R(R_i,r,m_3+1) \supset I_R(R_i,r,m_1)$. Since $m_3+1 \le m'$, it follows that $I_R(R_i,r,m') \supset I_R(R_i,r,m_1)$. Since i does not process any messages from its right between time m_2 and m, by definition, $I_R(i,r,m-1) = I_R(i,r,m_2)$. It follows that $I_R(R_i,r,m') \supset I_R(i,r,m-1)$ and that

$$\begin{aligned} I_R(i,r,m) &= I_R(i,r,m-1) \cup I_R(R_i,r,m') = \{i\} \cup I_R(R_i,r,m_1) \cup I_R(R_i,r,m') \\ &= \{i\} \cup I_R(R_i,r,m') \supset \{i\} \cup I_R(R_i,r,m_1) = I_R(i,r,m-1). \end{aligned}$$

This proves part (i) of (b) for time m. For part (ii), by definition, $I_L(i,r,m) = I_L(i,r,m-1) \cup I_L(R_i,r,m') - \{R_i\}$. By the induction hypothesis, it easily follows that $I_L(R_i,r,m') - \{R_i\} \subseteq I_L(i,r,m') \subseteq I_L(i,r,m-1)$. Thus, $I_L(i,r,m) = I_L(i,r,m-1)$.

Part (c) is immediate, since i does not process a message from the left at time m.

For the first half of part (d), there are two cases to consider. If R_i was active at the point (r,m'), then the result is immediate from part (e) of the inductive hypothesis. Otherwise, by the inductive hypothesis, $val_R = val_R(i,r,m) = val_R(R_i,r,m')$. By the inductive hypothesis, val_R is greater than or equal to the maximum id in $I_R(R_i,r,m') - \{R_i\}$. Since the first value of val_R must be R_i's id, it follows that

$$\max_{\{m' \le m : val_R(i,r,m') \ne \bot\}} val_R(i,r,m')$$

is greater than or equal to the maximum id in $I_R(i,r,m) - \{i\} = I_R(R_i,r,m')$. Since $val_R(i,r,m')$ must be an id in $I_R(i,r,m)$, we are done. The second half of part (d) is immediate from the induction hypothesis, since $I_L(i,r,m) = I_L(i,r,m-1)$ by part (b), and $val_L(i,r,m) = val(i,r,m-1)$.

Finally, part (e) is immediate from the induction hypothesis if i is passive at time $m-1$. So suppose that i is active at time $m-1$. By the induction hypothesis, i's id is the largest in $I_L(i,r,m-1) \cup I_R(i,r,m-1)$. If i is active at time m then, by the description of P2$'$, i's id must be greater than $val_R(i,r,m)$. Applying part (d) of the induction hypothesis and the fact that i's id is at least as large as all those in $I_R(i,r,m-1)$, it follows that i's id is at least as large as $\max_{\{m' \le m : val_R(i,r,m') \ne \bot\}} val_R(i,r,m')$. By part (d), at time m, i's id is at least as large all those in $I_R(i,r,m)$. Since $I_L(i,r,m) = I_L(i,r,m-1)$, it follows that i's id is the maximum id in $I_R(i,r,m) \cup I_L(i,r,m)$. Conversely, if i's id is the maximum id in $I_R(i,r,m) \cup I_L(i,r,m)$, then by part (d) at time m, i's id must be greater than $val_R(i,r,m)$, and hence by the description of P2$'$, i is active at (r,m). ∎

It is not difficult to see that P2$'$ ensures that, for all agents i, $I_L(i,r,m) \cup I_R(i,r,m)$ increases with time m. Thus, eventually at least one agent must know it has all the information. (Recall that we have not yet given the pseudocode for P2$'$ for the case that an agent knows it has all the information.)

Corollary C.3: *In all runs r consistent with P2$'$, eventually at least one agent knows that it has all the information, i.e., there exist an agent i and time m such that $I_L(i,r,m) \cap I_R(i,r,m) - \{i\} \neq \emptyset$.*

We say that message msg received by i at time m *originated with j at time m'* if j is the active agent who first sent msg, and msg was sent by j at time m'. More formally, we define origination by induction on the time m that msg was received. If msg is received by i from the right, then msg originated with R_i at the time that R_i sent it if R_i was not passive when it sent msg; otherwise, if msg was received at some time $m'' < m$ by R_i, then the message msg received by i at m originated with the same agent and at the same time as the message msg received by R_i at m''. The definition is analogous if msg is received by i from the left.

Let $[i..j]_R$ denote the agents to i's right starting at i and going to j; similarly, let $[i..j]_L$ denote the agents to i's left starting at i and going to j.

Lemma C.4: *For all runs r of P2$'$ and agents i, j in r,*

(a) if at time m agent i processes a message msg from the right that originated with j at m', msg is the pth message j sent left, and no agent in $[i..j]_R$ knows that it has all the information when it sends msg, then msg is the pth message that i processes from the right, and $I_R(i,r,m) = I_R(j,r,m') \cup [i..j]_R$.

(b) if at time m agent i processes a message msg from the left that originated with j at m', and msg is the pth message j sent right, and no agent in $[i..j]_L$ knows that it has all the information when it sends msg, then msg is the pth message that i processes from the left and $I_L(i,r,m) = I_L(j,r,m') \cup [i..j]_L$.

Proof: We do the proof for case (a); the proof of (b) is similar and left to the reader. The proof proceeds by induction on the number of agents in $[i..j]_R$. Since $i \neq j$, there are at least two agents in $[i..j]_R$. If there are exactly two, then $j = R_i$. Since the only messages that i processes from the right are those sent by j, it is immediate that msg is the pth message i processed from the right. Moreover, by definition $I_R(i,r,m) = I_R(j,r,m') \cup \{i\} = I_R(j,r,m') \cup [i..j]_R$.

Now suppose that (a) holds for all pairs of agents i', j' such that $[i'..j']_R$ consists of $d \geq 2$ agents and $[i..j]_R$ consists of $d+1$ agents. Let m_{R_i} be the time R_i sends the message msg to i. Since $[i..j]_R$ consists of at least 3 agents, it cannot be the case that $R_i = j$. Thus, R_i was passive when it received the message msg. Let m'_{R_i} be the time R_i processed msg. Since $[R_i..j]_R$ has d agents, by the induction hypothesis, it follows that msg was the pth message that R_i processed from the right. By Lemma C.1, prior to m'_{R_i}, R_i sent exactly $p-1$ messages to the left. Moreover, since R_i must process $p-1$ messages from the left before processing its pth message from the right, it follows from Lemma C.1 that i must have processed all the $p-1$ messages R_i sent to it before R_i processed msg. It now easily follows that msg is the pth message processed by i from the right. By the induction hypothesis, $I_R(R_i,r,m'_{R_i}) = I_R(j,r,m') \cup [R_i..j]_R$. Thus, $I_R(i,r,m) = I_R(R_i,r,m'_{R_i}) \cup \{i\} = I_R(j,r,m') \cup [i..j]_R$. ∎

By Lemma C.1, we can think of P2$'$ as proceeding in phases while agents do not know all the information. For $p = 1,2,3,\ldots$, we say that in run r, phase $2p-1$ begins for agent i when i sends left for the pth time and phase $2p$ begins for agent i when i sends right for the pth time; phase p for agent i ends when phase $p+1$ begins.

The following lemma provides some constraints on what agents know about which agents are active and passive.

Lemma C.5: *For all runs r of P2$'$, times m, and agents i, if $m > 0$, the last message that i processed before time m was the pth message, and no agent knows all the information at time $m-1$, then*

(a) *if $j_1, \ldots, j_k$ are the active agents at time m in $I_R(i,r,m)$, listed in order of closeness to i on the right (so that j_1 is the closest active process to i's right with $j_1 = i$ if i is active, and j_k is the farthest) then (i) $id_{j_1} > \ldots > id_{j_k}$, (ii) if $j_1 \neq i$, then j_l will be passive after having processed its $(p-l+1)$st message, for $l = 2, \ldots, k$, provided that j_l processes its $(p-l+1)$st message before knowing all the information; (iii) if $j_1 = i$, then j_l will be passive after after having processed its $(p-l+3)$rd message, for $l = 2, \ldots, k$, provided that j_l processes its $(p-l+3)$rd message before knowing all the information; and (iv) the last message that i processed from the right originated with j_1.*

(b) *if $h_1, \ldots, h_{k'}$ are the active agents at time m in $I_L(i,r,m)$ listed in order of closeness to i on the left, then (i) $id_{h_1} > \ldots > id_{h_{k'}}$, (ii) if $h_1 \neq i$, then h_l will be passive after having processed its $(p-l+1)$st message, for $l = 2, \ldots, k'$, provided that h_l processes its $(p-l+1)$st message before knowing all the information; (iii) if $h_1 = i$, then h_l will be passive after having processed its $(p-l+3)$rd message, provided that it processes its $(p-l+3)$rd message before knowing all the information; and (iv) the last message that i processed from the left originated with h_1.*

Proof: We proceed by induction on m. The lemma is trivially true if $m = 1$, since $I_L(i,r,1) = I_R(i,r,1) = \{i\}$. If $m > 1$, then the result is trivially true if i does not process a message at time $m-1$ (since $I_L(i,r,m) = I_L(i,r,m{-}1)$ unless i processes a message from the left at time $m{-}1$, and similarly for $I_R(i,r,m)$; and even if some agents in $I_L(i,r,m) \cup I_R(i,r,m)$ may become passive between time $m-1$ and time m, the result continues to hold). So suppose that i processes a message from the left at time $m-1$. Since $I_R(i,r,m) = I_R(i,r,m-1)$, it is immediate from the induction hypothesis that part (a) continues to hold. For part (b), by Lemma C.4, we have that $I_L(i,r,m) = I_L(j,r,m') \cup [i..j]_L$, where the message that i processed from the left at time $m-1$ originated with j at time m'. By the definition of origination, all agents in $[i..j]_L - \{i,j\}$ must be passive at time $m-1$. Thus, the result follows immediately from the induction hypothesis applied to j and time m', together with the following observations:

- If j originated the message at time m', then it follows easily from Lemma C.1 that it is the pth message sent by j. Moreover, either $I_L(j,r,m') = \{j\}$ or $I_L(j,r,m') = I_L(j,r,m'')$, where $m''-1$ is the time that j processed its $(p-2)$nd message (since this is the last message that j processed from the left prior to time m').
- If i is active at time m, then $id_i > id_j$, and the $(p+1)$st message that j processes will originate from i (if j does not know all the information before processing the message) and will cause j to become passive.

The argument is similar if i processes a message from the right at time $m-1$. ∎

We say that agent i *can be the first to learn all the information in network N* if there is a run r of P2$'$ such that $N_r = N$ and, in run r, i knows all the information at some time m and no agent knows all the information at the point $(r, m-1)$. Our goal is to prove that there can be at most two agents that can be first to learn all the information in a network N.[5] To prove this result, we first show that, although we are considering asynchronous systems, what agents know depends only on how many messages they have processed.

Lemma C.6: *If $N_r = N_{r'} = N$, no agent knows all the information at the point (r, m) or the point (r', m'), and agent i has processed exactly k messages at both the points (r, m) and (r', m'), then $I_L(i, r, m) = I_L(i, r', m')$ and $I_R(i, r, m) = I_R(i, r', m')$. Moreover, the kth message that i processed in run r originated with j iff the kth message that i processed in run r' originated with j.*

Proof: We proceed by a straightforward induction on $m + m'$. Clearly the result is true if $m = m' = 1$. If i does not process a message at the point $(r, m-1)$, then $I_L(i, r, m) \cup I_R(i, r, m) = I_L(i, r, m-1) \cup I_R(i, r, m-1)$, and the result is immediate from the induction hypothesis; similarly, the result follows if i does not process a message at the point $(r', m'-1)$. Thus, we can assume that i processes a message at both $(r, m-1)$ and $(r', m'-1)$. Moreover, it follows from Lemma C.1 that i either processes from the left at both $(r, m-1)$ and $(r', m'-1)$ or processes from the right at both of these points. Assume without loss of generality that i processes from the left. Then, using the induction hypothesis, we have that $I_R(i, r, m) = I_R(i, r, m-1) = I_R(i, r', m'-1) = I_R(i, r', m')$. Moreover, $I_L(i, r, m) = I_L(L_i, r, m_1) \cup \{i\}$, where m_1 is the time L_i sent the message that i processes at time $m-1$ in r; $I_L(i, r', m') = I_L(L_i, r', m'_1) \cup \{i\}$, where m'_1 is the time that L_i sent the message that i processes at time $m'-1$ in r'. It follows from Lemma C.1 that we must have $k = 2k'$, L_i has sent k' messages left at the points (r, m_1) and (r', m'_1), and has processed $k-1$ messages at both of these points. By the induction hypothesis, $I_L(L_i, r, m_1) = I_L(L_i, r', m'_1)$. The desired result follows immediately. ∎

Lemma C.7: *There are at most two agents that can be first to learn all the information in network N. If an agent that can be first to learn all the information is active when it learns all the information, then it must be the agent with the highest id.*

Proof: Suppose, by way of contradiction, that three agents can be the first to learn all the information, say i_1, i_2, and i_3. Suppose that i^* is the agent in N with the highest id. Suppose that the message that i_h processed which caused it to know all the information was the p_hth message that i_h processed, for $h = 1, 2, 3$. First assume that $i^* \notin \{i_1, i_2, i_3\}$. It easily follows from Lemma C.5 that, for $h = 1, 2, 3$, either the p_hth message or the $(p_h - 1)$st message that i_h processed must have come from i^*. Suppose that for two of i_1, i_2, or i_3, the message that i_h processed from i^* came from the right. Suppose, without loss of generality, that these two agents are i_1 and i_2. Now a simple case analysis shows that either i_1 knows all the information before i_2 in all runs of P2$'$ where $N_r = N$, or i_2 knows all the information before i_1 in all runs where $N_r = N$. For example, suppose that the message that originated with i^* is the p'_hth message that i_h processed, for $h = 1, 2$; note that p'_h is either p_h or $p_h - 1$. (By Lemma C.6, p'_h is same in all runs r such that $N_r = N$.) If $p'_1 > p'_2$ then it follows from Lemma C.1 that $p'_1 \geq p'_2 + 2$, and it is easy to see that i_1 must learn all the information before i_2. Similarly, if $p'_2 > p'_1$, then it is easy to see that i_2 must learn all the information before i_1. Finally, suppose that $p' = p'_1 = p'_2$. Without loss

[5] In all the examples we have constructed, there is in fact only one agent that can be first to learn all the information in network N, although that agent may not be the eventual leader. However, we have not been able to prove that this must be the case.

of generality, assume that going from i^* left on the ring, we reach i_1 before i_2. Then it is easy to see that if $p'_1 = p_1$, so that i_1 knows it has all the information after processing the message from i^*, then i_1 knows it has all the information before i_2 in all runs r with $N_r = N$, while if $p_1 = p'_1 + 1$, then i_1 must learn it after i_2 in all runs (since the p_1th message processed by i_1 must originate with a process farther to the left of i^* than i_2). Thus, it cannot be the case that both i_1 and i_2 can be first to learn the message, a contradiction. A similar contradiction arises if both i_1 and i_2 process i^*'s message from the left.

Thus, it follows that $i^* \in \{i_1, i_2, i_3\}$; without loss of generality, assume that $i^* = i_3$. Again, if both of i_1 and i_2 process i^*'s message from the left, or both process it from the right, then we get a contradiction as above. So suppose without loss of generality that i_1 processes i^*'s message from the left, i_2 processes i^*'s message from the right, and $i^* = i_3$ processes its p_3th message from the left. Again, it is easy to show that if $p_1 \leq p_3$, then in all runs r with $N_r = N$, i_1 knows it has all the information before $i_3 = i^*$; if $p_1 > p_3$, then in all runs r with $N_r = N$, i^* knows it has all the information before i_1. Either way, we have a contradiction. ∎

We can now describe the remainder of protocol P2$'$, after an agent i learns all the information. What happens depends on (a) which agents can be first to learn all the information, and whether i is one of them; (b) whether i is active or passive just after learning all the information, and (c) whether the message that results in i learning all the information is processed from the left or the right. Note that when an agent learns all the information, it can easily determine which agents can be first to learn all the information. Rather than writing the pseudocode for P2$'$, we give just an English description; we do not think that the pseudocode will be more enlightening.

- Suppose that the only agent that can be first to learn all the information is the leader. We now do essentially what is done in Peterson's algorithm. Suppose that the message that resulted in the leader learning all the information was processed from the left (if the message was processed from the right, the rest of the argument remains the same, replacing left by right everywhere), the message originated with agent i, and was the pth message processed by the leader. We claim that after processing the pth message, all agents other than the leader will be passive. If i is the leader, this is almost immediate. If i is not the leader, then it follows from Lemma C.5. The leader then sends its $(p+1)$st message to the left. After an agent processes the leader's $(p+1)$st message, it will then know all the information. We require it to send a message to the left with all the information unless it is the leader's right neighbor. (Of course, once it knows all the information, the leader's right neighbor will realize that the neighbor to the left is the leader and that the leader already knows all the information, so it does not need to forward the information.) After this process is completed, all the agents know all the information.

- Suppose that agent i is the only agent that can know all the information and i is passive when it first knows all the information. Suppose that the message that resulted in i's learning all the information was processed from the left (again, the argument is similar if it was processed from the right), the message originated with agent j, and was the pth message processed by i. It is easy to see that i must have been active just prior to processing the pth message, for otherwise the agent to i's left will learn all the information before i. Moreover, i's pth message must have originated with the leader (since i could not have known about the leader prior to receiving the message, or it would not have been active). Then i sends the message with all the information back to the leader, who forwards the message all the way around the ring up to the agent to i's right, at which point all the agents know all the information.

- Suppose that two passive agents, say i and i', can be first to learn all the information. Again, it is not hard to see that i and i' must have been active just before learning all the information. If i and i' both first learn all the information after processing the pth message, then by Lemma C.5, the pth message of one of them, say i, originated with i^*. Suppose without loss of generality that i and i' received this message from the left. Then i sends a message with all the information to the left, where it is forwarded up to and including i^*; similarly, i' sends a message to the left, which is forwarded up to but not including i. Note that i' will also receive a $(p+1)$st message that originates with i^* from the right. After receiving this message, i' sends a message with all the information to the right up to but not including i^*.

- Suppose that one passive agent, say i, and i^* can be first to learn all the information. If they both learn all the information after receiving their pth message, then i must have been active just before receiving the message, i's message originated with i^*, and i^*'s message either originated with i or with an agent i' such that the pth message received by i' originated with i, and i' becomes passive after receiving this message. Suppose without loss of generality that the pth message was received from the left. Then i sends a message with all the information to the left where it is forwarded up to but not including i^*; similarly, i^* sends a message with all the information to the left, where it is forwarded up to but not including i. A straightforward case analysis shows that it cannot be the case that there exist p and p' with $p \neq p'$ such that i learns all the information after receiving its pth message and i^* learns all the information after receiving the p'th message. For if $p < p'$, then i must learn all the information before i^* in all runs, and if $p' < p$, then i^* must learn all the information before i in all runs.

This completes the description of P2$'$.

Having completed the description of P2$'$, we can finally prove that P2$'$ de facto implements Pg_{cb}^{GC} in contexts where (i) all networks are bidirectional rings and (ii) agents have distinct identifiers. Let $(\gamma^{br,u}, \pi)$ denote the interpreted context for global computation where the initial states are the bidirectional rings with unique identifiers. Suppose that o is an order generator that respects protocols, σ is a deviation-compatible ranking function, and $\mathcal{J} = (\mathcal{R}^+(\gamma^{br,u}), \pi, \mu_{\gamma^{br,u}}, o(P2'), \sigma(P2'))$ is the interpreted system corresponding to P2$'$ in the cb context $\chi^{br,u} = (\gamma^{br,u}, \pi, o, \sigma)$. Proving that P2$'$ de facto implements Pg_{cb}^{GC} in the cb context $\chi^{br,u}$ amounts to showing that $P2'_i(\ell) = \mathsf{Pg}_{cb}^{GC}{}_i^{\mathcal{J}}(\ell)$ for every local state ℓ such that there exists $r \in \mathbf{R}(P2', \gamma^{ur,u})$ and m such that $\ell = r_i(m)$. That is, for all $r \in \mathbf{R}(P2', \gamma^{ur,u})$ and times m, we must show that $P2'_i(r_i(m)) = \mathsf{act}$ iff $(\mathcal{J}, r, m, i) \models \varphi_{\mathsf{act}}$, where φ_{act} is the precondition in Pg_{cb} for action act.

Lemma C.8: *For all runs r of P2$'$ in the context $\gamma^{br,u}$, times m, and agents i in N_r, $P2'_i(r_i(m)) = \mathsf{Pg}_{cb}^{GC}{}_i^{\mathcal{J}}(r_i(m))$.*

Proof: As we observed above, we must show that for all $r \in \mathbf{R}(P2', \gamma^{br,u})$ and times m, we have that $P2'_i(r_i(m)) = \mathsf{act}$ iff $(\mathcal{J}, r, m, i) \models \varphi_{\mathsf{act}}$. So suppose that $P2'_i(r_i(m)) = \mathsf{act}$. The relevant actions act have the form $send_{\mathbf{n}}(new\_info)$, where $\mathbf{n} \in \{L, R\}$. We consider the case that $\mathbf{n} = L$ here; the proof for $\mathbf{n} = R$ is almost identical, and left to the reader. The precondition of $send_L(new\_info)$ is

$$\neg B_I[\neg do(send_L(new\_info)) > \Diamond(\exists \mathbf{n}'(Calls(L, I, \mathbf{n}') \wedge B_L(\mathbf{n}'\text{'s}\, cont(new\_info))) \vee \exists v B_L(f = v))].$$

Since R is the unique name that i's left nieghbor calls i in a ring, we have that $(\mathcal{J}, r, m, i) \models \mathit{Calls}(L, I, R)$. By the definitions in Section A, $(\mathcal{J}, r, m, i) \models \varphi_{send_L(new\_info)}$ if and only if there exists a situation (r', m', i') such that

(a) $r'_{i'}(m') = r_i(m)$,

(b) $\sigma(P2')(r') = \mathsf{min}_i^{\sigma(P2')}(r, m)$, and

(c) $(\mathcal{I}, r', m', i') \models \neg[\neg do(send_L(new\_info)) > \Diamond(\exists \mathbf{n}'(\mathit{Calls}(L, I, \mathbf{n}') \wedge B_L(\mathbf{n}'\text{'s} cont(new\_info))) \vee \exists v B_L(f = v))]$, so there exists a situation $(r'', m'', i'') \in \mathtt{closest}([\![\neg do(send_{\mathbf{n}}\,(new\_info))]\!]_{\mathbf{I}(P2', \chi^{br,u})}, r', m', i')$ such that

$$(\mathcal{J}, r'', m'', i'') \models \Box(\neg B_L(R\text{'s } cont(new\_info)) \wedge \neg\exists v.\ B_L(f = v)).$$

Thus, we must show that there exists a situation (r', m', i') satisfying conditions (a), (b), and (c) above iff $P2'_i(r_i(m)) = send_L(new\_info)$. To prove this, we need to consider the various cases where i sends left.

- Case 1: at (r, m), i is active, does not know it has all the information, and sends its first message at time m. In this case, we can take r' to be a run of P2$'$ on the network $[i]$ (i.e., the network where the only agent is i), $m' = 0$, and $i' = i$, and take (r'', m'', i'') to be an arbitrary situation in $\mathtt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r', m', i')$ such that $|N_{r''}| > 1$. In r'', $L_{i''}$ does not receive a message from i'', so will never process any message. It easily follows that, in r'', $L_{i''}$ does not learn the content (i'')'s initial information, nor does it learn who the leader is.

- Case 2: i is active, does not know all the information, and does not send its first message to the left at time m. In this case, L_i must be passive. Suppose that i is about to send its kth message left at the point (r, m). By Lemma C.1, i must have received $k - 1$ message from L_i, so L_i must have processed $k - 1$ messages from i. Moreover, i considers it possible that L_i has already sent its kth message left, and is waiting to process its kth message from i. Since i does not have all the information at time m, it is easy to see that i must also consider it possible that L_i does not have all the information at time m. Thus, there exists a run r' such that $r_i(m) = r'_i(m)$ and, at the point (r', m), L_i does not have all the information and is waiting to process the kth message from i. Let (r'', m'', i'') be an arbitrary situation in $\mathtt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r', m, i)$. Since i'' does not send left at (r'', m''), $L_{i''}$ will wait forever to process a message from i''. Thus, in r'', $L_{i''}$ never learns the content of (i'')'s kth message, nor does it learn who the leader is.

- Case 3: i is passive at the point (r, m) and does not have all the information. Since i is about to send left and it is passive, i must have last processed a message from its right; without loss of generality, assume that i has processed p messages from its right, and so must have processed $(p - 1)$ messages from its left by time m. It easily follows from Lemma C.1 that $p > 1$. Suppose that the $(p - 1)$st message that i processed from its left originated with k. Since i does not have all the information at time m, k did not have all the information when it sent this message to the right. After receiving its $(p - 1)$st message from the left, i must consider it possible that the ring is sufficiently large that, even after k processes its $(p - 1)$st message from the left, k will still not know all the information. That is, there exists a situation (r', m', i') with $r' \in \mathbf{R}(P2', \gamma^{br,u})$ such that conditions (a) and (b) are satisfied, and if i''s $(p - 1)$st message from the left in r' originated with k', then k' does not have all the information at the point (r', m'), despite have processed its $(p-1)$st message from the left by this point. Let (r'', m'', i'') be an arbitrary situation in $\mathtt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r', m', i')$. Suppose that (i'')'s $(p-1)$st message from the left in r'' originated with k''. At the point (r'', m''), k'' has already processes its $(p-1)$st message from the left and does not have all the information (because this was the case for the agent k' corresponding to k'' in r'). In r'', all processes between i'' and k'' are passive. Thus, regardless of whether k'' is active or passive, in r'', k'' and i'' and all agents between them are deadlocked, because k'' is waiting from a message from the right, which must pass through i'', and i'' is waiting for a message from its left, which must pass through k''. It easily follows that $L_{i''}$ does not learn (i'')'s new information in r'', nor does $L_{i''}$ learn who the leader is.

- Case 4: i has all the information at time m in r. There are a number of subcases to consider. We focus on one of them here, where two agents, the leader i^* and i, are the first to learn all the information; the arguments for the other cases are similar in spirit, and left to the reader. We have shown that, in this case, i turns passive when it learns all the information as a result of processing a message msg that originated with i^*, and that the number of messages i^* and i have processed by the time they learn all the information is the same. Without loss of generality, assume that both i^* and i first learned all the information after processing their pth message from the left. We showed that either the pth message that i^* processed from its left originated with i, or it originated with some agent i' whose pth message from the left originated with i. It is easy to see that all agents other than i^* and i are passive after they process their pth message, do not have all the information, and are waiting to receive a message from the right. Thus, if i does not send left, then all agents to the left of i up to but not including i^* are deadlocked. Since i is supposed to send left, it cannot be the case that $L_i = i^*$. It easily follows that if i does not send left, and (r', m, i') is an arbitrary situation in $\mathtt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r, m, i)$, then $L_{i'}$ does not learn (i')'s new information nor who the leader is in r'.

We have shown that, for all $r \in \mathbf{R}(P2', \gamma^{br,u})$ and times m, if $P2'_i(r_i(m)) = \mathsf{act}$ then $(\mathcal{J}, r, m, i) \models \varphi_{\mathsf{act}}$. For the converse, suppose that $P2'_i(r_i(m)) \neq \mathsf{act}$. Again, suppose that act is $send_L(new\_info)$. Let (r', m', i') be a situation that i considers possible at time m in run r (i.e., such that conditions (a) and (b) above hold). Since i does not send left at the point (r, m), i' does not send left at the point (r', m'). Thus, by definition, $\mathtt{close}(\overline{do(send_{\mathbf{n}}(new\_info))}, P2', \gamma^{br,u}, r', m', i') = \{(r', m', i')\}$. Since r' is a run of P2$'$, and every agent eventually learns who the leader is in every run of P2$'$, it follows that $(\mathcal{J}, r', m', i') \models \Diamond B_L(f = v)$, and hence

$$(\mathcal{J}, r, m, i) \models \neg do_i(send_{\mathbf{n}}(new\_info)) > \Diamond(\exists \mathbf{n}'(Calls(L, I, \mathbf{n}') \wedge B_L(\mathbf{n}'\text{'s}cont(new\_info))) \vee \exists v B_L(f = v)).$$

Thus, $(\mathcal{J}, r, m, i) \models \neg\varphi_{send_L(new\_info)}$. This completes the proof. ∎

References

Angluin, D. (1980). Local and global properties in netwroks of processors. In *Proc. 12th ACM Symp. on Theory of Computing*, pp. 82–93.

Attyia, H., A. Gorbach, and S. Moran (2002). Computing in totally anonymous asynchronous shared memory systems. *Information and Computation 173(2)*, 162–183.

Attyia, H., M. Snir, and M. K. Warmuth (1988). Computing on an anonymous ring. *Journal of ACM 35(4)*, 845–875.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics 16*(1), 87–90.

Bickford, M., R. L. Constable, J. Y. Halpern, and S. Petride (2005). Knowledge-based synthesis of distributed systems using event structures. In *Proc. 11th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2004)*, Lecture Notes in Computer Science, vol. 3452, pp. 449–465. Springer-Verlag.

Chang, E. and R. Roberts (1979). An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM 22(5)*, 281–283.

Dwork, C. and Y. Moses (1990). Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation 88*(2), 156–186.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1995). *Reasoning about Knowledge*. Cambridge, Mass.: MIT Press. A revised paperback edition was published in 2003.

Fagin, R., J. Y. Halpern, Y. Moses, and M. Y. Vardi (1997). Knowledge-based programs. *Distributed Computing 10*(4), 199–225.

Ford, L. R. and D. R. Fulkerson (1962). *Flows in Networks*. Princeton, N. J.: Princeton University Press.

Friedman, N. and J. Y. Halpern (1997). Modeling belief in dynamic systems. Part I: foundations. *Artificial Intelligence 95*(2), 257–316.

Gallager, R. G., P. A. Humblet, and P. M. Spira (1983). A distributed algorithm for minimum-weight spanning trees. *ACM Trans. on Programming Languages and Systems 5*(1), 66–77.

Grove, A. J. (1995). Naming and identity in epistemic logic II: a first-order logic for naming. *Artificial Intelligence 74*(2), 311–350.

Grove, A. J. and J. Y. Halpern (1993). Naming and identity in epistemic logics, Part I: the propositional case. *Journal of Logic and Computation 3*(4), 345–378.

Hadzilacos, V. (1987). A knowledge-theoretic analysis of atomic commitment protocols. In *Proc. 6th ACM Symp. on Principles of Database Systems*, pp. 129–134.

Halpern, J. Y. and Y. Moses (2004). Using counterfactuals in knowledge-based programming. *Distributed Computing 17*(2), 91–106.

Halpern, J. Y., Y. Moses, and O. Waarts (2001). A characterization of eventual Byzantine agreement. *SIAM Journal on Computing 31*(3), 838–865.

Halpern, J. Y. and L. D. Zuck (1992). A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM 39*(3), 449–478.

Johnson, R. E. and F. B. Schneider (1985). Symmetry and similarity in distributed systems. In *Proc. 4th ACM Symp. on Principles of Distributed Computing*, pp. 13–22.

Le Lann, G. (1977). Distributed systems–towards a formal approach. In *IFIP Congress*, Volume 7, pp. 155–160.

Lewis, D. K. (1973). *Counterfactuals*. Cambridge, Mass.: Harvard University Press.

Lynch, N. (1997). *Distributed Algorithms*. San Francisco: Morgan Kaufmann.

Mazer, M. S. and F. H. Lochovsky (1990). Analyzing distributed commitment by reasoning about knowledge. Technical Report CRL 90/10, DEC-CRL.

Milner, R. (1989). *Communication and Concurrency*. Hertfordshire: Prentice Hall.

Moses, Y. and G. Roth (1989). On reliable message diffusion. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pp. 119–128.

Peterson, G. L. (1982). An $O(n \log n)$ unidirectional distributed algorithm for the circular extrema problem. *ACM Trans. on Programming Languages and Systems 4*(4), 758–762.

Stalnaker, R. C. (1968). A semantic analysis of conditional logic. In N. Rescher (Ed.), *Studies in Logical Theory*, pp. 98–112. Oxford University Press.

Stulp, F. and R. Verbrugge (2002). A knowledge-based algorithm for the Internet protocol (TCP). *Bulletin of Economic Research 54*(1), 69–94.

Yamashita, M. and T. Kameda (1996). Computing on anonymous networks. I. Characterizing the solvable cases. *IEEE Trans. on Parallel and Distributed Systems 7*(1), 69–89.

Yamashita, M. and T. Kameda (1999). Leader election problem on networks in which processor identity numbers are not distinct. *IEEE Trans. on Parallel and Distributed Systems 10*(9), 878–887.