ASPECTS OF KNOWLEDGE AND BELIEF-BASED PROGRAMMING

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Sabina Petride

May 2009

ASPECTS OF KNOWLEDGE AND BELIEF-BASED PROGRAMMING

Sabina Petride, Ph.D.

Cornell University 2009

It has long been recognized that many distributed problems can be analyzed in terms of how agents act based on what they *know* about the system they are in. To make this intuition formal, Fagin, Halpern, Moses, and Vardi [25, 26] proposed a theory of programs for multi-agent systems in which preconditions of actions are formulas in a logic of knowledge. *Knowledge-based programs* have been successfully applied to a number of fundamental problems. This dissertation aims at further investigating the role of knowledge-based programs in the study of distributed systems.

We focus first on a general problem in distributed settings: given a function f whose value depends on the whole network N, the goal is for every agent to eventually compute the value $f(N)$. We call this problem *global function computation*. We give a necessary and sufficient condition for the problem to be solvable and provide a knowledge-based program that solves the global function computation problem whenever possible. The program guarantees a certain level of optimality by having agents send messages only when they believe it is necessary to do so, and allows for systems in which agents are anonymous.

Second, we consider the synthesis of knowledge-based programs. In general, to produce a program guaranteed to satisfy a given specification, one can try to synthesize it from a formal proof that a computation satisfying that specification exists. We build on a technique proposed by Bickford and Constable [11, 12] for extracting message automata from specifications of multi-agent systems to show how knowledge-based message automata can be synthesized using a proof development system such as Nuprl [19].

Third, we consider the problem of ensuring secrecy in a multi-agent system by disallowing unwanted flows of information. We discuss knowledge-based formulations of information-flow properties, with a focus on the basic security predicates in the Modular Assembly Kit proposed by Mantel [62].

BIOGRAPHICAL SKETCH

I was born in Bucharest, Romania. I was a very serious child, not that there was much room for laughter back in those years. It was also a happy time, as my parents were very young and my sister was my best friend. The school system was strict, but I owe to it, and to the long list of problems to be solved in summer holidays, a good mathematical education. I went to the University of Mathematics in Bucharest. In my last year, my thesis advisor, Luminita State, had stacks of artificial intelligence books and I could borrow any of them. Everybody was talking about going for grad school abroad, and I followed. I chose Cornell University. I was very enthusiastic in my first year, and wanted to enroll immediately in the most advanced classes. Advanced distributed system was one of them, but soon I realized that, if I want to be serious about it, I needed to learn the basics. I switched to the Computer Science department, and for the next few years joined the group of students who take some sort of hidden pride in complaining about Qualifying exams, knowing that they would never give up, though life could be much easier somewhere else. By the end of this experience I finally felt free to wake up before sunrise, make my way through the freezing cold of Ithaca winters, stop at Collegetown Bagels to warm up, and head to my office, where I could read almost anything I found of interest. Five years after coming to Cornell, I accepted a position as a software developer at Oracle. This forced me to see what I learned in school with different eyes, and realize the purpose and place of what seemed just course material. For me, it was very important to connect the present years of (I hope) growing up with the past years of idealism. I see this thesis as a product of the latter, but with a newly found interest for academics.

To my family

ACKNOWLEDGEMENTS

I am very grateful to my professors, here at Cornell and back home in Bucharest. I have immense respect for the graduate system in US for admitting people based solely on academic credentials. And I have great respect for the people who keep a bit of the enthusiasm they started with in the years of big decisions and leaps of faith that come after.

There are many people this dissertation would have not been finished without, professionally and personally. I would like to thank my advisor, Joe Halpern, for almost eight years of open communication. From him I have learned to strive for clarity and truthfulness in expression, and to write with an audience in mind. I thank him for the financial support that gave me a sense of freedom in pursuing any interest I have, and for providing me the opportunity to get a glimpse at the academic community through conferences and extended group meetings. I am grateful to Riccardo Pucella for lighthearted and confidence-building encouragement, for always finding time and interest to talk, and for making the best out of any new ideas. I also want to thank Robert Constable and Mark Bickford for the long hours of discussions in the Nuprl lab. There are other people in the Nuprl group that always had an open door: Christoph Kreitz and Rich Eaton. I would also like to thank Dexter Kozen and Robbert van Renesse for their understanding and trust; although I did not collaborate with them, some of the interest for choosing my thesis topic comes from reading their articles. Being somehow disconnected from the university life for a number of years made me appreciate the advice of all my committee members. It also made things much easier to always be able to count on Cindy Robinson, Becky Stewart, and Kelly Patwell for help.

In spite of the long graduate years of training for developing a fluid, polished and seemingly effortless writing style, it is still difficult for me to put into words what I owe to my family. I hope it goes without saying that I sincerely wish finishing this effort makes them happy. My parents have unselfishly accepted having their daughters live so far away. My sister has always listened to what I have to say, and my brother-in-law trusted that I can face any situation and I have the right to choose. I credit the happiness of most of my graduate-school years to Hasan; from him, I learned to be patient, think more, and tolerate what I may find difficult to understand.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Agents in a distributed system act based on what they know, or do not know. The notion of action as a function of agents' knowledge is well-established in the distributed systems literature [15, 27, 28] and a number of fundamental results make use of it [21, 39, 42, 46, 65, 74].

Fagin, Halpern, Moses, and Vardi [25, 26] suggested the use of *knowledge-based programs* to describe the connection between knowledge and actions. Formally, knowledge-based programs [26, 40, 41, 73] are programs in which preconditions of actions are epistemic formulas. For example, an agent i that sends a message msg to agent j only if he does not know that j knows some fact φ can be seen as following the knowledge-based program

$$\textbf{if } K_i(K_j\varphi) \textbf{ then } skip \textbf{ else } send(msg).$$

(As usual, we write $K_i\varphi$ to denote that agent i knows φ.)

Knowledge-based programs have been successfully applied to a number of problems, such as simultaneous actions and coordination [73], sequence transmission [50], analyzing the TCP protocol [82], and consensus [71]. They have the advantage of abstracting away from details of implementations or specifics of the system. Often, a single knowledge-based program can be written to characterize the behavior of agents in different environments. This is particularly important in the context of distributed systems that vary widely in the types of assumptions made about communication (e.g., via shared memory or message-passing), network structure, reliability, and agents (e.g., faulty or robust, anonymous, mobile, active participants in a protocol or passive wire-tappers, honest or malicious, etc.).

This thesis is concerned with the use of knowledge-based programs for solving distributed problems, and focuses on three aspects: (1) analyzing the general problem of computing a function of the network in a distributed system and formalizing a program that solves this problem in a message-optimal way; (2) extending existing techniques for program synthesis to allow for knowledge-based specifications, and (3) analyzing information-flow properties as imposing constraints on what an adversary may know or should never learn about the system. The purpose of this section is to provide evidence that these goals are relevant and to motivate our approach.

1.1 The global function computation problem

A number of classic distributed problems can be seen as instances of a more general problem of computing a *global function*, i.e., a function of the entire network. For example, if each agent has a certain input value, the goal may be for all agents to eventually compute an average (maximum etc.) of all the inputs. This is the case for the well-known *leader election problem* [60]: the leader is the agent with the largest (or smallest) input. Computing characteristics of the network seen as a graph, like the diameter, size, upper bound on the size, a minimum spanning tree [34], or the shortest paths between nodes [10, 32], can also be seen as instances of the global function computation problem.

If the network is known by all agents, then computing a global function is typically straightforward. However, in a distributed setting, agents have only limited information about the network, which makes the problem more difficult. For example, agents may have information only about their input value and a small neighborhood in the network, but have no information about the number of agents in the network. The problem increases in complexity in settings where agents cannot be uniquely identified (by, for example, an IP address or input value) by others agent in the system. For instance, when agents know their identifiers and all identifiers are unique, several solutions for the leader-election problem have been proposed, both in the synchronous and asynchronous settings [17, 56, 77]. On the other hand, Angluin [4], and Johnson and Schneider [54] proved that it is impossible to deterministically elect a leader if agents may share names.

In general, the difficulty of solving the global function computation problem depends on what agents know. The questions we want to answer are:

- What do agents need to know for the problem of computing a global function f to be solvable?
- What is a general protocol for solving the global function computation problem?
- What do existing protocols for computing a global function f have in common?

To answer these questions, we describe a simple knowledge-based program for computing a global function. The program says that, at each point in time, and for any agent, if the agent has some new information about the network or the function value, he should send it to all his neighbors he does not know that they know his new information. Intuitively, if it is possible for agents on N to compute $f(N)$, then this program should allow agents to do so.

There are, however, a number of subtleties in making this simple knowledge-based program precise, like defining what means for an agent to have new information, allowing settings in which agents may not know the identities of their neighbors, and formalizing what it means for an agent to reason about what different agents know about his information. More explicitly, if we are to take anonymous agents into account, we cannot write a formula $K_iK_j\varphi$ to denote the fact that agent i knows that j knows φ, since i may not know that his neighbor has identifier j. Instead, we can assume that agent i has a way of naming each of his neighbors (for example, by "left" or "right" if the network is known to be a ring), and if i names his neighbor $\mathbf{n}$, then in a program for agent i we $K_iK_{\mathbf{n}}\varphi$ instead of $K_iK_j\varphi$.

While the knowledge-based program can be shown to solve the global function computation problem whenever possible, it is not optimal. We can use fewer messages by requiring that agents send messages only when they know this is necessary, that is, if some goal would not be achieved if the message were not sent. For the purpose of global function computation, the goal is for the agent to learn the function value.

Changing the knowledge-based program along these lines is not straightforward. The key problem is that, while in all executions of the program on a given network, an agent may actually always send his new information to his neighbors, to decide whether to send his information or not, the agent has to reason about what would happen had he not sent the information. In other words, the agent has to reason about counterfactual situations. *Counterfactual-based programs* [43] are programs in which preconditions on actions may contain counterfactual implications. We extend the formalization of counterfactual-based programs to allow for anonymous agents. This allows us to write a program that solves the global function computation problem whenever possible, where no unnecessary messages are sent. This is the subject of Chapter 2. This material is joint work with Joseph Halpern; a preliminary version appeared in [48].

1.2 Program synthesis from knowledge-based specifications

For the analysis of problems like global function computation, there is a clear intuition about what the knowledge-based program for solving the problem should be; the difficulty lies in formalizing the program and proving its correctness. However, in general, finding a program that satisfies a given specification may not be easy.

One proposed methodology builds upon *refinement calculus* [7] and *constructive logic* [58]. In refinement calculus, the specification is given as a formula, which is then refined into subgoals. With constructive logic, the proof of the formula characterizing the specification actually provides a program that satisfies the specification. Moreover, the program constructed this way is *guaranteed* to be correct. For this reason, this approach is known as *correct by construction* program extraction.

Constructive logic has been successfully applied to extract sequential programs from proofs that a specification is correct [1], but relatively less research has been carried out for the extraction of distributed programs from specifications about distributed systems. An approach for program extraction from specifications of distributed systems in an event-based formalism has been proposed by the Nuprl group [11, 12]. Distributed systems are described in terms of sequences of events with additional structure, called *event structures*. Specifications are then predicates on event structures. From the proof that a specification is satisfiable, a program in this language that satisfies the specification can be extracted.

Our goal is to extend the framework to express knowledge. We show how Nuprl can be extended to allow epistemic formulas, and how the programming language can be generalized to allow for knowledge-based programs. We give a number of axioms that connect knowledge-based programs to the specifications they satisfy. We extend the framework to allow for extracting knowledge-based programs from knowledge-based specifications and apply it to the sequence transmission problem [50]. This is the subject of Chapter 3. This material represents joint work with Mark Bickford, Robert Constable, and Joseph Halpern; a preliminary version appeared in [13].

1.3 Knowledge-based formulations of information-flow properties

In Chapter 4 we turn our attention to a problem that has received increased interest in distributed systems community, *secrecy*. In particular, we focus on the notion of secrecy interpreted as the lack of information flow between domains at different levels of security [35, 36, 66, 75, 85] and on its formulation in a logic of knowledge.

A system is considered to exhibit an insecure information flow if a user at a lower level of security can gain information about the activity at higher levels of security. Many variations on this theme are possible, depending on what type of information is considered sensitive, and on the capabilities of the unauthorized users. Our focus is on one of the frameworks for expressing information-flow requirements that has gained widespread acceptance, the Modular Assembly Kit (MAKS) proposed by Mantel [62]. The main motivation in our choice is that MAKS was proven to express a number of well-known information-flow constraints [64]. Furthermore, a key feature of the framework is modularity, as it allows building complex information-flow properties from basic security predicates. With this approach, the problem of analyzing whether a system satisfies a complex security property reduces to verifying that the system satisfies all the basic security predicates that make up the complex property [51]. Thus, the MAKS framework is easily amenable to proof techniques based on refinement calculus. Indeed, a number of automatic provers have been developed for the framework [63].

It has been long recognized that information-flow properties have simple intuitive reformulations as restrictions on what users at lower levels of security may know about users at higher levels of security. Formally, these restrictions can be expressed as epistemic formulas, which have the advantage of simplicity and closeness to natural-language formulations [47]. Up to now, the information-flow properties that have been expressed in epistemic logic are ones that preclude the low-level users from ever knowing some function of the local state of the high-level users. Some of the building blocks in Mantel's proposed framework do not seem to naturally fit into this category. More specifically, Halpern and O'Neill [47] define a j-information function f as any function that depends only on agent j's local state, and focus on formalizing what it means for an agent i to never learn any information about the values of such functions. Broadly speaking, this is shown equivalent to ensuring the validity of all formulas $\neg K_i\varphi$, where φ is any nontrivial fact whose interpretation depends only on agent j's state. MAKS properties, on the other hand, are typically formulated as closure conditions on a set of traces (i.e., sequences of events) that describe the executions of a system. They say that for each trace in the system, there exists another trace in the system that satisfies some properties. For example, one may require that removing the last confidential event in a trace gives a trace the low-level user cannot distinguish from the original trace.

We show how we can express a number of MAKS-style definitions in terms of knowledge. In the process of doing this, we show that, while the intuition behind the definitions seems clear, the translation into epistemic framework is not straightforward. A closer look at these definitions raises some questions:

- To what extent do the MAKS building blocks match the informal explanations typically given for them?
- Can we provide intuitive explanations for the differences between MAKS properties and their natural variations?

There has been increased interest in extensions of the definitions for some of the MAKS building blocks to take cryptography into account. Some definitions have been given recently by Hutter and Schairer [53]. We show that the reformulations of MAKS-style information-flow properties as knowledge-based formulas can be naturally extended to cryptographic settings, where agents' inability to distinguish two encrypted messages is taken into account. Our definitions are different from those proposed by Hutter and Schairer. We show that the differences lie in the assumptions about the adversary's capabilities. While these assumptions are implicit elsewhere in the literature, the knowledge-based formulation of information-flow properties has the advantage of making explicit the connection between an adversary's knowledge and his capabilities.

CHAPTER 2
KNOWLEDGE-BASED ANALYSIS OF GLOBAL FUNCTION COMPUTATION

2.1 Overview

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 [34] is known for its complexity.[1] Moreover, the algorithm does not work for all networks, although 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 [10, 32]. The well-known *leader election problem* [60] 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 agents know.

[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.

For example, when agents 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 [17, 56, 77]. On the other hand, Angluin [4], and Johnson and Schneider [54] proved that it is impossible to deterministically elect a leader if agents may share names. In a similar vein, Attiya, Snir and Warmuth [6] 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 [5] 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 [25, 26].

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 [6]. 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 [86] 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 that 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 2.2, 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* [69], 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 [6] and Yamashita and Kameda [87].

We then show that the simple program where each agent just forwards all the new information he 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 by using techniques introduced by Moses and Roth [72] and further developed by Grove and Halpern [37, 38]. Observe that it makes perfect sense to talk about each agent acting based on his own knowledge by saying "if *I* know φ, then $\ldots$". 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 [43] the most obvious way of expressing it does not work. However, we can capture this intuition by using *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 2.3.1, 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 2.6 that the counterfactual-based program is implemented by the protocols of Le Lann, Chang and Roberts [56, 17], and Peterson [77], which all work in rings (under slightly different assumptions), and by the optimal flooding protocol [60] in networks of bounded diameter. Thus, the kb program with counterfactuals shows the underlying commonality of all these programs and captures a key intuition behind their design.

The rest of this paper is organized as follows. In Section 2.2 we give our characterization of when global function computation is possible. In Section 2.3 we formalize the flooding protocol and prove that it solves the global function computation problem whenever possible. In Section 2.4 we describe the kb program for global function computation, and in Section 2.5 show how to optimize it so as to minimize messages. In Section 2.6, we show that the program essentially implements some standard solutions to leader election in a ring.

2.2 Characterizing when a global function is computable

We model a network as a directed, simple (no self-loops and at most one edge between each pair of nodes), strongly connected, and 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 message delivery is handled by the channel (and is not under the control of the agents).

We assume that initially agents know their *local information*, i.e., their own input value, the number and orientation of their links, and the weights associated with their outgoing links. Moreover, agents can distinguish among links with the same orientation and weight; that is, agents view their incoming and outgoing links an ordered tuples, not as multisets. However, agents do not necessarily initially know the weights on non-local edges, the weights on their incoming 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 have the same names as 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. A *global function* f is a function on networks that 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 $In_N(i) = \{j \in V(N) \mid (j,i) \in E(N))\}$; an undirected link between i and j is modeled by having both (i,j) and (j,i) in N. The *neighbors* of i are the agents in $Out_N(i) \cup In_N(i)$. Let $inp_N(i)$ denote i's input value. Finally, if e is an edge in $E(N)$, then $w_N(e)$ denotes e's label.

Definition 2.2.1: A global function f is *computable* in a set $\mathcal{N}$ of networks if and only if there exists a protocol P such that, for all networks $N \in \mathcal{N}$, if agents run P on N, then eventually P terminates and returns $f(N)$. In this case, we say that P *computes* f in $\mathcal{N}$. ∎

Note that we consider here an information-theoretical notion of computability; we are not concerned with computation time. Also note that computability is relative to the set $\mathcal{N}$ of networks. To capture the fact that only networks in $\mathcal{N}$ are being considered, we assume that $\mathcal{N}$ is part of each agent's initial information (and, thus, is common knowledge).

We consider both synchronous and asynchronous systems. In a synchronous system, we assume that a round consists of a sending phase and a receiving phase. All round k messages that are sent in the sending phase are received in the receiving phase of round k. In the asynchronous setting, we assume that all messages sent are eventually received, but make no assumptions about the delivery time. Messages can be received in arbitrary order. Our focus in this section is on *full-information protocols*, where an agent starts by sending to all his neighbors his initial information, and then forwards to all his neighbors all the messages he receives.

Intuitively, the full-information protocol reduces uncertainty. For example, suppose that $\mathcal{N}$ consists of all directed 3-node rings, and let N be a 3-node ring in which agents have inputs a, b, and c, and all edges have the same weight w. For simplicity, suppose that the system is synchronous. Let i be the external name of the agent with input a. Initially, i considers possible all 3-node 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. More explicitly, i knows the weighted graph N up to isomorphism. But this depends on the fact that i knows that the ring has size 3.

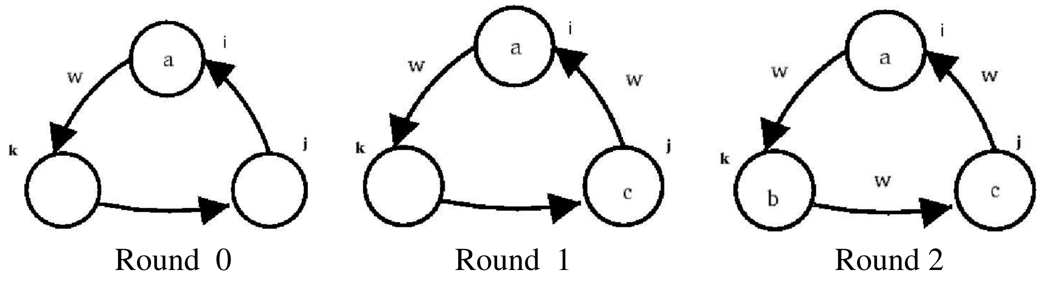

Figure 2.1: How i's information changes with the full-information protocol.

Now consider the same network N, but suppose that agents do not know the ring size, that is, $\mathcal{N}$ is the set of all directed rings (i.e., rings where messages can be sent in only one direction), 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 2.2). 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.

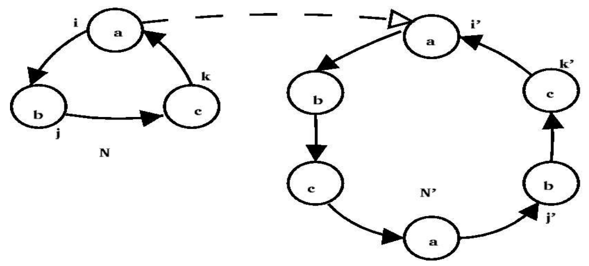

Figure 2.2: Two indistinguishable networks.

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 2.2.2: Given networks N and N' and agents $i \in V(N)$ and $i' \in V(N')$, i and i' are 0-*bisimilar*, written $(N,i) \sim_0 (N',i')$, iff

- $inp_N(i) = inp_{N'}(i')$;
- 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 $w_N(i,j) = w_{N'}(i', f^{out}(j))$, and
- $In_N(i)$ and $In_{N'}(i')$ have the same number of elements.

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

- $(N,i) \sim_{k-1} (N',i')$, and

- there is a bijection $f^{in} : In_N(i) \longrightarrow In_{N'}(i')$ such that for all $j \in In_N(i)$
  - $w_N(j,i) = w_{N'}(f^{in}(j), i')$, 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 in synchronous systems:

Lemma 2.2.3: *The following are equivalent in synchronous systems:*

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

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

(c) Agents i and i' have the same initial 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 on k, with the base case following from the definition of $\sim_0$. Suppose that, if $(N,i) \sim_k (N',i')$, then i and i' have the same initial information and receive the same messages 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 induction hypothesis, it follows that i and i' have the same initial information and receive the same messages in the first k rounds of P; moreover, for each incoming neighbor j 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.

The proof that (c) implies (b) is immediate, since the full-information protocol is a special case of a deterministic protocol.

Finally, we prove that (b) implies (a) by induction on k. For $k = 0$, it is clear from Definition 2.2.2 that $(N, i) \sim_0 (N', i')$ exactly when i and i' have the same initial information. For the inductive step, suppose that i and i' have the same initial information and receive the same messages at each round $k' \leq k+1$. Since agents are running a full-information protocol, we can construct a mapping 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 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, $(N, i) \sim_{k+1} (N', i')$. ∎

We now show that, even in asynchronous systems, if the function f can be computed in a network $N \in \mathcal{N}$ at, then it can be computed using a full-information protocol. An agent can output the value of f when he knows the value, that is, when f takes on the same value at all networks that the agents consider possible. The exact round at which it happens (if ever) depends on the network N, the function f, and the set $\mathcal{N}$ of possible networks. Moreover, if there is no such round (i.e., at all times, there exists some agent that considers it possible that the network is $N' \in \mathcal{N}$, where $f(N') \neq f(N)$), then f is not computable in N with respect to $\mathcal{N}$. Using Lemma 2.2.3, we can characterize if and when it happens.

Definition 2.2.4: f is *knowable with respect to N and $\mathcal{N}$* if

$$\exists k_{\mathcal{N},N,f}.\, \forall N' \in \mathcal{N}.\, \forall i \in V(N).\, \forall i' \in V(N').\, ((N,i) \sim_{k_{\mathcal{N},N,f}} (N',i') \Rightarrow f(N') = f(N));$$

that is, if there exists a constant $k_{\mathcal{N},N,f}$ (which may depend on $\mathcal{N}$, N and f) such that f takes the same value on N as on any network N' in $\mathcal{N}$ such that there exist two $k_{\mathcal{N},N,f}$-bisimilar agents i in N and i' in N', respectively. *f is knowable with respect to $\mathcal{N}$* if f is knowable with respect to N and $\mathcal{N}$ for all $N \in \mathcal{N}$. ∎

Definition 2.2.5: Agent i *considers (N, i') possible at time k in run (execution) r of protocol P* if $i' \in V(N)$, i has the same initial information as i', and there exists a run r' of protocol P and time k' such that i has received the same sequence of messages at time k in run r as i' has at time k' in run r'. Agent i *knows that the value of f is v at time k in run r of protocol P* if $f(N) = v$ for all networks N such that i considers (N, i') possible for some $i' \in V(N)$ at time k in run r. ∎

We remark that the notion of knowledge used here is consistent with the formal definition of knowledge given in Section 2.4.2.

Theorem 2.2.6: *The global function f is computable in $\mathcal{N}$ if and only if f is knowable with respect to $\mathcal{N}$.*

Proof: First suppose that the system is synchronous. Consider the following protocol P. In round 1, each agent sends all its initial information to all its neighbors. For $k > 1$, if agent i receives the value of f in round $k-1$ or knows the value of f at the beginning of round k of the full-information protocol, then he outputs the value of f, sends the value to all his neighbors in round k, and then halts. Otherwise, i forwards all the messages he has received in round $k-1$ to all his neighbors.

We now show that protocol P computes f. Suppose not. Then there must be some network $N \in \mathcal{N}$ and agent $i \in V(N)$ such that agent i does not output $f(N)$. Since N is strongly connected, it easily follows that no agent in N outputs $f(N)$. (If j outputs $f(N)$, and k is the length of the shortest path from j to i, then i will output $f(N)$ within at most k steps.) Thus, all agents run the full-information protocol in N. Since f is knowable with respect to N and $\mathcal{N}$, there must be some constant $k_{\mathcal{N},N,f}$ such that $\forall i \in V(N).\ \forall i' \in V(N').\ ((N,i) \sim_{k_{\mathcal{N},N,f}} (N',i') \Rightarrow f(N') = f(N))$. By Lemma 2.2.3, it follows that i knows the value of f at some round $k \leq k_{\mathcal{N},N,f}$, and thus outputs $f(N)$, contradicting our initial assumption.

Now suppose that the system is not synchronous. We consider essentially the same protocol P: each agent sends his initial information to all his neighbors. Suppose that i receives a message. If the message is either the value of f or i knows the value as the result of receiving the message using the full-information protocol, then i outputs the value of f, sends it to all his neighbors, and terminates; otherwise, i forwards the message to all his neighbors. We claim that, again, P computes f in $\mathcal{N}$. Suppose not. Then there must exist some $N \in \mathcal{N}$, agent $i \in V(N)$, and a run r of P such that agent i does not output $f(N)$ in r. Again, it easily follows that no agent in N outputs $f(N)$ r, so all agents are essentially following the full-information protocol in r. Since all the messages sent are eventually received, for each all agents j and j' and times m, agent j must send a message after time m in r.

We claim that, for all rounds k, and all agents j, there exists a time k_j such that the set of (N', j') pairs that j considers possible at time k_j in r is a (not necessarily strict) subset of the pairs that j considers possible at the beginning of round k of a run of the full-information protocol in network N in a synchronous system. Roughly speaking, this says that, even in an asynchronous system, agents eventually learn everything that they do in the synchronous system using the full-information protocol. We proceed by induction on k. The base case is immediate: at time 0 in r, each agent j considers exactly the same set of networks possible as at the beginning of round 1 of a synchronous system, since j has the same initial information in both cases. For the inductive step, first note that j considers (N', j') possible at the beginning of round k in the synchronous system iff (a) j considers (N', j') possible at the beginning of round $k-1$ and, (b) for all $h \in In_N(j)$, there exists a neighbor $h' \in In_{N'}(j')$ such that h considers (N', h') possible at the beginning of round $k-1$. Since the set of pairs that an agent considers possible is non-increasing over time, by the induction hypothesis, there exists a time k' in run r such that, for all $h \in In_N(j) \cup \{j\}$, the set of pairs that h considers at time k' in r is a subset of the set of pairs that h considers possible at the beginning of round $k-1$ of a run of the synchronous system in N. Since all agents in $In_N(j)$ send j a message after time k' in r, and these messages are received by some time k'', it follows that at time k'', the set of pairs that j considers possible must be a subset of the set of pairs that j considers possible at the beginning of round k in the synchronous system. Since i eventually knows the value of f in the synchronous system, i must also eventually know the value of f in run r, contradicting the initial assumption.

Now suppose that it is not the case that f is knowable with respect to $\mathcal{N}$. There must be some network $N \in \mathcal{N}$ such that f is not knowable with respect to N and $\mathcal{N}$. We first show that f is not computable in $\mathcal{N}$ in a synchronous system using the protocol P described above. For suppose that f is computable, and let j be the first (or one of the first, in case of a tie) that outputs $f(N)$. Suppose that j does so in round k. Thus, j must know the value of f at the beginning of round k. Let d be the diameter of N. It is easy to see that all agents will know the value of f by the beginning of round $k+d$. But, since f is not knowable with respect to N and $\mathcal{N}$, there must be some network $N' \in \mathcal{N}$ and agents $i \in V(N)$ and $i' \in V(N')$ such that $(N, i) \sim_{k+d} (N', i')$. By Lemma 2.2.3, agents i and i' have the same initial information and receive the same messages in each of the first $k + d$ rounds of protocol P. Thus, they must output the same value of f in both N and N'; one of these values must be wrong. Thus, P does not compute f in $\mathcal{N}$.

Finally, suppose that f is computable in $\mathcal{N}$ in an asynchronous system using some protocol P'. Let r' be a run of P' where the network is N, and let j be the first agent (or one of the first, in case of a tie) to output the value of f in r'. Suppose that j outputs $f(N)$ at time k in r'. We claim that j (and hence all agents) will output the value of f using P in network N in a synchronous system, contradicting the previous paragraph. For suppose not. Then P in the synchronous system works just like the full-information protocol. For all k', let $g(k')$ be one more than the total number of messages received by all agents by time k' in run r. We claim that for each time $k' \leq k$ in r' and each agent $j \in V(N)$, if j considers (N', j') possible at the beginning of round $g(k)$ in the synchronous system, using P, then j considers (N', j') possible at time k' in r'. That is, all agents know at least as much at the beginning of round $g(k)$ using P as they do at time k' in r. We can prove this by a simple induction on $g(k')$; we leave details to the reader. Since j knows the value of f at time k in r', it follows that j must know the value of f at time $g(k)$ using protocol P in the synchronous system, giving us the desired contradiction. ∎

Intuitively, $k_{\mathcal{N},N,f}$ in the definition of knowable 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 2.2.7: *If no network in $\mathcal{N}$ has two agents that are locally the same, then all global functions are knowable (and hence computable) in $\mathcal{N}$; indeed, we can take $k_{\mathcal{N},N,f} = diam(N) + 1$ for all global functions f and networks $N \in \mathcal{N}$.*

Proof: Since $f(N) = f(N')$ if N and N' are isomorphic, by Theorem 2.2.6, it suffices to show that $(N,i) \sim_{diam(N)+1} (N',i')$ implies that N and N' are isomorphic for all $N, N' \in \mathcal{N}$. So suppose that $(N,i) \sim_{diam(N)+1} (N',i')$. 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, j' must be unique, since if $(N,j) \sim_{diam(N)+1-k} (N',j'')$, then 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 that j and $h(j)$ are locally the same guarantees 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 exists a node $j_1' \in V(N')$

of distance $diam(N) + 1$ from i'. Let $j_2' \in V(N')$ be such that j_2' is the last last node preceding j_1' on a shortest path in N' from i' to j_1'. It follows that the distance from i' to j_2' in N' is $diam(N)$, so by the definition of N'', j_2' is a node of N''. Let $j_2 = h'(j_2')$. By the definition of h', $(N', j_2') \sim_1 (N, j_2)$. Since j_1' is one of the outgoing neighbors of j_2' in N', j_2 must have an outgoing neighbor j_1 in N such that $(N', j_1') \sim_0 (N, j_1)$. Let $j_3' = h(j_1)$. The distance in N' between i' and j_3' is the same as the distance in N from i and j_1, and so must be at most $diam(N)$. Moreover, since j_3' and j_1' are locally the same, we must have $j_3' = j_1'$. It follows that the distance from i' to j_1' is at most $diam(N)$, which contradicts the assumption that j_1' is of distance $diam(N) + 1$ from i'.

▮

Attiya, Snir, and Warmuth [6] prove an analogue of Lemma 2.2.3 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 that $f(N) = f(N')$ for Theorem 2.2.6.[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 2.2.6 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,

[2]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.

but our characterization is arguably cleaner.

Theorem 2.2.7 sheds light on why the well-known protocol for minimum spanning tree construction proposed by Gallager, Humblet, and Spira [34] 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.

2.3 A standard program for global function computation

2.3.1 Standard programs with shared names

A standard *program* Pg has the form

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

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 whose test t_j is satisfied; if no such action exists, i does nothing. We sometime use obvious abbreviations like **if** . . . $\mathbf{then}$ . . . **else** (which can be expressed using a case statement).

Following Grove and Halpern [37, 38] (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. In ring networks, for example, $\mathbf{N}$ may contain names like "left", "right", or the "neighbor to the left of the left neighbor"; in arbitrary networks, $\mathbf{N}$ may contain names like "the neighbor on link 1", "the neighbor on link 2 of the neighbor on link 1", etc. Although we do not require the set of names to be finite, we assume that $\mathbf{N}$ is commonly known. Note that this assumption is not unreasonable, since the set $\mathcal{N}$ of all possible networks is common knowledge.

We assume that an agent assigns distinct names to all his neighbors on incoming and outgoing edges. (An agent may not initially realize that a neighbor on an outgoing link is the same as a neighbor on an incoming link, and thus will initially assign them different names.) 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 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. 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.

We are interested in facts that an agent knows about the network. Intuitively, these are facts about the network viewed as a weighted graph, such as the number of nodes in the network, the number and orientation of links, the weights on the links, the number of neighbors of an agent i, and the input of agent i. Formally, a piece of information can be identified with a set of pairs (N', i'), intuitively, the networks N' for which the fact is true from the point of view of i'. In the program, we use a primitive proposition $some\_new\_info$ that we interpret as true for agent i if and only if i has learned some new information in the last round. When $some\_new\_info$ is true with respect to agent i, we take new_info to be a representation of the new information that i has learned; that is, new_info is a representation of the pairs (N', i') that i learned are not possible (or, equivalently, the set of pairs (N', i') that i currently considers possible). (This is formalized in Section 2.3.2.) The action $send_{\mathbf{n}}(new\_info)$ in the program has the effect of i sending $\mathbf{n}$ whatever new information i learned. (We expect that in most cases of interest, new_info can be represented by a succinct formula, although defining an appropriate language for representing new_info is beyond the scope of this thesis.)

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

$$\textbf{if } some\_new\_info \textbf{ then } send_{\mathbf{Nbr}}(new\_info).$$

We can modify it to get a terminating protocol for global function computation so that an agent sends 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.

2.3.2 Protocols, systems, and contexts

We interpret programs in the *runs and systems* framework of Fagin et al. [25], 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 is in some local state. 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 an undirected ring, each agent must have two outgoing edges according to his 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.

Given a *context*, we can associate with each joint protocol P a system. 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* $\mu : \mathcal{G} \times \mathcal{A} \times \mathbf{N} \rightarrow \mathcal{A}$, 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, a subset of the set $\mathcal{G}$ of global states, $\tau : \mathcal{G} \rightarrow \mathcal{G}$ is a *transition function* mapping global states to global states, 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 associates 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. As we shall see, we use the naming function to ensure that actions that involve names are interpreted correctly. 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 $\gamma = (P_e, \mathcal{G}_0, \tau, \mu)$ for global function computation has the following features:

[3]Fagin et al. [25] 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.

- 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. Formally, we assume that P_e has actions of the form $deliver_i(\mathbf{m})$, which result in message $\mathbf{m}$ being delivered to player i. In a *synchronous context*, the environment's protocol ensures that all messages sent at the beginning of round k are received by the beginning of round $k + 1$.
- 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 information (which includes $\mathcal{N}$, the set of possible networks). Formally, we assume that agent i's local state is a sequence whose first element is i's initial information, and the later elements are the events of sending and receiving messages. 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 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 $s' = \tau(\mathsf{act})(s)$, 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 or received as a result of action act to i's local state in s. We assume that τ 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) and $s' = \tau(\mathsf{act})(s)$, then in s', j's local state records the fact that j has received the information from i.

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 $(\mathsf{act}_e, \{\mathsf{act}_i : i \in \mathcal{A}(r)\})$ such that $\mathsf{act}_e \in P_e(\ell_e)$, $\mathsf{act}_i \in P_i(\ell_i)$ for $i \in \mathcal{A}(r)$, and $r(m+1) = \tau(\mathsf{act}_e, \{\mathsf{act}_i : i \in \mathcal{A}(r)\})(r(m))$ (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 the programs we consider, the truth of a test may be relative to an agent. For example, the test $\mathit{some\_new\_info}$ is true relative to agent i at a point (r, m) if i has just received some new information about the network. We actually want the interpretation of $\mathit{some\_new\_info}$ to depend, not on the point (r, m), but only the global state $r(m)$ at that point. Thus, given a set Φ of primitive propositions, we take an *interpretation* (of the primitive propositions in Φ) to be a function $\pi : \mathcal{G} \times \mathcal{A} \times \Phi \rightarrow \{\mathit{true}, \mathit{false}\}$. Intuitively, $\pi(g, i, p) = \mathit{true}$ if p is true at the global state g relative to agent i. Of course, we can easily extend this truth assignment to arbitrary propositional formula; for example, we take $\pi(g, i, \neg\varphi) = \mathit{true}$ iff $\pi(g, i, \varphi) = \mathit{false}$, $\pi(g, i, \varphi \wedge \psi) = \mathit{true}$ iff $\pi(g, i, \varphi) = \mathit{true}$ and $\pi(g, i, \psi) = \mathit{true}$, etc.

An interpretation π is *local* (for program Pg) 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 $\pi(g, i, \varphi) = \mathit{true}$ iff $\pi(g', j, \varphi) = \mathit{true}$. In this case, we write $\pi(\ell, \varphi) = \mathit{true}$. Given an interpretation π that is local, we can associate with a program Pg for agent i a protocol Pg^π. Define $Pg^\pi(\ell) = \{\mathsf{act}_j \mid \pi(\ell, t_j) = \textit{true}\}$ if there exist tests t_j such that $\pi(\ell, t_j) = \textit{true}$, and take $Pg^\pi(\ell) = \mathsf{skip}$ otherwise. Define $\mathbf{I}(Pg, \gamma, \pi) = \mathbf{R}(Pg^\pi, \gamma)$.

An *interpreted context for global function computation* is a pair $\eta = (\gamma, \pi)$, where γ is a context for global function computation and π interprets $some\_new\_info$ appropriately. Since the only information that is relevant here is what networks an agent considers possible, the interpretation of $some\_new\_info$ is in the spirit of Definition 2.2.5. The truth of $some\_new\_info$ for player i depends only on i's local state. If i's local state is just the sequence of length 1 consisting of i's initial information, then $\pi(g, i, some\_new\_info) = \textit{true}$. If i's local state ℓ in g has length greater than 1, then let ℓ' be the prefix of ℓ which has all but the last element in the sequence. (That is, ℓ' is i's local state just before it was ℓ.) Intuitively, $some\_new\_info$ is true if i learned something in the transition from ℓ' to ℓ. More precisely, let $\mathrm{inf}_i(\ell)$ consist of all the pairs (N', i') that i considers possible in local state ℓ, where i considers (N', i') possible if there exists a global state g' such that (a) N' is the network encoded in the environment state of g' and (b) the local state of i' in g' is ℓ. Then $\pi(g, i, some\_new\_info) = \textit{true}$ if $\mathrm{inf}_i(\ell') \neq \mathrm{inf}_i(\ell)$. (Note that $\mathrm{inf}_i(\ell) \subseteq \mathrm{inf}_i(\ell')$; i does not consider more networks possible after receiving messages; the only question is whether $\mathrm{inf}_i(\ell)$ is a strict subset of $\mathrm{inf}_i(\ell')$.)

Given an interpreted context $\eta = (\gamma, \pi)$ for global function computation, let $\mathcal{N}(\eta)$ denote the set of all networks encoded in the initial global states of γ.

We say that *agents eventually know f with program Pg in interpreted context η* if, in all runs r in $\mathbf{I}(Pg, \eta)$, there exists a time m such that, for all agents $i \in \mathcal{A}(r)$, i knows the value of f at (r, m) in the sense of Definition 2.2.5. Using the notation of this section, this means that there exists a v such that for all $i \in \mathcal{A}(r)$, and all $(N', i') \in \mathrm{inf}_i(r_i(m))$,

$f(N') = v$. Clearly, if all agents eventually know f with program Pg in η, then we can use a trivial modification of Pg to compute f in $\mathcal{N}(\eta)$: the agents simply output f when they know its value.

2.3.3 Using Pg^{GC} for global function computation

The following result shows that we can use Pg^{GC} for global function computation, if anything can be used at all.

Theorem 2.3.1: *If η is an interpreted context for global function computation, then f is computable in $\mathcal{N}(\eta)$ iff all agents eventually know f with Pg^{GC} in η.*

Proof: Clearly if all agents eventually know f with Pg^{GC} in η, then f is computable in $\mathcal{N}(\eta)$, where each agent follows the protocol P used in Theorem 2.2.6 (which acts like Pg^{GC} up to the point where some agent learns the value of f, and then outputs the value and forwards it). Conversely, if f is computable in $\mathcal{N}(\eta)$, then it is knowable. The proof of Theorem 2.2.6 shows that the protocol P computes f in $\mathcal{N}(\eta)$. We leave it to the reader to check that the proof also shows that all agents eventually know f with Pg^{GC} in η. ∎

2.4 Knowledge-based programs with shared names

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.

2.4.1 Syntax

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". We allow quantification over names; for a formula φ with a free variable $\mathbf{n}$, we interpret $\forall \mathbf{n}.\ \varphi$ as saying that $\varphi\,[\mathbf{n}/\mathbf{n}']$ (i.e., φ will all occurences of $\mathbf{n}$ replaced by $\mathbf{n}'$) is true for all names $\mathbf{n}' \in \mathbf{N}$. We also have formulas of the form $\mathit{Calls}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$, where $\mathbf{n}_1$, $\mathbf{n}_2$, and $\mathbf{n}_3$ are names. When interpreted relative to an agent i, $\mathit{Calls}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ holds if the agent that i calls $\mathbf{n}_1$ gives the agent that i calls $\mathbf{n}_2$ the name $\mathbf{n}_3$; that is, it holds relative to i if there exist agents j and k such that i calls j $\mathbf{n}_1$, i calls k $\mathbf{n}_2$, and j calls k $\mathbf{n}_3$. For example, if $\mathcal{N}$ consists only of directed rings, $\mathit{Calls}(L, I, R)$, when interpreted with respect to agent i, says that, according to i, i is the right neighbor of his left neighbor. Finally, we define modal operators $\mathit{Acc}_{\mathbf{n}}$, one for each name $\mathbf{n}$; intuitively, $\mathit{Acc}_{\mathbf{n}}\varphi$ says that φ is true when interpreted relative to the agent called $\mathbf{n}$. For instance, $\mathit{Acc}_{\mathbf{n}}(\mathit{input} = 10)$, interpreted relative to agent i, says that, if j is the agent i calls $\mathbf{n}$, then j's input is 10. Notice that this is different from saying that i's input is 10; that is, when interpreting a formula $\mathit{Acc}_{\mathbf{n}}\varphi$ with respect to some agent i, φ is interpreted relative the agent i calls $\mathbf{n}$, and not relative to i. The need for these syntactic constructs will become clearer in the next section, when we see how they are used in definition of a knowledge-based variant of Pg^{GC}.

A knowledge-based program Pg_{kb} has the form

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

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).

Recall from Section 2.3.1 that $some\_new\_info$ is a proposition taken to be true with respect to an agent i whenever i has learned some new information about the network. We take new_info to be a proposition describing the new information. For example, suppose that N is a directed ring, and agent i learns that his left neighbor has input value v_1. Then new_info is true of (N', i') if the left neighbor of i' in the ring N' has value v_1. As we said earlier, we do not specify here the language used to characterize i's information, since this will depend on the underlying set of networks. We simply assume that the language is expressive enough to characterize all the relevant facts.

In analogy with Pg^{GC}, it seems that the following kb program should solve the global function computation problem, while decreasing the number of messages:

$$\begin{gathered}\textbf{if } some\_new\_info \textbf{ then } \; send_A(new\_info),\\ \textbf{where } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg K_I K_{\mathbf{n}}(new\_info)\}.\end{gathered} \tag{2.1}$$

While this is essentially true, there are 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.

2.4.2 Semantics

We can use the machinery that we have developed to give semantics to formulas such as $K_{\mathbf{n}}\varphi$. As we said before, the formula $K_{\mathbf{n}}\varphi$ must be interpreted relative to the agent making the statement; this means that we interpret $K_{\mathbf{n}}\varphi$ with respect to a point (r, m) and an agent i in r. We call a tuple (r, m, i) a *situation*. If φ is a standard test (i.e., one that does not involve knowledge operators), we take $(\mathcal{I}, r, m, i) \models \varphi$ precisely when $\pi(r(m), i, \varphi) =$ *true*; we define the truth of a conjunction and negation in the standard way. Intuitively, if the agent that i names $\mathbf{n}$ is j, then "$\mathbf{n}$ knows φ" is true relative to i if φ holds in all situations j considers possible, that is, in all situations where j has the same local state as in (r, m). Since j may be uncertain about his own identity, these are exactly the situations (r', m', j') such that j has the same local state in (r, m) as j' in (r', m'). Thus,

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

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 $\mathit{left\_input}$ 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 [37], 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 $\mathit{left\_input}$ 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). We do this as well. Nevertheless, in a formula such as $K_I K_{\mathbf{n}}\ \mathit{new\_info}$, we want to interpret $\mathit{new\_info}$ 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 our intended interpretation, we need the modal operator $\mathit{Acc}_{\mathbf{n}}$, the formulas $\mathit{Calls}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ and quantification over names. Recall that $\mathit{Acc}_{\mathbf{n}}\varphi$ says that φ is true when interpreted relative to the agent i names $\mathbf{n}$:

$$(\mathcal{I}, r, m, i) \models \mathit{Acc}_{\mathbf{n}}\varphi \text{ iff } (\mathcal{I}, r, m, j) \models \varphi, \text{ where } \mu(r(m), i, \mathbf{n}) = j.$$

The semantics of quantification is straightforward:

$$(\mathcal{I}, r, m, i) \models \forall \mathbf{n}.\, \varphi \text{ iff, for all } \mathbf{n}' \in \mathbf{N},\ (\mathcal{I}, r, m, i) \models \varphi\,[\mathbf{n}/\mathbf{n}'],$$

where $\varphi\,[\mathbf{n}/\mathbf{n}']$ is, φ with all occurences of $\mathbf{n}$ replaced by $\mathbf{n}'$. Finally, recall that $\mathit{Calls}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3)$ is intended to be true relative to agent i if the agent j that i calls $\mathbf{n}_1$ calls the agent that i calls $\mathbf{n}_2$ $\mathbf{n}_3$. Thus,

$$(\mathcal{I}, r, m, i) \models \mathit{Calls}(\mathbf{n}_1, \mathbf{n}_2, \mathbf{n}_3) \text{ iff, if } \mu(r(m), i, \mathbf{n}_1) = j \text{ and } \mu(r(m), i, \mathbf{n}_2) = k,$$
$$\text{then } \mu(r(m), j, \mathbf{n}_3) = k.$$

With these constructs, we can express statements of the form "agent $\mathbf{n}$ knows the content of my new information" as "if the agent I name $\mathbf{n}$ gives me name $\mathbf{n}'$, then $\mathbf{n}$ knows that $\mathbf{n}'$ knows $\mathit{new\_info}$", that is, using the formula $\forall \mathbf{n}'.\ \mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \Rightarrow K_{\mathbf{n}}(\mathit{Acc}_{\mathbf{n}'}\mathit{new\_info})$.

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 $(\mathit{Pg}^{kb})_i^{\mathcal{I}}$. 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 $(\mathit{Pg}^{kb})_i^{\mathcal{I}}(\ell)$ 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 $(Pg^{kb})_i^{\mathcal{I}}(\ell)$ performs the null action (which leaves the state unchanged).

A joint protocol P is said to *implement* Pg^{kb} in the 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 $P_i = (Pg^{kb})_i^{\mathbf{I}(P,\gamma,\pi)}$. Here we are 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 [43] (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 $r \in \mathbf{R}(P, \gamma)$. If $P \approx_\gamma P'$, then it easily follows that $\mathbf{R}(P, \gamma) = \mathbf{R}(P', \gamma)$. (Proof: Suppose that $r \in \mathbf{R}(P, \gamma)$. By the definition of $\mathbf{R}(P, \gamma)$, this means that $r(0)$ is an initial state in γ, and for all times m, there exists a joint action act such that $\mathsf{act}_e \in P_e(r_e(m))$, $\mathsf{act}_i \in P_i(r_i(m))$ for all $i \in \mathcal{A}(r)$, and $r(m+1)$ is the result of applying the joint action act to $r(m)$. It easily follows by induction on m, using the assumption that $P \approx_\gamma P'$, that $r(m)$ is a global state in $\mathbf{R}(P', \gamma)$ and that $\mathsf{act}_i \in P'_i(r_i(m))$ for all $i \in \mathcal{A}(r)$ and for all m. Thus, $r \in \mathbf{R}(P', \gamma)$. The same argument shows that if $r' \in \mathbf{R}(P', \gamma)$, then $r' \in \mathbf{R}(P, \gamma)$.) P *de facto implements* Pg^{kb} in context γ if $P \approx_\gamma Pg^{kb^{\mathbf{I}(P,\gamma,\pi)}}$. 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 $\eta = (\gamma, \pi)$ if, for all protocols P that de facto implement Pg_{kb} in γ and all runs r in $\mathcal{R}(P, \gamma)$, eventually all agents in $\mathcal{A}(r)$ know the value $f(N_r)$. We can show that the following variant of the kb program (2.1) solves the global function computation problem for all functions f in all interpreted contexts η such that f is computable in $\mathcal{N}(\eta)$:

$$\begin{array}{l} \textbf{if } \mathit{some\_new\_info} \textbf{ then } \mathit{send}_A(\mathit{new\_info}), \\ \quad \textbf{where } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg K_I(\forall \mathbf{n}'.\ \mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \Rightarrow K_{\mathbf{n}}(\mathit{Acc}_{\mathbf{n}'} \mathit{new\_info}))\}. \end{array} \tag{2.2}$$

However, 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.

2.5 Avoiding redundant communication with counterfactual tests

2.5.1 The general approach

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 temporal $\Diamond$ operator to the kb program (2.2) to get the following program, which we call $Pg^{kb}_{\diamond}$:

$$\textbf{if } \mathit{some\_new\_info} \ \textbf{then } \mathit{send}_A(\mathit{new\_info}),$$
$$\textbf{where } A = \{\mathbf{n} \in \mathbf{Nbr} : \neg K_I \Diamond (\forall \mathbf{n}'.\ \mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \Rightarrow K_{\mathbf{n}}(\mathit{Acc}_{\mathbf{n}'} \mathit{new\_info}))\}.$$

Unfortunately, this modification will not work: as observed by HM, $Pg^{kb}_{\diamond}$ has no implementation in the context γ. For suppose that there exists a protocol P that implements $Pg^{kb}_{\diamond}$. Let $\mathcal{I} = \mathbf{I}(P, \gamma, \pi)$. Does i (the agent represented by I) send $\mathit{new\_info}$ to $\mathbf{n}$ in $\mathcal{I}$? If i sends his new information to $\mathbf{n}$ at time m in a run r of $\mathcal{I}$, then, since communication is reliable, eventually $\mathbf{n}$ will know i's new information and i knows that this is the case, i.e., $(\mathcal{I}, r, m, i) \models K_I \Diamond (\forall \mathbf{n}'.\ \mathit{Calls}(\mathbf{n}, I, \mathbf{n}') \Rightarrow K_{\mathbf{n}}(\mathit{Acc}_{\mathbf{n}'} \mathit{new\_info}))$. Since P implements $Pg^{kb}_{\diamond}$ and $\mathcal{I} = \mathbf{I}(P, \gamma, \pi)$, it follows that i does not send his 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, [57, 80]), 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 his 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 his new information to $\mathbf{n}$ are such that, in r', all agents use the same protocol as in r, except that, at time m in r', i sends his new information to all agents to which he sends his 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). Once we allow the system to include runs that are not runs of the program, it becomes necessary to distinguish the runs of the program from the others. Following HM, we do this by associating to each run a *rank*, which is a natural number. Intuitively, the higher the rank, the less likely the run. The runs of minimal rank 0 are exactly the runs of the program. This gives us a way of distinguishing the runs of the program from others.

In the setting where the system consisted only of runs of the program, agents *knew* all properties of the program (since they were true in all runs in the system). 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 a 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 he 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. This allows us to reason about the program by introducing a new family of modal operators $B_{\mathbf{n}}$. Intuitively, $B_{\mathbf{n}}\varphi$ ("the agent named $\mathbf{n}$ believes φ") is true if φ holds at all points in runs of minimal rank (i.e., in all runs of the program of interest) that the agent named $\mathbf{n}$ considers possible. We provide the formal semantics of belief and counterfactuals, which is somewhat technical, in Section 2.5.2; we hope that the intuitions we have provided will suffice for understanding what follows.

Using counterfactuals, we can modify $Pg_{\diamond}^{kb}$ 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 $\mathbf{n}$, and the formulas $\exists v.\, B_{\mathbf{n}}(f = v)$, which intuitively say that the agent with name $\mathbf{n}$ knows the value of f. (We give formal semantics to $do(send_{\mathbf{n}}(new\_info))$ and $\exists v.\, B_{\mathbf{n}}(f = v)$ in Section 2.5.2.) Let Pg^{GC}_{cb} denote the following modification of Pg^{GC}:

```
if $some\_new\_info$ then $send_A(new\_info)$,
  where $A = \{\mathbf{n} \in \mathbf{Nbr}:$
    $\neg B_I[\neg do(send_{\mathbf{n}}(new\_info)) >$
        $\Diamond((\forall \mathbf{n}'.\, Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'}\, new\_info)) \vee (\exists v.\, B_{\mathbf{n}}(f = v)))]\}.$
```

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 even if the message is not sent. In particular, if the recipient is bound to evnetually know the value of f, the value is not sent. As we will show in Section 2.5.3, this improved program still solves the global function computation problem whenever possible.

Theorem 2.5.1: *If η is an interpreted context for global function computation and f is computable in $\mathcal{N}(\eta)$, then Pg^{GC}_{cb} solves the global function computation problem for f in η.*

2.5.2 Counterfactual belief-based programs with names

The standard approach to giving semantics to counterfactuals [57, 80] 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 to deal with names. We just briefly review the relevant details here; we encourage the reader to consult [43] for more details and intuition. Define an *order assignment* for an extended interpreted system $\mathcal{I} = (\mathcal{R}, \pi, \mu)$ to be a function $\ll$ that associates with every situation (r, m, i) a partial order relation $\ll_{(r,m,i)}$ over situations. The partial orders must satisfy the constraint that (r, m, i) is a minimal element of $\ll_{(r,m,i)}$, so that there is no situation (r', m', i') such that $(r', m', i') \ll_{(r,m,i)} (r, m, i)$. Intuitively, $(r_1, m_1, i_1) \ll_{(r,m,i)} (r_2, m_2, i_2)$ 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 $\mathcal{J} = (\mathcal{I}, \ll)$, where $\mathcal{I}$ is an extended interpreted system and $\ll$ is an order assignment for the situations in $\mathcal{I}$.

Given a counterfactual system $\mathcal{J} = (\mathcal{I}, \ll)$, 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 $\texttt{closest}(A, r, m, i)$, by taking

$$\begin{aligned}&\texttt{closest}(A, r, m, i) =\\ &\qquad \{(r', m', i') \in A : \quad \text{there is no situation } (r'', m'', i'') \in A \text{ such that}\\ &\qquad\qquad\qquad\qquad\qquad (r'', m'', i'') \ll_{(r,m,i)} (r', m', i')\}.\end{aligned}$$

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

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

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 [43], 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^{GC}_{cb}, which have an antecedent $\neg do(send_{\mathbf{n}}(msg))$, get interpreted appropriately.[4] HM defined a way of doing so for counterfactual tests whose antecedent has the form $do(i,\mathsf{act})$. In our programs, the counterfactual tests have antecedents of the form $\neg do(i, send_{\mathbf{n}}(msg))$. To deal with these formulas, we modify the HM construction as follows: Given a context γ, situation (r,m,i) in $\mathcal{R}^+(\gamma)$, and a deterministic protocol P,[5] 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) $r'(m') = r(m')$ and $r'_{i'}(m') = r_i(m')$ for all $m' \leq m$, (c) if $P_i(r_i(m)) = \mathsf{act}$ and either $\mathsf{act} = send_A(msg')$ and $\mathbf{n} \notin A$ or $msg \neq msg'$ or act does

[4]$(\mathcal{J},r,m,i) \models do(a)$ if $r_i(m+1)$ is the result (according to the transition function in the context) of agent i taking action a in $r_i(m)$.

[5]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.

not involve sending a message, then $r' = r$ and $i = i'$, (d) if $P_i(r_i(m)) = send_A(msg)$, and $\mathbf{n} \in A$, then i' performs $send_{A-\{\mathbf{n}\}}(msg)$ in local state $r_i(m) = r'_{i'}(m)$ 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 $r'\}$. Condition (c) says that if i is not supposed to send $\mathbf{n}$ the message msg according to P_i, then the closest point to (r, m) according to i where the message is not sent is just (r, m) itself. Condition (d) says that if i is supposed to send $\mathbf{n}$ the message msg according to P_i, then in the closest points where it does not send this message to $\mathbf{n}$, the agent i' (who i considers it possible that he might be) still sends the message to everyone else he was supposed to send the message to according P_i. 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 $\ll^P = o(P)$ on the situations of $\mathcal{R}^+(\gamma)$. We are interested in order generators that prefer runs in 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 $\mathtt{act} = send_A(msg)$, $\mathtt{closest}([\![\neg send_A(msg)]\!]_{\mathbf{I}(P,\zeta)}, r, m, i)$ 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. [33]).

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

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

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 $\mathcal{J} = (\mathcal{I}, \ll, \kappa)$, where $(\mathcal{I}, \ll)$ 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

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

Thus, whereas we earlier considered systems consisting of all (and only) the runs of the program of interest, now we have larger systems, but can recover the runs of the program of interest by considering runs of rank 0. In the original setting, agents know all properties of the program of interest (and their consequences); now agents *believe* that these properties hold.

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 2.5.2: [43] *Suppose that $\mathcal{J} = (\mathcal{R}, \pi, \mu, \lll, \kappa)$ 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 formulas of the form $\exists v.\, B_{\mathbf{n}}(f = v)$. Recall that the value of f in run r is $f(N_r)$. Intuitively, $\exists v. B_{\mathbf{n}}(f = v)$ is true at a point (r, m) if at all runs $\mathbf{n}$ believes possible, f has the same value.

$(\mathcal{I}, \lll, \kappa, r, m, i) \models \exists v.\, B_{\mathbf{n}}(f = v)$ iff, if $\mu((r, m), i, \mathbf{n},) = j$, then for all r', m', if $r_j(m) = r'_{j'}(m')$ and $\kappa(r') = \mathsf{min}^{\kappa}_{j}(r, m)$, then $f(N_r) = f(N_{r'})$.

With all these definitions in hand, we can define the semantics of counterfactual belief-based programs such as Pg^{GC}_{cb}. 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 $\mathcal{J} = (\mathcal{I}, \lll, \kappa)$, an agent i, and a cbb program Pg_{cb}, let $(\mathsf{Pg}_{cb})^{\mathcal{J}}_{i}$ 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 $\kappa(r') = \mathsf{min}^{\kappa}_{j}(r, m)$. 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 $\mathcal{J} = (\mathcal{I}, \lll, \kappa)$

represents the cbb program Pg_{cb} in cb context (γ, π, o, σ) if (a) $\mathcal{I} = (\mathcal{R}^+(\gamma), \pi, \mu_\gamma)$, (b) $\ll\, = o(\mathsf{Pg}_{cb}^{\mathcal{J}})$, 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 $P \approx_\gamma \mathsf{Pg}_{cb}^{(\mathcal{I}, o(P), \sigma(P))}$.

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

Theorem 2.5.1: *If η is an interpreted context for global function computation and f is computable in $\mathcal{N}(\eta)$, then Pg_{cb}^{GC} solves the global function computation problem for f in η.*

Proof: Suppose that f is computable in $N(\eta)$. 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 2.3.1; 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 $send_{\mathsf{A}}(new\_info)$ action with $\mathbf{n} \in A$, i will eventually learn his new information or the function value anyway, or (b) j does not believe this. In case (b), it follows that

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models\ & \neg B_I[\neg do(send_{\mathbf{n}}(new\_info)) > \\ & \Diamond(\forall \mathbf{n}'.\ Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'} new\_info)) \vee (\exists v.\ B_{\mathbf{n}}(f = v))].\end{aligned}$$

Since P implements Pg^{GC}_{cb} in χ^{GC}, in case (b), j sends i 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

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models\ & B_I[\neg do(send_{\mathbf{n}}(new\_info)) > \\ & \Diamond((\forall \mathbf{n}'.\ Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'} new\_info)) \vee (\exists v.\ B_{\mathbf{n}}(f = v)))].\end{aligned}$$

Since σ is deviation compatible by assumption, and r is a run of P, it follows that $\kappa(r) = 0$. Thus by Lemma 2.5.2,

$$\begin{aligned}(\mathcal{J}, r, m'_k, j) \models\ & \neg do(send_{\mathbf{n}}(new\_info)) > \\ & \Diamond((\forall \mathbf{n}'.\ Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'} new\_info)) \vee (\exists v.\ B_{\mathbf{n}}(f = v))).\end{aligned}$$

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

$$(\mathcal{J}, r, m'_k, j) \models (\forall \mathbf{n}'.\ (Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'} new\_info))) \vee (\exists v.\ B_{\mathbf{n}}(f = v)).$$

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

$$(\mathcal{J}, r, m'_k, j) \models \forall \mathbf{n}'.\ (Calls(\mathbf{n}, I, \mathbf{n}') \Rightarrow B_{\mathbf{n}}(Acc_{\mathbf{n}'} new\_info)).$$

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 2.3.1, that some agent learns $f(N_r)$ in r, and that eventually all agents learn it. We omit details here. ▮

2.6 Case study: 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 2.2.7 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 an optimal flooding protocol [60], a protocol combining ideas of Le Lann [56] and Chang and Roberts [17] (LCR from now on) presented by Lynch [61], which works in directed rings, and Peterson's [77] protocol P1 for directed rings and P2 for undirected rings. We briefly sketch the LCR protocol, and Peterson's protocols P1 and P2, following Lynch's [61] treatment.

The optimal flooding protocol takes a parameter d (intuitively, an upper bound on the network diameter), and proceeds in rounds. Agents keep track of the maximum id they have seen. Initially agents send their id to their neighbors, and, in each round, if they have heard of a value larger than the current maximum, they forward it on all outgoing links. After d rounds, the agent whose id is the maximum he has seen declares himself leader. It is easy to see that this protocol is correct in all networks whose diameter is bounded by d, since by round d, all agents will have heard about the maximum id in the network, and will know that the leader is the agent whose id is the maximum.

The LCR protocol works in directed rings, and does not assume a bound on their size. Each agent starts by sending his 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 he receives his own id. If this id is M, the agent then sends the message "the agent with id M is the leader" to his 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 his own id back. That agent then knows that his id is larger than that of any other agent, and thus becomes the leader.

Peterson's protocol P2 for undirected 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 his id with the ids of the closest active agent to his right and to his left. (We assume that agents keep track of message received, and that ids are numbers, so that any two ids can be compared.) If his 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 his own id, it declares himself leader. Then if his 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 his 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 directional 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. Like P2, P1 operates in phases, and agents are either active or passive; passive agents just forward messages. With P1, in addition to their actual id, agents also maintain a "temporary" id. Initially, the temporary id of a process is his id. But at the end of phase $p > 0$, an active process's temporary id becomes the temporary id in phase $p - 1$ of the closest active process to his left. While with P2 agents compare the ids they receive with their own id in order to decide whether they stay active or not, with P1 they compare the ids they receive to their temporary ids. The rule with P1 is that an agent stays active if his temporary id is larger than the temporary ids of the following and preceding active agents in the ring. Since the ring is directed, the way to discover if this is the case is for an active agent to forward his 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, he remains active, and takes as his 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 P2 iff his id is active in P1 (i.e., iff his id is the temporary id of an active agent in P1). When an agent receives his original value, then he declares himself 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 his 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 his 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 his left neighbor has this value. We can easily deal with this by modifying LCR, P1, and P2 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. To see this, let LCR$^{\textit{fullinfo}}$ be the result of modifying LCR so that agents send all the newn information they have learned, rather than just maximum ids. Suppose that j is the agent with highest id. When j receives the message with his id back and sends it around the ring again in LCR$^{\textit{fullinfo}}$ (this is essentially the message saying that j is the leader), j's second message will include the id j' of the agent just before j. Thus, when j' receives j's second message, it will not need to forward it to j. Let LCR$'$ be the result of modifying LCR$^{\textit{fullinfo}}$ so that the last message is not sent. 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 agent 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 Chapter 6.

In the following, we say that a protocol P' *agrees with* $P2$ *up to the last phase* if there exist functions $F1$ from runs of P2 to runs of P' and $F2$ from runs of P' to runs of

$P2$ such that, for every run r of P2 (resp., run r' of P') and every agent i and j, i sends a message to j at the point (r, m) and i does not yet know the maximum id iff i sends a message to j at the point $(F_1(r), m)$ (resp., $(F_2(r'), m)$) and i does not yet know the maximum id. (In the case of P2, i knows the maximum id iff i receives its own id or receives a message with the value of the maximum id.) Notice that the messages sent at (r, m) and $(F_1(r), m)$ might, in general be different. Indeed, whereas in P1 i may send j an id, in $F(r)$, i may send a potentially longer message containing all the new information it has learned. We can similarly define what it means for P' to agree with P1 up to the last phase.

If $\mathcal{N}$ is a set of graphs where each node in N has an associate id, then let $\mathit{fmaxid}(N)$ be the maximum id in N, for $N \in \mathcal{N}$.

Theorem 2.6.1: *The following all hold:*

(a) The optimal flooding protocol with parameter d de facto implements Pg^{GC}_{cb} (with $f = \mathit{fmaxid}$) in contexts where (i) all networks have diameter at most d and (ii) all agents have distinct identifiers.

(b) LCR′ de facto implements Pg^{GC}_{cb} (with $f = \mathit{fmaxid}$) in all contexts where (i) all networks are directed rings and (ii) agents have distinct identifiers.

(c) There exists a protocol P1′ that agrees with P1 up to the last phase and implements Pg^{GC}_{cb} (with $f = \mathit{fmaxid}$) in all contexts where (i) all networks are directed rings and (ii) agents have distinct identifiers.

(d) There exists a protocol P2′ that agrees with P2 up to the last phase and de facto implements Pg^{GC}_{cb} (with $f = \mathit{fmaxid}$) in all contexts where (i) all networks are undirected rings and (ii) agents have distinct identifiers.

Theorem 2.6.1 points out a significant commonality among these protocols. Moreover, the process of modifying $P1$ and $P2$ to obtain programs that implement Pg_{cb}^{GC} (and thus ensure that agents send messages only when they believe it is necessary to do so) suggests a general approach for finding message-optimal protocols: starting with a high-level knowledge-based program that uses counterfactuals to ensure that messages are not sent unnecessarily, and then implementing it with a standard program. 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. Indeed, our hope is that we will be able to synthesize standard programs by starting with high-level knowledge-based specifications, synthesizing a knowledge-based program that satisfies the specification, and then instantiating the knowledge-based program as a standard program. This makes the subject of the next chapter.

CHAPTER 3

PROGRAM SYNTHESIS FROM KNOWLEDGE-BASED SPECIFICATIONS

Synthesis is a methodology of producing programs from specifications. With *correct-by-construction* program synthesis [9] programs are constructed from *proofs* that the specifications are satisfiable. That is, a constructive proof that a specification is satisfiable gives a program that satisfies the specification. Correct-by-construction program synthesis has been successfully used to construct large complex sequential programs, but it has not yet been used to create substantial realistic distributed programs.

The Cornell Nuprl proof development system was among the first tools used to produce correct-by-construction functional and sequential programs [19]. Nuprl has also been used extensively to optimize distributed protocols, and to specify them in the language of I/O Automata [60]. Recent work [11] has resulted in the definition of a fragment of the higher-order logic used by Nuprl tailored to specifying distributed protocols, called *event theory*, and the extension of Nuprl methods to synthesize distributed protocols from specifications written in event theory. However, as has long been recognized [25], designers typically think of specifications at a high level, which often involves knowledge-based statements.

In this chapter, we add knowledge operators to event theory raising its level of abstraction and show by example that knowledge-based programs can be synthesized from constructive proofs that specifications in event theory with knowledge operators are satisfiable. Our example uses the *sequence-transmission problem* (STP from now on), where a sender must transmit a sequence of bits to a receiver in such a way that the receiver eventually knows arbitrarily long prefixes of the sequence. Halpern and Zuck [50] provide knowledge-based programs for STP, prove them correct, and show that a number of standard programs proposed for solving STP implement them. Here we show that one of these knowledge-based programs can be synthesized from the specifications of the problem, expressed in event theory augmented by knowledge. We can then translate the arguments of Halpern and Zuck to Nuprl, to show that the knowledge-based program can be transformed to the standard programs in the literature.

This chapter is organized as follows. In the next section we give a brief overview of the Nuprl system, review event theory, discuss the type of programs we use (distributed message automata), and show how automata can be synthesized from a specification. In Section 3.2 we show how epistemic logic can be translated into Nuprl, and how to formalize knowledge-based automata in Nuprl. The sequence transmission problem is analyzed in Section 3.3. We conclude with references to related work in Section 3.4.

3.1 Synthesizing programs from constructive proofs

3.1.1 Nuprl: a brief overview

Much current work on formal verification using theorem proving, including Nuprl, is based on type theory (see [18] for a recent overview). A type can be thought of as a set with structure that facilitates its use as a data type in computation; this structure also supports constructive reasoning. The set of types is closed under constructors such as $\times$ and $\to$, so that if A and B are types, so are $A \times B$ and $A \to B$, where, intuitively, $A \to B$ represents the computable functions from A into B.

Constructive type theory, on which Nuprl is based, was developed to provide a foundation for constructive mathematics. The key feature of constructive mathematics is that "there exists" is interpreted as "we can construct (a proof of)". Reasoning in the Nuprl type theory is intuitionistic [14], in the sense that proving a certain fact is understood as constructing evidence for that fact. For example, a proof of the fact that "there exists x of type A" builds an object of type A, and a proof of the fact "for any object x of type A there exists an object y of type B such that the relation $R(x, y)$ holds" builds a function f that associates with each object a of type A an object b of type B such that $R(a, b)$ holds.

One consequence of this approach is that the principle of excluded middle does not apply: while in classical logic, $\varphi \vee \neg\varphi$ holds for all formulas φ, in constructive type theory, it holds exactly when we have evidence for either φ or $\neg\varphi$, and we can tell from this evidence which of φ and $\neg\varphi$ it supports. A predicate $Determinate$ is definable in Nuprl such that $Determinate(\varphi)$ is true iff the principle of excluded middle holds for formula φ. (From here on in, when we say that a formula is true, we mean that it is constructively true, that is, provable in Nuprl.)

In this chapter, we focus on synthesizing programs from specifications. Thus we must formalize these notions in Nuprl. As a first step, we define a type Pgm in Nuprl and take programs to be objects of type Pgm. Once we have defined Pgm, we can define other types of interest.

Definition 3.1.1: A *program semantics* is a function S of type $Pgm \rightarrow Sem$ assigning to each *program* Pg of type Pgm a *meaning* of type $Sem = 2^{Sem'}$. Sem' is the type of *executions* consistent with the program Pgm under the semantics S. A *specification* is a predicate X on Sem'. A program Pg *satisfies* the specification X if $X(e)$ holds for all e in $S(Pg)$. A specification X is *satisfiable* if there exists a program that satisfies X. ▮

As Definition 3.1.1 suggests, all objects in Nuprl are typed. To simplify our discussion, we typically suppress the type declarations. Definition 3.1.1 shows that the satisfiability of a specification is definable in Nuprl. The key point for the purposes of this chapter is that from a constructive proof that X is satisfiable, we can extract a program that satisfies X.

Constructive type logic is highly undecidable, so we cannot hope to construct a proof completely automatically. However, experience has shown that, by having a large library of lemmas and proof tactics, it is possible to "almost" automate quite a few proofs, so that with a few hints from the programmer, correctness can be proved. For this general constructive framework to be useful in practice, the parameters Pgm, Sem', and S must be chosen so that (a) programs are concrete enough to be compiled, (b) specifications are naturally expressed as predicates over Sem', and (c) there is a small set of *rules* for producing proofs of satisfiability.

To use this general framework for synthesis of *distributed*, *asynchronous* algorithms, we choose the programs in Pgm to be *distributed message automata*. Message automata are closely related to *IO-Automata* [60] and are roughly equivalent to *UNITY* programs [16] (but with message-passing rather than shared-variable communication). We describe distributed message automata in Section 3.1.3. As we shall see, they satisfy criterion (a).

The semantics of a program is the *system*, or set of *runs*, consistent with it. Typical specifications in the literature are predicates on runs. We can view a specification as a predicate on systems by saying that a system satisfies a specification exactly if all the runs in the system satisfy it. To satisfy criterion (b), we formalize runs as structures that we call *event structures*, much in the spirit of Lamport's [55] model of events in distributed systems. Event structures are explained in more detail in the next section. We have shown [11] that, although satisfiability is undecidable, there is indeed a small set of rules from which we can prove satisfiability in many cases of interest; these rules are discussed in Section 3.1.3.

3.1.2 Event structures

Consider a set AG of processes or *agents*; associated with each agent i in AG is a set X_i of *local variables*. Agent i's local state at a point in time is defined as the values of its local variables at that time. We assume that the sets of local variables of different agents are disjoint. Information is communicated by message passing. The set of links is $Links$. Sending a message on some link $l \in Links$ is understood as enqueuing the message on l, while receiving a message corresponds to dequeuing the message. Communication is point-to-point: for each link l there is a unique agent $source(l)$ that can send messages on l, and a unique agent $dest(l)$ that can receive message on l. For each agent i and link l with $source(l) = i$, we assume that $msg(l)$ is a local variable in X_i.

We assume that communication is asynchronous, so there is no global notion of time. Following Lamport [55], changes to the local state of an agent are modeled as *events*. Intuitively, when an event "happens", an agent either sends a message, receives a message or chooses some values (perhaps nondeterministically). As a result of receiving the message or the (nondeterministic) choice, some of the agent's local variables are changed.

Lamport's theory of events is the starting point of our formalism. To help in writing concrete and detailed specifications, we add more structure to events. Formally, an event is a tuple with three components. The first component of an event e is an agent $i \in AG$, intuitively the agent whose local state changes during event e. We denote i as $agent(e)$. The second component of e is its *kind*, which is either a link l with $dest(l) = i$ or a local action a, an element of some given set Act of local actions. The only actions in

Act are those that set local variables to certain values. We denote this component as $kind(e)$. We often write $kind(e) = rcv(l)$ rather than $kind(e) = l$ to emphasize the fact that e is a receive event; similarly we write $kind(e) = local(a)$ rather than $kind(e) = a$ to emphasize the fact that a is a local action. The last component of e is its ***value*** v, a tuple of elements in some domain Val; we denote this component as $val(e)$. If e is a receive event, then $val(e)$ is the message received when e occurs; if e is a local event a, then $val(e)$ represents the tuple of values to which the variables are set by a. (For more details on the reasons that led to this formalism, see [12].)

Rather than having a special kind to model send events, we model the sending of a message on link l by changing the value of a local variable $msg(l)$ that describes the message sent on l. A special value $\perp$ indicates that no message is sent when the event occurs; changing $msg(l)$ to a value other than $\perp$ indicates that a message is sent on l. This way of modeling send events has proved to be convenient. One advantage is that we can model multicast: the event e of i broadcasting a message m to a group of agents just involves a local action that sets $msg(l)$ to m for each link l from i to one of the agents in the group. Similarly, there may be an action in which agent i sends a message to some agents and simultaneously updates other local variables.

Following Lamport [55], we model an execution of a distributed program as a sequence of events satisfying a number of natural properties. We call such a sequence an event structure. We take an event structure es to be a tuple consisting of a set E of events and a number of additional elements that we now describe. These elements include the functions $dest$, $source$, and msg referred to above, but there are others. For example, Lamport assumes that every receive event e has a corresponding (and unique) event where the message received at e was sent. To capture this in our setting, we assume that the description of the event structure es includes a function $send$ whose domain is the receive events in es and whose range is the set of events in es; we require that $agent(send(e)) = source(l)$. Note that, since we allow multicasts, different receive events may have the same corresponding send event.

For each $i \in AG$, we assume that the set of events e in es associated with i is totally ordered. This means that, for each event e, we can identify the sequence of events (*history*) associated with agent i that preceded e. To formalize this, we assume that, for each agent $i \in AG$, the description of es includes a total order $\prec_i$ on the events e in es such that $agent(e) = i$. Define a predicate *first* and function *pred* such that $\mathit{first}(e)$ holds exactly when e is the first event in the history associated with $agent(e)$ in es; if $\mathit{first}(e)$ does not hold, then $pred(e)$ is the unique predecessor of e in es. Following Lamport [55], we take $\prec$ to be the least transitive relation on events in es such that $send(e) \prec e$ if e is a receive event and $e \prec e'$ if $e \prec_i e'$. We assume that $\prec$ is well-founded. We abbreviate $(e' \prec e) \vee (e = e')$ as $e' \preceq e$, or $e \succeq e'$. Note that $\prec_i$ is defined only for events associated with agent i: we write $e \prec_i e'$ only if $agent(e) = agent(e') = i$.

The local state of an agent defines the values of all the variables associated with the agent. While it is possible that an event structure contains no events associated with a particular agent, for ease of exposition, we consider only event structures in which each agent has at least one local state, and denote the initial local state of agent i as $initstate_i$. In event structures es where at least one event associated with a given agent i occurs, $initstate_i$ represents i's local state before the first event associated with i occurs in es. Formally, the *local state* of an agent i is a function that maps X_i and a special symbol $\mathbf{val}_i$ to values. (The role of $\mathbf{val}_i$ will be explained when we give the semantics of the logic.) If $x \in X_i$, we write $s(x)$ to denote the value of x in i's similarly, $s(\mathbf{val}_i)$ is the value of $\mathbf{val}_i$ in s. If $agent(e) = i$, we take $state\ before\ e$ to be the local state of agent i before e; similarly, $state\ after\ e$ denotes i's local state after event e occurs.

The value $(\mathit{state\ after}\ e)(x)$ is in general different from $(\mathit{state\ before}\ e)(x)$. How it differs depends on the event e, and will be clarified in the semantics. We assume that $(\mathit{state\ after}\ e)(\mathbf{val}_i) = \mathit{val}(e)$; that is, the value of the special symbol $\mathbf{val}_i$ in a local state is just the value of the event that it follows. If $x \in X_i$, we take $x\ \mathit{before}\ e$ to be an abbreviation for $(\mathit{state\ before}\ e)(x)$; that is, the value of x in the state before e occurs; similarly, $x\ \mathit{after}\ e$ is an abbreviation for $(\mathit{state\ after}\ e)(x)$.

Example 3.1.2: Suppose that Act contains send and $\mathit{send}{+}\mathit{inc}(x)$, where $x \in X_i$, and that Val contains the natural numbers. Let n and v be natural numbers. Then

- the event of agent i receiving message m on link l in the event structure es is modeled by the tuple $e = (i, l, m)$, where $\mathit{agent}(e) = i$, $\mathit{kind}(e) = \mathit{rcv}(l)$, and $\mathit{val}(e) = m$;
- the event of agent i sending message n on link l in es is represented by the tuple $e = (i, \mathit{send}, m)$, where $\mathit{msg}(l)\ \mathit{after}\ e = m$;
- the event e of agent i sending m on link l and incrementing its local variable x by v in es is represented by the tuple e such that $\mathit{agent}(e) = i$, $\mathit{kind}(e) = \mathit{send}{+}\mathit{inc}(x)$, and $\mathit{val}(e) = \langle m, v\rangle$, where $\mathit{msg}(l)\ \mathit{after}\ e = m$ and $x\ \mathit{after}\ e = x\ \mathit{before}\ e + v$.

Definition 3.1.3: An event structure is a tuple $es = \langle AG, \mathit{Links}, \mathit{source}, \mathit{dest}, \mathit{Act}, \{X_i\}_{i \in AG}, \mathit{Val}, \{\mathit{initstate}_i\}_{i \in AG}, E, \mathit{agent}, \mathit{send}, \mathit{first}, \{\prec_i\}_{i \in AG}, \prec\rangle$ where AG is a set of agents, Links is a set of links such that $\mathit{source} : \mathit{Links} \longrightarrow AG$, $\mathit{dest} : \mathit{Links} \longrightarrow AG$, Act is a set of actions, X_i is a set of variables for agent $i \in AG$ such that, for all links $l \in \mathit{Links}$, $\mathit{msg}(l) \in X_i$ if $i = \mathit{source}(l)$, Val is a set of values, $\mathit{initstate}_i$ is the initial local state of agent $i \in AG$, E is a set of events for agents AG, kinds $\mathit{Kind} = \mathit{Links} \cup \mathit{Act}$, and domain Val, functions agent, send and first are defined as explained above, $\prec_i$s are local precedence relations and $\prec$ is a causal order such that the following axioms, all expressible in Nuprl, are satisfied:

- if e has kind $rcv(l)$, then the value of e is the message sent on l during event $send(e)$, $agent(e) = dest(l)$, and $agent(send(e)) = source(l)$:

$$\begin{aligned}&\forall e \in es.\forall l.\ (kind(e) = rcv(l)) \Rightarrow\\ &\quad (val(e) = msg(l)\ after\ send(e)) \wedge (agent(e) = dest(l)) \wedge\\ &\quad (agent(send(e)) = source(l))\end{aligned}$$

- for each agent i, events associated with i are totally ordered:

$$\forall e \in es.\forall e' \in es.(agent(e) = agent(e') = i \Rightarrow e \prec_i e' \vee e' \prec_i e \vee e = e').$$

- e is the first event associated with agent i if and only if there is no event associated with i that precedes e:

$$\forall e \in es\ \forall i.\ (agent(e) = i) \Rightarrow (first(e) \Leftrightarrow \forall e' \in es.\ \neg(e' \prec_i e)).$$

- the initial local state of agent i is the state before the first event associated with i, if any:

$$\forall i.\ (\forall e \in es.\ (agent(e) = i \Rightarrow (first(e) \Leftrightarrow (state\ before\ e = initstate_i)))).$$

- the predecessor of an event e immediately precedes e in the causal order:

$$\begin{aligned}&\forall e \in es.\ \forall i.\ ((agent(e) = i) \wedge \neg first(e)) \Rightarrow\\ &\quad ((pred(e) \prec_i e) \wedge (\forall e' \in es.\ \neg(pred(e) \prec_i e' \prec_i e))).\end{aligned}$$

- the local variables of agent $agent(e)$ do not change value between the predecessor of e and e:

$$\begin{aligned}&\forall e \in es.\ \forall i.\ (agent(e) = i \wedge \neg first(e)) \Rightarrow\\ &\qquad\qquad \forall x \in X_i.\ (x\ after\ pred(e) = x\ before\ e).\end{aligned}$$

- the causal order $\prec$ is well-founded:

$$\forall P.\ (\forall e.\ (\forall e' \prec e.\ P(e')) \Rightarrow P(e)) \Rightarrow (\forall e.\ P(e)),$$

  where P is an arbitrary predicate on events. (It is easy to see that this axiom is sound if $\prec$ is well-founded. On the other hand, if $\prec$ is not well-founded, then let P be a predicate that is false exactly of the events e such that there there is an infinite descending sequence starting with e. In this case, the antecedent of the axiom holds, and the conclusion does not.)

In our proofs, we will need to argue that two events e and e' are either causally related or they are not. It can be shown [11] that this can be proved in constructive logic iff the predicate first satisfies the principle of excluded middle. We enforce this by adding the following axiom to the characterization of event structures:

$$\forall e \in es.\ \mathit{Determinate}(\mathit{first}(e)).$$

The set of event structures is definable in Nuprl (see [11]). We use event structures to model executions of distributed systems. We show how this can be done in the next section.

3.1.3 Distributed message automata

As we said, the programs we consider are *message automata*. Roughly speaking, we can think of message automata as nondeterministic state machines, though certain differences exist. Each basic message automaton is associated with an agent i; a message automaton associated with i essentially says that, if certain preconditions hold, i can take certain local actions. (We view $\mathit{receive}$ actions as being out of the control of the agent, so the only actions governed by message automata are local actions.) At each point in time, i nondeterministically decides which actions to perform, among those whose precondition is satisfied. We next describe the syntax and semantics of message automata.

Syntax

We consider a first-order language for tests in automata. Fix a set AG of agents, a set X_i of local variables for each agent i in AG, and a set X^* of variables that includes $\cup_{i \in AG} X_i$ (but may have other variables as well). The language also includes special constant symbols $\mathbf{val}_i$, one for each agent $i \in AG$, predicate symbols in some finite set $\mathcal{P}$, and function symbols in some finite set $\mathcal{F}$. Loosely speaking, $\mathbf{val}_i$ is used to denote the value of an event associated with agent i; constant symbols other than $\mathbf{val}_1, \ldots, \mathbf{val}_n$ are just 0-ary function symbols in $\mathcal{F}$. We allow quantification only over variables other than local variables; that is, over variables $x \notin \cup_{i \in AG} X_i$.

Message automata are built using a small set of *basic programs*, which may involve formulas in the language above. Fix a set Act of local actions and a set $Links$ of links between agents in AG.[1] There are five types of basic programs for agent i:

- $@i$ $\mathbf{initially}$ ψ;
- $@i$ $\mathbf{if}$ $kind = k$ $\mathbf{then}$ $x := t$, where $k \in Act \cup Links$ and $x \in X_i$;
- $@i$ $kind = local(a)$ $\mathbf{only\ if}$ φ;
- $@i$ $\mathbf{if\ necessarily}$ φ $\mathbf{then\ i.o.}$ $kind = local(a)$; and
- $@i$ $\mathbf{only}$ L $\mathbf{affects}$ x, where L is a list of kinds in $Act \cup Links$ and $x \in X_i$.

[1]We are being a little sloppy here, since we do not distinguish between an action a and the name for the action that appears in a program, and similarly for links and the variables in X_i.

Note that all basic program for agent i are prefixed by $@i$.

We can form more complicated programs from simpler programs by *composition*. We can compose automata associated with different agents. Thus, the set (type) Pgm of programs is the smallest set that includes the basic programs such that if Pg_1 and Pg_2 are programs, then so is $Pg_1 \oplus Pg_2$.[2]

Semantics

We give semantics by associating with each program the set of event structures consistent with the program. Intuitively, a set of event structures is consistent with a distributed message automaton if each event structure in the set can be seen as an execution of the automaton. The semantics can be defined formally in Nuprl as a relation between a distributed program Pg and an event structure es. In this section, we define the consistency relation for programs and give the intuition behind these programs.

In classical logic, we give meaning to formulas using an interpretation. In the Nuprl setting, we are interested in *constructive interpretations* I, which can be characterized by a formula φ_I. We can think of φ_I as characterizing a domain Val_I and the meaning of the fuction and predicate symbols. If I is an interpretation with domain Val_I, an *I-local state for i* maps $X_i \cup \{\mathbf{val}_i\}$ to Val_I; an *I-global state* is a tuple of I-local states, one for each agent in AG. Thus, if $s = (s_1, \ldots, s_n)$ is an I-global state, then s_i is i's local state in s. (Note that we previously used s to denote a local state, while here s denotes a global state. We will always make it clear whether we are referring to local or global states.)

For consistency with our later discussion of knowledge-based programs, we allow

[2]Here we are deliberately ignoring the difference between sets and types.

the meaning of some predicate and function symbols that appear in tests in programs to depend on the global state. We say that a function or predicate symbol is *rigid* if it does not depend on the global state. For example, if the domain is the natural numbers, we will want to treat $+$, $\times$, and $<$ as rigid. However, having the meaning of a function or predicate depend on the global state is not quite as strange as it may seem. For example, we may want to talk about an array whose values are encoded in agent 1's variables x_1, x_2, and x_3. An array is just a function, so the interpretation of the function may change as the values of x_1, x_2, and x_3 change. For each nonrigid predicate symbol P and function symbol f in $\mathcal{P} \cup \mathcal{F}$, we assume that there is a predicate symbol P^+ and function symbol f^+ whose arity is one more than that of P (resp., f); the extra argument that is a global state. We then associate with every formula φ and term t that appears in a program a formula φ^+ and term t^+ in the language of Nuprl. We define φ^+ by induction on the structure of φ. For example, for an atomic formula such as $P(c)$, if P and c are rigid, then $(P(c))^+$ is just $P(c)$. If P and c are both nonrigid, then $(P(c))^+$ is $P^+(c^+(\mathrm{s}), \mathrm{s})$, where s is a variable interpreted as a global state.[3] We leave to the reader the straightforward task of defining φ^+ and t^+ for atomic formulas and terms. We then take $(\varphi \wedge \psi)^+ = \varphi^+ \wedge \psi^+$, $(\neg\varphi)^+ = \neg\varphi^+$, and $(\forall x \varphi)^+ = \forall x \varphi^+$.

An I-*valuation* V associates with each non-local variable (i.e., variable not in $\cup_{i \in AG} X_i$) a value in Val_I. Given an interpretation I, an I-global state s, and an I-valuation V, we take $I_V(\varphi)(s)$ to be an abbreviation for the formula (expressible in Nuprl) that says φ_I together with the conjunction of atomic formulas of the form $x = V(x)$ for all non-local variables x that appear in φ, $x = s_i(x)$ for variables $x \in X_i$, $i \in AG$, that appear in φ, and $\mathrm{s} = s$ implies φ^+. Thus, $I_V(\varphi)(s)$ holds if there is a constructive proof that the formula that characterizes I together with the (atomic) formulas that describe $V(x)$ and s, and a formula that says that s is represented by s, imply φ^+.

[3]Since Nuprl is a higher-order language, there is no problem having a variable ranging over global states that is an argument to a predicate.

It is beyond the scope of this thesis (and not necessary for what we do here) to discuss constructive proofs in Nuprl; details can be found in [20]. However, it is worth noting that, for a first-order formula φ, if $I_V(\varphi)(s)$ holds, then φ^+ is true in state s with respect to the semantics of classical logic in I. The converse is not necessarily true. Roughly speaking, $I_V(\varphi)(s)$ holds if there is evidence for the truth of φ^+ in state s (given valuation V). We may have evidence for neither φ^+ nor $\neg\varphi^+$. We also take $I_V(t)(s)$ to be the value v such that there is a constructive proof of $I_V(t = v)(s)$. Note that this says that, just as we may not have evidence for either φ nor $\neg\varphi$ in constructive logic, not all terms are computable in Nuprl and $I_V(t)(s)$ may not be defined for all terms and states s.

A formula φ is an *i-formula in interpretation I* if its meaning in I depends only in i's local state; that is, for all global states s and s' such that $s_i = s'_i$, $I_V(\varphi)(s)$ holds iff $I_V(\varphi)(s')$ does. Similarly, t is an *i-term in I* if $x = t$ is an i-formula in I, for x a non-local variable. It is easy to see that φ is an i-formula in all interpretations I if all the predicate and function symbols in φ are rigid, and φ does not mention variables in X_j for $j \neq i$ and does not mention the constant symbol $\mathbf{val}_j$ for $j \neq i$. Intuitively, this is because if we have a constructive proof that φ holds in s with respect to valuation V, and φ is an i-formula, then all references to local states of agents other than i can be safely discarded from the argument to construct a proof for φ based solely on s_i. If φ is an i-formula, then we sometimes abuse notation and write $I_V(\varphi)(s_i)$ rather than $I_V(\varphi)(s)$. Note that the valuation V is not needed for interpreting formulas whose free variables are all local; in particular, V is not needed to interpret i-formulas. For the rest of this chapter, if the valuation is not needed, we do not mention it, and simply write $I(\varphi)$. Given a formula φ and term t, we can easily define Nuprl formulas *i-formula*(φ,I) and *i-term*(t,I) that are constructively provable if φ is an i-formula in I (resp., t is an i-term in I).

We define a predicate $Consistent_I$ on programs and event structures such that, intuitively, $Consistent_I\ (Pg, es)$ holds if the event structure es is consistent with program Pg, given interpretation I. We start with basic programs. The basic program $@i$ $\mathbf{initially}$ ψ is an *initialization* program, which is intended to hold in an event structure es if ψ is an i-formula and i's initial local state satisfies ψ. Thus,

$$Consistent_I(@i\ \mathbf{initially}\ \psi, es) =_{\text{def}} \textit{i-formula}(\psi, I) \wedge I(\psi)(initstate_i).$$

(This notation implicitly assumes that $initstate_i$ is as specified by es, according to Definition 3.1.1. For simplicity, we have opted for this notation instead of $es.initstate_i$.)

We call a basic program of the form $@i$ $\mathbf{if}$ $kind = k$ $\mathbf{then}$ $x := t$ an *effect* program. It says that, if t is an i-term, then the effect of an event e of kind k is to set x to t. We define

$$\begin{aligned}&Consistent_I(@i\ \mathbf{if}\ kind = k\ \mathbf{then}\ x := t, es) =_{\text{def}}\\ &\quad \textit{i-term}(t, I) \wedge\\ &\quad \forall e@i \in es.\ (kind(e) = k \Rightarrow (state\ after\ e)(x) = I(t)(state\ before\ e)),\end{aligned}$$

where we write $\forall e@i \in es.\ \varphi$ as an abbreviation for $\forall e \in es.agent(e) = i \Rightarrow \varphi$. As above, the notation above implicitly assumes that $before$ and $after$ are as specified by es. Again, this expression is an abbreviation for a formula expressible in Nuprl whose intended meaning should be clear; $Consistent_I(@i\ \mathbf{if}\ kind = k\ \mathbf{then}\ x := t, es)$ holds if there is a constructive proof of the formula.

We can use a program of this type to describe a message sent on a link l. For example,

$$@i\ kind = local(a)\ \mathbf{then}\ \mathbf{msg}(l) := \mathbf{f}(\mathbf{val}_i)$$

says that for all events e, $f(v)$ is sent on link l if the kind of e is a, the local state of agent i before e is s_i, and $v = s_i(\mathbf{val}_i)$.

The third type of program, $@i\ kind = local(a)$ **only if** φ, is called a *precondition* program. It says that an event of kind a can occur only if the precondition φ (which must be an i-formula) is satisfied:

$$Consistent_I(@i\ kind = local(a)\ \textbf{only if}\ \varphi, es) =_{\text{def}}$$
$$i\textit{-formula}(\varphi, I)\ \wedge \forall e@i \in es.\ (kind(e) = local(a) \Rightarrow I(\varphi)(state\ before\ e)).$$

Note that we allow conditions of the form $kind(e) = local(a)$ here, not the more general condition of the form $kind(e) = k$ allowed in effect programs. We do not allow conditions of the form $kind(e) = rcv(l)$ because we assume that receive events are not under the control of the agent.

Standard formalizations of input-output automata (see [60]) typically assume that executions satisfy some fairness constraints. We assume here only a weak fairness constraint that is captured by the basic program $@i$ **if necessarily** φ **then i.o.** $kind = local(a)$, which we call a *fairness program*. Intuitively, it says that if φ holds from some point on, then an event with kind $local(a)$ will eventually occur. For an event sequence with only finitely many states associated with i, we take φ to hold "from some point on" if φ holds at the last state. In particular, this means that the program cannot be consistent with an event sequence for which there are only finitely many events associated with i if φ holds of the last state associated with i. Define

$$Consistent_I(@i\ \textbf{if necessarily}\ \varphi\ \textbf{then i.o.}\ kind = local(a), es) =_{\text{def}}$$
$$i\textit{-formula}(\varphi, I) \wedge$$
$$[((\exists e@i \in es) \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ I(\neg\varphi)(state\ after\ e') \vee (kind(e') = local(a)))$$
$$\vee (\neg(\exists e@i \in es) \wedge I(\neg\varphi)(initstate_i))].$$

The last type of basic program, $@$**i only L affects** x, is called a *frame program*. It ensures that only events of kinds listed in L can cause changes in the value of variable

x. The precise semantics depends on whether x has the form $msg(l)$. If x does not have the form $msg(l)$, then

$$\begin{aligned}&\mathit{Consistent}_I(@i\ \textbf{only L affects}\ x, es) =_{\text{def}}\\ &\quad \forall e@i \in es.\ ((x\ \mathit{after}\ e) \neq (x\ \mathit{before}\ e) \Rightarrow (\mathit{kind}(e) \in L)).\end{aligned}$$

If x has the form $msg(l)$, then we must have $\mathit{source}(l) = i$. Recall that sending a message m on l is formalized by setting the value of $msg(l)$ to m. We assume that messages are never null (i.e., $m \neq \bot$). No messages are sent during event e if $msg(l)\,\mathit{after}\ e = \bot$. If x has the form $msg(l)$, then

$$\begin{aligned}&\mathit{Consistent}_I(@i\ \textbf{only L affects msg}(l), es) =_{\text{def}}\\ &\quad \forall e@i \in es.\ ((msg(l)\ \mathit{after}\ e \neq \bot) \Rightarrow (\mathit{kind}(e) \in L)).\end{aligned}$$

Finally, an event structure es is said to be consistent with a distributed program Pg that is not basic if es is consistent with each of the basic programs that form Pg:

$$\mathit{Consistent}_I(Pg_1 \oplus Pg_2, es) =_{\text{def}} \mathit{Consistent}_I(Pg_1, es) \wedge \mathit{Consistent}_I(Pg_2, es).$$

Definition 3.1.4: Given an interpretation I, the semantics of a program Pg is the set of event structures consistent with Pg under interpretation I. We denote by S_I this semantics of programs: $S_I(Pg) = \{es \mid \ \mathit{Consistent}_I(Pg, es)\}$. We write $Pg \mathrel{|\!\!\approx}_I X$ if Pg satisfies X with respect to interpretation I; that is, if $X(es)$ is true for all $es \in S_I(Pg))$. ∎

Note that $S_I(Pg_1 \oplus Pg_2) = S_I(Pg_1) \cap S_I(Pg_2)$. Since the $\mathit{Consistent}_I$ predicate is definable in Nuprl, we can formally reason in Nuprl about the semantics of programs.

A specification is a predicate on event structures. Since our main goal is to derive from a proof that a specification X is satisfiable a program that satisfies X, we want to rule out the trivial case where the derived program Pg has no executions, so that it vacuously satisfies the specification X.

Definition 3.1.5: Program Pg is *consistent (with respect to interpretation I)* if $S_I(Pg) \neq \emptyset$. The specification X is *realizable (with respect to interpretation I)* if it is not vacuously satisfied, that is, if $\exists Pg.(Pg \models\!\approx_I X \wedge S_I(Pg) \neq \emptyset)$. Pg *realizes* X (with respect to I) if $Pg \models\!\approx_I X$ and Pg is consistent (with respect to I). ∎

Thus, a specification is realizable if there exists a consistent program that satisfies it, and, given an interpretation I, a program is realizable if there exists an event structure consistent with it (with respect to I). Since we reason constructively, this means that a program is realizable if we can *construct* an event structure consistent with the program. This requires not only constructing sequences of events, one for each agent, but all the other components of the event structure as specified in Definition 3.1.3, such as AG and Act.

All basic programs other than initialization and fairness programs are vacuously satisfied (with respect to every interpretation I) by the empty event structure es consisting of no events. The empty event structure is consistent with these basic programs because their semantics in defined in terms of a universal quantification over events associated with an agent. It is not hard to see that an initialization program $@i$ **initially** ψ is consistent with respect to interpretation I if and only if ψ is satisfiable in I; i.e., there is some global state s such that $I(\psi)(s_i)$ holds. For if es is an event structure with $initstate_i = s_i$, then clearly es realizes $@i$ **initially** ψ.

Fair programs are realizable with respect to interpretations I where the precondition φ satisfies the principle of excluded middle (that is, $\varphi_I \Rightarrow Determinate(\varphi^+)$ is provable in Nuprl), although they are not necessarily realized by a finite event structure. To see this, note that if φ satisfies the principle of excluded middle in I, then either there is an I-local state s_i^* for agent i such that $I(\neg\varphi)(s_i^*)$ holds, or $I(\varphi)(s_i)$ holds for all I-local states s_i for i. In the former case, consider an empty event structure es with domain Val_I and $initstate_i = s_i^*$; it is easy to see that es is consistent with $@i$ **if necessarily** φ **then i.o.** $kind = local(a)$. Otherwise, let $Act = \{a\}$. Let es be an event structure where Act is the set of local actions, Val_I is the set of values, the sequence of events associated with agent i in es is infinite, and all events associated with agent i have kind $local(a)$. Again, it is easy to see that es is consistent with $@i$ **if necessarily** φ **then i.o.** $kind = local(a)$.

If φ does not satisfy the principle of excluded middle in I, then $@i$ **if necessarily** φ **then i.o.** $kind = local(a)$ may not be realizable with respect to I. For example, this would be the case if for example, neither $I(\varphi)(s_i)$ nor $I(\neg\varphi)(s_i)$ holds for any local state s_i.

Note that two initialization programs may each be consistent although their composition is not. For example, if both ψ and $\neg\psi$ are satisfiable i-formulas, then each of $@i$ **initially** ψ and $@i$ **initially** $\neg\psi$ is consistent, although their composition is not. Nevertheless, all programs synthesized in this chapter can be easily proven consistent.

Axioms

Bickford and Constable [11] derived from the formal semantics of distributed message automata some Nuprl axioms that turn out to be useful for proving the satisfiability of a specification. We now present (a slight modification of) their axioms. The axioms have the form $Pg \models\!\!\approx_I X$, where Pg is a program and X is a specification, that is, a predicate on event structures; the axiom is sound if all event structures es consistent with program

Pg under interpretation I satisfy the specification X. We write $\approx\!\!\!|_I$ to make clear that the program semantics in given with respect to an interpretation I. There is an axiom for each type of basic program other than frame programs, two axioms for frame programs (corresponding to the two cases in the semantic definition of frame programs), together with an axiom characterizing composition and a refinement axiom.

Ax-init:

$$@i\ \textbf{initially}\ \psi \mathrel{|\!\!\approx_I} \lambda es.\ i\text{-}\textit{\textbf{formula}}(\psi, I) \wedge I(\psi)(\mathit{initstate}_i).$$

(Note that the right-hand side of $|\!\!\approx$ is a specification; given an event structure es, it is true if $i\text{-}\textit{\textbf{formula}}(\psi, I) \wedge I(\psi)(\mathit{initstate}_i)$ holds in event structure es.)

Ax-cause:

$$\begin{aligned}&@i\ \textbf{if}\ \mathit{kind} = k\ \textbf{then}\ x{:=}t \mathrel{|\!\!\approx_I}\\ &\quad \lambda es.\quad i\text{-}\textit{\textbf{term}}(t, I) \wedge \forall e@i \in es.\ (\mathit{kind}(e) = k \Rightarrow\\ &\qquad\qquad (\mathit{state\ after}\ e)(x) = I(t)(\mathit{state\ before}\ e)).\end{aligned}$$

Ax-if:

$$\begin{aligned}&@i\ \mathit{kind} = \mathit{local}(a)\ \textbf{only if}\ \varphi \mathrel{|\!\!\approx_I}\\ &\quad \lambda es.\quad i\text{-}\textit{\textbf{formula}}(\varphi, I) \wedge\\ &\qquad\quad \forall e@i \in es.\ (\mathit{kind}(e) = \mathit{local}(a) \Rightarrow I(\varphi)(\mathit{state\ before}\ e)).\end{aligned}$$

Ax-fair:

$$\begin{aligned}&@i\ \textbf{if necessarily}\ \varphi\ \textbf{then i.o.}\ \mathit{kind} = \mathit{local}(a) \mathrel{|\!\!\approx_I}\\ &\quad \lambda es.\quad i\text{-}\textit{\textbf{formula}}(\varphi, I)\wedge\\ &\qquad\quad [((\exists e@i \in es) \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ I(\neg\varphi)(\mathit{state\ after}\ e')\\ &\qquad\quad \vee(\mathit{kind}(e') = \mathit{local}(a)))\\ &\qquad\quad \vee(\neg(\exists e@i \in es) \wedge I(\neg\varphi)(\mathit{initstate}_i))].\end{aligned}$$

Ax-affects:

$$@i \textbf{ only L affects } x \mathrel{|\!\approx_I}$$
$$\lambda es.\, \forall e@i \in es.\, (x \textit{ after } e \neq x \textit{ before } e) \Rightarrow (\textit{kind}(e) \in L).$$

Ax-sends:

$$@i \textbf{ only L affects msg}(l) \mathrel{|\!\approx_I}$$
$$\lambda es.\, \forall e@i \in es.\, (\textit{msg}(l) \textit{ after } e \neq \bot) \Rightarrow (\textit{kind}(e) \in L).$$

Ax-$\oplus$: $(Pg_1 \mathrel{|\!\approx_I} P) \wedge (Pg_2 \mathrel{|\!\approx_I} Q) \Rightarrow (Pg_1 \oplus Pg_2 \mathrel{|\!\approx_I} P \wedge Q)$.

Ax-ref: $(Pg \mathrel{|\!\approx_I} P) \wedge (P \Rightarrow Q) \Rightarrow (Pg \mathrel{|\!\approx_I} Q)$.

Lemma 3.1.6: *Axioms* **Ax-init**, **Ax-cause**, **Ax-if**, **Ax-fair**, **Ax-affects**, **Ax-sends**, **Ax-$\oplus$**, *and* **Ax-ref** *hold for all interpretations* I.

Proof: This is immediate from Definitions 3.1.1 and 3.1.4, and the definition of $Consistent_I$. ▮

A general scheme for program synthesis

Recall that, given a specification φ and an interpretation I, the goal is to prove that φ is satisfiable with respect to I, that is, to show that $\exists Pg.\, (Pg \mathrel{|\!\approx_I} \varphi)$ holds. We now provide a general scheme for doing this. Consider the following scheme, which we call $\mathcal{GS}$:

1. Find specifications $\varphi_1, \varphi_2, \ldots, \varphi_n$ such that $\forall es.\, (\varphi_1(es) \wedge \varphi_2(es) \wedge \ldots \wedge \varphi_n(es) \Rightarrow \varphi(es))$ is true under interpretation I.
2. Find programs $Pg_1, Pg_2, \ldots, Pg_n$ such that $Pg_i \mathrel{|\!\approx_I} \varphi_i$ holds for all $i \in \{1, \ldots n\}$.

3. Conclude that $Pg \mathrel{|\!\approx_I} \varphi$, where $Pg = Pg_1 \oplus Pg_2 \oplus \ldots \oplus Pg_n$.

Step 1 of $\mathcal{GS}$ is proved using the rules and axioms encoded in the Nuprl system; Step 2 is proved using the axioms given in Section 3.1.3. It is easy to see that $\mathcal{GS}$ is *sound* in the sense that, if we can show using $\mathcal{GS}$ that Pg satisfies φ, then Pg does indeed satisfy φ. We formalize this in the following proposition.

Proposition 3.1.7: *Scheme $\mathcal{GS}$ is sound.*

3.1.4 Example

As an example of a specification that we use later, consider the run-based specification $\mathit{Fair}_I(\varphi, t, l)$, where $i \neq j$, l is a link with $\mathit{source}(l) = i$ and $\mathit{dest}(l) = j$, φ is an i-formula, and t is an i-term. $\mathit{Fair}_I(\varphi, t, l)$ is a conjunction of a safety condition and a liveness condition. The safety condition asserts that if a message is received on link l, then it is the term t interpreted with respect to the local state of the sender, and that φ, evaluated with respect to the local state of the sender, holds. The liveness condition says that, if (there is a constructive proof that) condition φ is enabled from some point on in an infinite event sequence, then eventually a message sent on l is delivered. (Thus, the specification imposes a weak fairness requirement.) We define $Fair_I(\varphi, t, l)$ as follows:

$$
\begin{aligned}
& Fair_I(\varphi, t, l) =_{\text{def}} \lambda es.\ \textit{i-formula}(\varphi, I) \wedge \textit{i-term}(t, I)\\
& (\forall e' \in es.\ (\mathit{kind}(e') = \mathit{rcv}(l) \Rightarrow\\
& \quad I(\varphi)(\mathit{state\ before\ send}(e')) \wedge \mathit{val}(e') = I(t)(\mathit{state\ before\ send}(e'))) \wedge\\
& ((\exists e@i \in es \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ I(\neg\varphi)(\mathit{state\ after\ } e')) \vee\\
& (\neg(\exists e@i \in es) \wedge I(\neg\varphi)(\mathit{initstate}_i)) \vee\\
& (\exists e@i \in es \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ \mathit{kind}(e') = \mathit{rcv}(l) \wedge \mathit{send}(e') \succeq_i e)).
\end{aligned}
$$

We are interested in this fairness specification only in settings where communication satisfies a (strong) fairness requirement: if infinitely often an agent sends a message on a link l, then infinitely often some message is delivered on l. We formalize this assumption using the following specification:

$$\begin{aligned} \mathit{FairSend}(l) =_{\text{def}} \lambda es.\quad & (\forall e@i \in es.\, \exists e' \succ_i e.\, \mathit{msg}(l)\ \mathit{after}\ e' \neq \bot) \\ & \Rightarrow (\forall e@i \in es.\, \exists e'.\quad \mathit{kind}(e') = \mathit{rcv}(l) \wedge \mathit{send}(e') \succ_i e). \end{aligned}$$

We explain below why we need communication to satisfy strong fairness rather than weak fairness (which would require only that if a message is sent infinitely often, then a message is eventually delivered).

For an arbitrary action a, let $\mathit{Fair\text{-}Pg}(\varphi, t, l, a)$ be the following program for agent i:

$$\begin{aligned} & @i\ \mathit{kind} = \mathit{local}(a)\ \textbf{only if}\ \varphi \oplus \\ & @i\ \textbf{if}\ \mathit{kind} = \mathit{local}(a)\ \textbf{then msg}(l){:=}t \oplus \\ & @i\ \textbf{only events in}\ [a]\ \textbf{affect msg}(l) \oplus \\ & @i\ \textbf{if necessarily}\ \varphi\ \textbf{then i.o.}\ \mathit{kind} = \mathit{local}(a). \end{aligned}$$

The first basic program says that i takes action a only if φ holds. The second basic program says that the effect of agent i taking action a is for t to be sent on link l; in other words, a is i's action of sending t to agent j. The third program ensures that only action a has the effect of sending a message to agent j. With this program, if agent j (the receiver) receives a message from agent i (the sender), then it must be the case that the value of the message is t and that φ was true with respect to i's local state when it sent the message to j. The last basic program ensures that if φ holds from some point on in an infinite event sequence, then eventually an event of kind a holds; thus, i must send the message t infinitely often. The fairness requirement on communication ensures that if an event of kind a where i sends t occurs infinitely often, then t is received infinitely often.

Lemma 3.1.8: *For all actions a, $\mathit{Fair}\text{-}\mathit{Pg}(\varphi,t,l,a)$ satisfies $\lambda es.\mathit{FairSend}(l)(es) \Rightarrow \mathit{Fair}_I(\varphi,t,l)(es)$ with respect to all interpretations I such that φ is an i-formula and t is an i-term in I.*

Proof: We present the key points of the proof here, omitting some details for ease of exposition. We follow the scheme $\mathcal{GS}$. We assume that i-*formula*(φ, I) and i-*term*(t, I) both hold.

Step 1. For each event structure es, $\mathit{Fair}_I(\varphi,t,l)(es)$ is equivalent to a conjunction of three formulas:

$$\begin{aligned}
\varphi_1(es):\quad & \forall e' \in es.\ (\mathit{kind}(e') = \mathit{rcv}(l)) \Rightarrow I(\varphi)(\mathit{state\ before\ send}(e'))\\
\varphi_2(es):\quad & \forall e' \in es.\ (\mathit{kind}(e') = \mathit{rcv}(l)) \Rightarrow \mathit{val}(e') = I(t)(\mathit{state\ before\ send}(e'))\\
\varphi_3(es):\quad & (\exists e@i \in es \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ I(\neg\varphi)(\mathit{state\ after\ } e')) \vee\\
& (\neg(\exists e@i \in es) \wedge I(\neg\varphi)(\mathit{initstate}_i) \vee\\
& (\exists e@i \in es \wedge \forall e@i \in es.\ \exists e'.\mathit{kind}(e') = \mathit{rcv}(l) \wedge \mathit{send}(e') \succ_i e).
\end{aligned}$$

We want to find formulas $\psi_1(es), \ldots, \psi_4(es)$ that follow from the four basic programs that make up $\mathit{Fair}\text{-}\mathit{Pg}(\varphi,t,l,a)$ and together imply $\varphi_1(es) \wedge \varphi_2(es) \wedge \varphi_3(es)$. It will simplify matters to reason directly about the events where a message is sent on link l. We thus assume that, for all events e, agent i sends a message on link l during event e iff $\mathit{kind}(e) = \mathit{local}(a)$. This assumption is expressed by:

$$\psi_1(es) =_{\text{def}} \forall e@i \in es.\ (\mathit{msg}(l)\ \mathit{after}\ e \neq \bot) \Rightarrow (\mathit{kind}(e) = \mathit{local}(a)).$$

It is easy to check that $(\psi_1(es) \wedge \psi_2(es)) \Rightarrow \varphi_1(es))$ is true, where $\psi_2(es)$ is

$$\forall e@i \in es.\ (\mathit{kind}(e) = \mathit{local}(a)) \Rightarrow I(\varphi)(\mathit{state\ before\ } e).$$

Similarly, using the axiom of event structures given in Section 3.1.2 that says that the value of a receive event e on l is the value of $\mathit{msg}(l)$ after $\mathit{send}(e)$, it is easy to check that $(\psi_1(es) \wedge \psi_3(es)) \Rightarrow \varphi_2(es))$ is true, where $\psi_3(es)$ is

$$\forall e@i \in es.(kind(e) = local(a)) \Rightarrow msg(l) \; after \; e = I(t)(state \; before \; e).$$

We can show that $(\psi_3(es) \wedge \psi_4(es) \wedge FairSend(l)) \Rightarrow \varphi_3(es)$ is true, where ψ_4 is

$$(\exists e@i \wedge \forall e@i \in es.\, \exists e' \succeq_i e.\, I(\neg\varphi)(state \; after \; e')) \vee$$
$$(\neg(\exists e@i \in es) \wedge I(\neg\varphi)(initstate_i)) \vee$$
$$(\exists e@i \in es \wedge \forall e@i \in es.\, \exists e' \succeq_i e.\, kind(e') = local(a)).$$

It follows that

$$(\forall es.(\psi_1(es) \wedge \psi_2(es) \wedge \psi_3(es) \wedge \psi_4(es)) \Rightarrow (FairSend(l)(es) \Rightarrow Fair_I(\varphi, t, l)(es))).$$

Step 2. By **Ax-sends**

$$@i \textbf{ only } [a] \textbf{ affects msg}(l) \models\!\!\approx_I \psi_1.$$

By **Ax-if**,

$$@i \; kind = local(a) \textbf{ only if } \varphi \models\!\!\approx_I \psi_2.$$

By **Ax-cause**,

$$@i \textbf{ if } kind = local(a) \textbf{ then msg}(l) := t \models\!\!\approx_I \psi_3;$$

and by **Ax-fair**

$$@i \textbf{ if necessarily } \varphi \textbf{ then i.o}.\; kind = local(a) \models\!\!\approx_I \psi_4.$$

By the soundness of $\mathcal{GS}$ (Proposition 3.1.7), $Fair\text{-}Pg(\varphi, t, l, a)$ satisfies $\lambda es.FairSend(l)(es) \Rightarrow Fair_I(\varphi, t, l)(es)$ with respect to I. ∎

Lemma 3.1.9: *For all interpretations I such that φ is an i-formula and t is an i-term in I, if φ satisfies the principle of excluded middle with respect to I, then $Fair\text{-}Pg(\varphi,t,l,a)$ is consistent with respect to I.*

Proof: This argument is almost identical to that showing that fair programs are realizable with respect to interpretations where the precondition satisfies the principle of excluded middle. Since φ satisfies the principle of excluded middle with respect to I, either there exists an I-local state s_i^* for agent i such that $I(\neg\varphi)(s_i^*)$ holds, or $I(\varphi)(s_i)$ holds for all I-local states s_i for i. In the former case, let es be an empty event structure such that $i, j \in AG$, $l \in Links$, $a \in Act$, and $initstate_i = s_i^*$. In the latter case, choose es with AG and $Links$ as above, let $Act = \{a, b\}$, and where i and j alternate sending and receiving the message t on link l, where these events have kind a and b, respectively. ∎

Corollary 3.1.10: *For all interpretations I such that if φ is an i-formula and t is an i-term in I, if φ satisfies the principle of excluded middle with respect to I, then the specification $Fair_I(\varphi, t, l)$ is realizable with respect to I.*

Proof: This is immediate from Lemmas 3.1.8 and 3.1.9, and from the fact that the event structure constructed in Lemma 3.1.8 satisfies $FairSend(l)$. ∎

The notion of strong communication fairness is essential for the results above: $Fair_I(\varphi, t, l)$ may not be realizable if we assume that communication satisfies only a weak notion of fairness that says that if a message is sent after some point on, then it is eventually received. This is so essentially because our programming language is replacing standard "if condition then take action" programs with weaker variants that ensure that, if after some point a condition holds, then eventually some action is taken.

We now show that the composition of $\mathit{Fair}\text{-}\mathit{Pg}(\varphi, t, l, a)$ and $\mathit{Fair}\text{-}\mathit{Pg}(\varphi, t, l', a)$ for different links l and l' satisfies the corresponding fairness assumptions.

Lemma 3.1.11: *For all distinct actions a and a', and all distinct links l and l', $Fair\text{-}Pg(\varphi, t, l, a) \oplus Fair\text{-}Pg(\varphi', t', l', a')$ satisfies*

$$\lambda es.(\mathit{FairSend}(l)(es) \wedge \mathit{FairSend}(l')(es)) \Rightarrow$$
$$(Fair_I(\varphi, t, l)(es) \wedge Fair_I(\varphi', t', l')(es))$$

with respect to all interpretations I such that φ is an i-formula, t is an i-term, φ' is an i'-formula, and t' is an i'-term in I.

Proof: Suppose $a \neq a'$. We again use scheme $\mathcal{GS}$.

Step 1. Clearly, we can take φ_1 to be $\lambda es.\ \mathit{FairSend}(l)(es) \Rightarrow Fair_I(\varphi, t, l)(es)$ and φ_2 to be $\lambda es.\ \mathit{FairSend}(l')(es) \Rightarrow Fair_I(\varphi', t', l')(es)$.

Step 2. By Lemma 3.1.8, $Fair\text{-}Pg(\varphi, t, l, a) \models\!\approx_I \varphi_1$ and $Fair\text{-}Pg(\varphi', t', l', a') \models\!\approx_I \varphi_2$.
∎

Finally, we can show that $Fair\text{-}Pg(\varphi, t, l, a) \oplus Fair\text{-}Pg(\varphi', t', l', a')$ is consistent, where l is a link from i to j, l' is a link from i' to j', and $l \neq l'$ (so that we may have $i = i'$ or $j = j'$, but not both), and thus the specification $\lambda es.(\mathit{FairSend}(l)(es) \wedge \mathit{FairSend}(l')(es)) \Rightarrow (Fair_I(\varphi, t, l)(es) \wedge Fair_I(\varphi', t', l')(es))$ is realizable with respect to I. if both φ and φ' satisfy the principle of excluded middle with respect to I.

Lemma 3.1.12: *For all interpretations I such that φ is an i-formula, t is an i-term, φ' is an i'-formula, and t' is an i-term in I, if both φ and φ' satisfy the principle of excluded middle with respect to I, then, for all distinct actions a and a' and all distinct links l and l', $Fair\text{-}Pg(\varphi, t, l, a) \oplus Fair\text{-}Pg(\varphi', t', l', a')$ is consistent with respect to I.*

Proof: If $I(\neg\varphi \wedge \neg\varphi)(s)$ holds for some global state s, then let es be the empty event structure such that $initstate_i = s_i$ and $initstate_{i'} = s_{i'}$. Clearly es is consistent with $Fair\text{-}Pg(\varphi,t,l,a) \oplus Fair\text{-}Pg(\varphi',t',l',a')$. Otherwise, let es be an event structure with domain Val_I, $i,j,i',j' \in AG$, and $l,l' \in Links$, consisting of an infinite sequence of states such that if $I(\varphi)$ holds for infinitely many states, then i sends t on link l infinitely often; if $I(\varphi')$ holds for infinitely many states, then i' sends t' on link l' infinitely often; if t is sent on l infinitely often, then j receives it on link l infinitely often; and if t' is sent on l' infinitely often, then j' receives it on l' infinitely often. It is straightforward to construct such an event structure es. Again, it should be clear that es is consistent with $Fair\text{-}Pg(\varphi,t,l,a) \oplus Fair\text{-}Pg(\varphi',t',l',a')$. ∎

3.2 Adding knowledge to Nuprl

We now show how knowledge-based programs can be introduced into Nuprl.

3.2.1 Consistent cut semantics for knowledge

We want to extend basic programs to allow for tests that involve knowledge. For simplicity, we take $AG = \{1, 2, \ldots, n\}$. As before, we start with a finite set $\mathcal{P} \cup \mathcal{F}$ of predicates and functions, and close off under conjunction, negation, and quantification over non-local variables; but now, in addition, we also close off under application of the temporal operators $\Box$ and $\Diamond$, and the epistemic operators K_i, $i = 1, \ldots, n$, one for each process i.

We again want to define a consistency relation in Nuprl for each program. To do that, we first need to review the semantics of knowledge. Typically, semantics for knowledge is given with respect to a pair (r, m) consisting of a run r and a time m, assumed to be the time on some external global clock (that none of the processes necessarily has access to [29]). In event structures, there is no external notion of time. Fortunately, Panangaden and Taylor [76] give a variant of the standard definition with respect to what they call *asynchronous runs*, which are essentially identical to event structures. We just apply their definition in our framework.

The truth of formulas is defined relative to a pair (Sys, c), consisting of a system Sys (i.e., a a set of event structures) and a *consistent cut* c of some event structure $es \in Sys$, where a *consistent cut* c in es is a set of events in es closed under the causality relation. Recall from Section 3.1.2 that this amounts to c satisfying the constraint that, if e' is an event in c and e is an event in es that precedes e' (i.e., $e \prec e'$), then e is also in c. We write $c \in Sys$ if c is a consistent cut in some event structure in Sys.

Traditionally, a knowledge formula $K_i\varphi$ is interpreted as true at a point (r, m) if φ is true regardless of i's uncertainty about the whole system at (r, m). Since we interpret formulas relative to a pair (Sys, c), we need to make precise i's uncertainty at such a pair. For the purposes of this chapter, we assume that each agent keeps track of all the events that have occurred and involved him (which corresponds to the assumption that agents have *perfect recall*); we formalize this assumption below. Even in this setting, agents can be uncertain about what events have occurred in the system, and about their relative order. Consider, for example, the scenario in the left panel of Figure 3.1: agent i receives a message from agent j (event e_2), then sends a message to agent k (e_3), then receives a second message from agent j (e_6), and then performs an internal action (e_7). Agent i knows that $send(e_2)$ occurred prior to e_2 and that $send(e_6)$ occurred prior to e_6. However, i considers possible that after receiving his message, agent k sent a message to j which was received by j before e_7 (see the right panel of Figure 3.1).

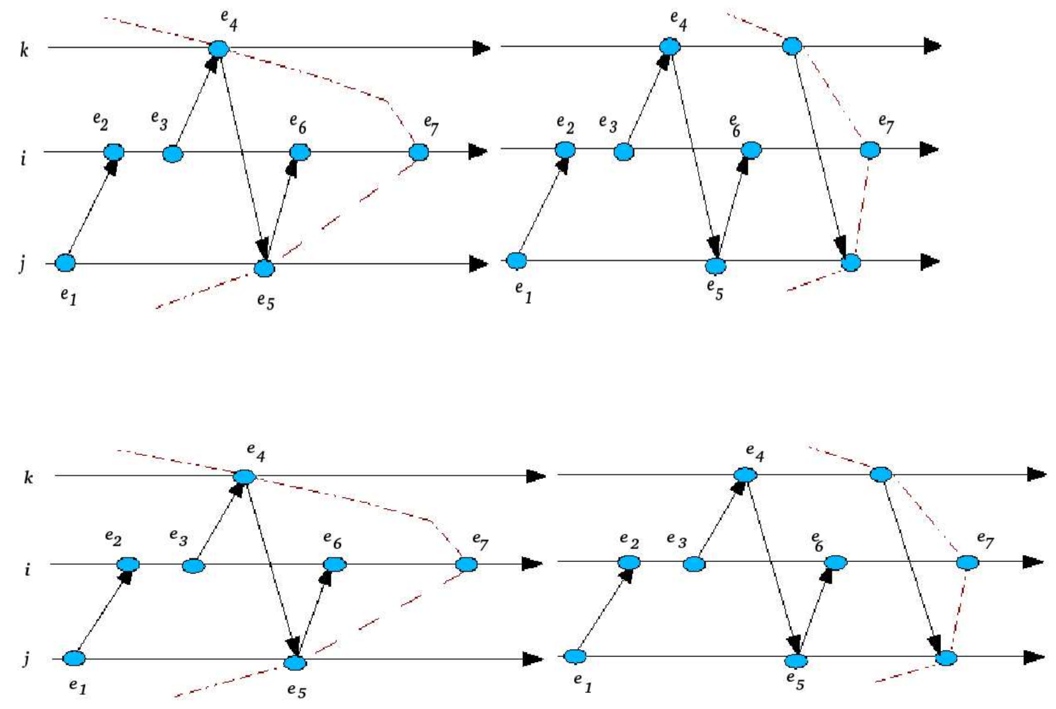

Figure 3.1: Two consistent cuts that cannot be distinguished by agent i.

In general, as argued by Panangaden and Taylor, agent i considers possible any consistent cut in which he has recorded the same sequence of events. To formalize this intuition, we define equivalence relations $\sim_i$, $i = 1, \ldots, n$, on consistent cuts by taking $c \sim_i c'$ if i's history is the same in c and c'. Given two consistent cuts c and c', we say that $c \preceq c'$ if, for each process i, process i's history in c is a prefix of process i's history in c'. Relative to (Sys, c), agent i considers possible any consistent cut $c' \in Sys$ such that $c' \sim_i c$.

Since the semantics of knowledge given here implicitly assumes that agents have perfect recall, we restrict to event structures that also satisfy this assumption. So, for the remainder of this chapter, we restrict to systems where *local states encode histories*, that is, we restrict to systems Sys such that, for all event structures $es, es' \in Sys$, if e is an event in es, e' is an event in es', $agent(e) = agent(e') = i$, and $state\ before\ e = state\ before\ e'$, then i has the same history in both es and es'. For simplicity, we guarantee this by assuming that each agent i has a local variable $history_i \in X_i$ that encodes its history. Thus, we take $initstate_i(history_i) = \perp$ and for all events e associated with agent i, we have $(s\ after\ e)(history_i) = (s\ before\ e)(history_i) \cdot e$. It immediately follows that in two global states where i has the same local state, i must have the same history. Let $System$ be the set of all such systems.

Recall that events associated with the same agent are totally ordered. This means that we can associate with every consistent cut c a global state s^c: for each agent i, s_i^c is i's local state after the last event e_i associated with i in c occurs. Since local states encode histories, it follows that if $s_i^c = s_i^{c'}$, then $c \sim_i c'$. It is not difficult to see that the converse is also true; that is, if $c \sim_i c'$, then $s_i^c = s_i^{c'}$. We also write $s^c \prec s^{c'}$ if $c \prec c'$. In the following, we assume that all global states in a system Sys have the form s^c for some consistent cut c.

Nuprl is rich enough that epistemic and modal operators can be defined within Nuprl. Thus, to interpret formulas with epistemic operators and temporal operators, we just translate them to formulas that do not mention them. Since the truth of an epistemic formula depends not just on a global state, but on a pair (Sys, c), where the consistent cut c can be identified with a global state in some event structure in Sys, the translated formulas will need to include variables that, intuitively, range over systems and global states. To make this precise, we expand the language so that it includes rigid binary predicates CC and $\succeq$, a rigid binary function ls, and rigid constants s and Sys. Intuitively, s represents a global state, Sys represents a system, $\mathrm{CC}(x, y)$ holds if y is a consistent cut (i.e., global state) in system x, $\mathbf{ls}(x, i)$ is i's local state in global state x, and $\succeq$ represents the ordering on consistent cuts defined above.

For every formula that does not mention modal operators, we take $\varphi^t = \varphi$. We define

$$(K_i\varphi)^t =_{\text{def}} \forall \mathrm{s}'((\mathbf{CC}(\mathbf{Sys}, \mathrm{s}') \wedge \mathbf{ls}(\mathrm{s}', i) = \mathbf{ls}(\mathrm{s}, i)) \Rightarrow \varphi^t[\mathrm{s}/\mathrm{s}'])),$$

$$(\Box\varphi)^t =_{\text{def}} \forall \mathrm{s}'((\mathbf{CC}(\mathbf{Sys}, \mathrm{s}') \wedge \mathrm{s}' \succeq \mathrm{s} \Rightarrow \varphi^t[\mathrm{s}/\mathrm{s}']),$$

and

$$(\Diamond\varphi)^t =_{\text{def}} \exists \mathrm{s}'((\mathbf{CC}(\mathbf{Sys}, \mathrm{s}') \wedge \mathrm{s}' \succeq \mathrm{s} \wedge \varphi^t[\mathrm{s}/\mathrm{s}']).$$

Given an interpretation I, let I' be the interpretation that extends I by adding to φ_I formulas characterizing $\mathbf{Sys}$, s, $\mathbf{CC}$, $\mathbf{ls}$, and $\succeq$ appropriately. That is, the formulas force $\mathbf{Sys}$ to represent a set of event structures, s to be a consistent cut in one of these event structures, and so on. These formulas are all expressible in Nuprl. More specifically, we restrict here to constructive systems, that is, systems that can be defined in Nuprl. A constructive system Sys can be characterized by a formula φ_{Sys} in Nuprl. φ_{Sys} has a free variable $\mathbf{Sys}$ ranging over systems such that φ_{Sys} holds under interpretation I' and valuation V iff $V(\mathbf{Sys}) = Sys$. We now define a predicate $I'_V(\varphi)$ on systems and global states by simply taking $I'_V(\varphi)(Sys, s)$ to hold iff $\varphi_{I'}$ together with the conjunction of atomic formulas of the form $x = V(x)$ for all non-local variables x that appear in φ, $x = s_i(x)$ for variables $x \in X_i$, $i \in AG$, that appear in φ, $\mathrm{s} = s$, and φ_{Sys}, imply $(\varphi^t)^+$ (where, in going from φ^t to $(\varphi^t)^+$, we continue to use the s). Thus, we basically reduce a modal formula to a non-modal formula, and evaluate it in system Sys using I_V.

Just as in the case of non-epistemic formulas, the valuation V is not needed to interpret formulas whose only free variables are in $\cup_{i \in AG} X_i$. For such formulas, we typically write $I'(\varphi)(Sys, s)$ instead of $I'_V(\varphi)(Sys, s)$. We can also define i-formulas and i-terms, but now whether a formula is an i-formula or a term is an i-term depends, not only on the interpretation, but on the system. A formula φ is an *i-formula in interpretation I' and system Sys* if, for all states s, s' in Sys, $I'_V(\varphi)(Sys, s) = I'_V(\varphi)(Sys, s')$ if $s_i = s'_i$; similarly, t is an *i-term in interpretation I' and system Sys* if $I'_V(t)(Sys, s) = I'_V(t)(Sys, s')$ if $s_i = s'_i$. We write this as *i-formula*(φ, I, Sys) and *i-term*(t, I, Sys), respectively. If φ is an i-formula and t is an i-term in I and Sys for all systems Sys, then we simply write *i-formula*(φ, I) and *i-formula*(t, I). For an i-formula, we often write $I'_V(\varphi)(Sys, s_i)$ rather than $I'_V(\varphi)(Sys, s)$. Note that a Boolean combination of epistemic formulas whose outermost knowledge operators are K_i is guaranteed to be an i-formula in every interpretation, as is a formula that has no nonrigid functions or predicates and does not mention K_j for $j \neq i$. The former claim is immediate from the following lemma.

Lemma 3.2.1: *For all formulas φ, systems Sys, and global states s and s', if $s_i = s'_i$, then $I'(K_i\varphi)(Sys, s)$ holds iff $I'(K_i\varphi)(Sys, s')$ does.*

Proof: Follows from the observation that if we have a proof in Nuprl that an i-formula holds given I', Sys, and $s \in Sys$, then we can rewrite the proof so that it mentions only s_i rather than s. Thus, we actually have a proof that the i-formula holds in all stats $s' \in Sys$ such that $s'_i = s_i$. ∎

3.2.2 Knowledge-based programs and specifications

In this section, we show how we can extend the notions of program and specification presented in Section 3.1 to knowledge-based programs and specifications. This allows us to employ the large body of tactics and libraries already developed in Nuprl to synthesize knowledge-based programs from knowledge-based specifications.

Syntax and semantics

Define *knowledge-based message automata* just as we defined message automata in Section 3.1.3, except that we now allow arbitrary epistemic formulas in tests. If we want to emphasize that the tests can involve knowledge, we talk about *knowledge-based* initialization, precondition, effect, and fairness programs. For the purposes of this chapter, we take knowledge-based programs to be knowledge-based message automata.

We give semantics to knowledge-based programs by first associating with each knowledge-based program a function from systems to systems. Let $(Pg^{kb})^t$ be the result of replacing every formula φ in Pg^{kb} by φ^t. Note that $(Pg^{kb})^t$ is a standard program, with no modal formulas. Given an interpretation I and a system Sys let $I(Sys)$ be the result of adding to φ_I the formula φ_{Sys}.

Now we can apply the semantics of Section 3.1.3 and get the system $S_{I(Sys)}((Pg^{kb})^t)$. In general, the system $S_{I(Sys)}((Pg^{kb})^t)$ will be different from the system Sys. A system Sys *represents* a knowledge-based program Pg^{kb} (with respect to interpretation I) if it is a fixed point of this mapping; that is, if $S_{I(Sys)}((Pg^{kb})^t) = Sys$. Following Fagin et al. [29, 26], we take the semantics of a knowledge-based program Pg^{kb} to be the set of systems that represent it.

Definition 3.2.2: A *knowledge-based program semantics* is a function associating with a knowledge-based program Pg^{kb} and an interpretation I the systems that represent Pg^{kb} with respect to I; that is, $S_I^{kb}(Pg^{kb}) = \{Sys \in \mathit{System} \mid S_{I(Sys)}((Pg^{kb})^t) = Sys\}$. ∎

As observed by Fagin et al. [29, 26], it is possible to construct knowledge-based programs that are represented by no systems, exactly one system, or more than one system. However, there exist conditions (which are often satisfied in practice) that guarantee that a knowledge-based program is represented by exactly one system. Note that, in particular, standard programs, when viewed as knowledge-based programs, are represented by a unique system; indeed, $S_I^{kb}(Pg) = \{S_I(Pg)\}$. Thus, we can view S_I^{kb} as extending S_I.

A (standard) program Pg *implements* the knowledge-based program Pg^{kb} with respect to interpretation I if $S_I(Pg)$ represents Pg^{kb} with respect to I, that is, if $S_{I(S_I(Pg))}((Pg^{kb})^t) = S_I(Pg)$. In other words, by interpreting the tests in Pg^{kb} with respect to the system generated by Pg, we get back the program Pg.

Knowledge-based specifications

Recall that a standard specification is a predicate on event structures. Following [26], we take a knowledge-based specification (*kb specification* from now on) to be a predicate on systems.

Definition 3.2.3: A *knowledge-based specification* is a predicate on $System$. A knowledge-based program Pg^{kb} satisfies a knowledge-based specification Y^{kb} with respect to I, written $Pg^{kb} \models\!\approx_I Y^{kb}$, if all the systems representing Pg^{kb} with respect to I satisfy Y^{kb}, that is, if the following formula holds: $\forall Sys \in S_I^{kb}(Pg^{kb}).\ Y^{kb}(Sys)$. The knowledge-based specification Y^{kb} is *realizable* with respect to I if there exists a (standard) program Pg such that $S_I(Pg) \neq \emptyset$ and $Pg \models\!\approx_I Y^{kb}$ (i.e., $Y^{kb}(S_I(Pg))$ is true). ∎

As for standard basic programs, it is not difficult to show that knowledge-based precondition, effect, and frame programs are trivially consistent: we simply take Sys to consist of only one event structure es with no events. A knowledge-based initialization program is realizable iff $\varphi_I \wedge \psi^t$ is satisfiable. Finding sufficient conditions for fair knowledge-based programs to be realizable is nontrivial. We cannot directly translate the constructions sketched for the standard case to the knowledge-based case because, at each step in the construction (when an event structure has been only partially constructed), we would have to argue that a certain knowledge-based fact holds when interpreted with respect to an entire system and an entire event structure. However, in the next section, the knowledge-based programs used in the argument for STP (which do include fairness requirements) are shown to be realizable.

Axioms

We now consider the extent to which we can generalize the axioms characterizing (standard) programs presented in Section 3.1.3 to knowledge-based programs.

Basic knowledge-based message automata other than knowledge-based precondition and fairness requirement programs satisfy analogous axioms to their standard counterparts. The only difference is that now we view the specifications as functions on systems, not on event structures. For example, the axiom corresponding to $\mathbf{Ax\text{-}init}$ is

$$\begin{aligned}\mathbf{Ax\text{-}initK}: \ @i\ \mathbf{initially}\ \psi \models\!\!\approx_I \lambda Sys.\quad & i\text{-}\mathit{formula}(\psi, I, Sys)\wedge \\ & \forall es \in Sys.\ I(\psi)(Sys, initstate_i).\end{aligned}$$

(Note that here, just as in the definition of $\mathbf{Ax\text{-}init}$, for simplicity, we write $initstate_i$ instead of $es.initstate_i$. Since ψ is constrained to be an i-formula in makes sense to talk about $I(\psi)(Sys, initstate_i)$ instead of $I(\psi)(Sys, s)$ for a global state s with $s_i = initstate_i$.) The knowledge-based analogues of axioms $\mathbf{Ax\text{-}cause}$, $\mathbf{Ax\text{-}affects}$, and $\mathbf{Ax\text{-}sends}$ are denoted $\mathbf{Ax\text{-}causeK}$, $\mathbf{Ax\text{-}affectsK}$, and $\mathbf{Ax\text{-}sendsK}$, respectively, and are identical to the standard versions of these axioms. The knowledge-based counterparts of $\mathbf{Ax\text{-}if}$ and $\mathbf{Ax\text{-}fair}$ now involve epistemic preconditions, which are interpreted with respect to a system:

$$\textbf{Ax-ifK}: \quad @i\ \mathit{kind} = \mathit{local}(a)\ \textbf{only if}\ \varphi \mathrel{|\!\!\approx_I} \lambda \mathit{Sys}.\ i\text{-}\textsf{formula}(\varphi, I, \mathit{Sys}) \wedge$$
$$\forall \mathit{es} \in \mathit{Sys}.\ \forall e@i \in \mathit{es}.\ (\mathit{kind}(e) = \mathit{local}(a)) \Rightarrow I(\varphi)(\mathit{Sys}, \mathit{state\ before}\ e)$$

$$\textbf{Ax-fairK}:$$
$$@i\ \textbf{if necessarily}\ \varphi\ \textbf{then i.o.}\ \mathit{kind} = \mathit{local}(a) \mathrel{|\!\!\approx_I} \lambda \mathit{Sys}.\ i\text{-}\textsf{formula}(\varphi, I, \mathit{Sys}) \wedge$$
$$\forall \mathit{es} \in \mathit{Sys}.\quad ((\exists e@i \in \mathit{es} \wedge \forall e@i \in \mathit{es}.\ \exists e' \succeq_i e.$$
$$I(\neg\varphi)(\mathit{Sys}, \mathit{state\ after}\ e') \vee \mathit{kind}(e') = \mathit{local}(a)) \vee$$
$$(\neg(\exists e@i \in \mathit{es}) \wedge I(\neg\varphi)(\mathit{Sys}, \mathit{initstate}_i(\mathit{es})))).$$

There are also obvious analogues axioms **Ax-ref** and **Ax-$\oplus$**, which we denote **Ax-refK** and **Ax-$\oplus$K** respectively.

Lemma 3.2.4 : *Axioms* **Ax-initK**, **Ax-causeK**, **Ax-affectsK**, **Ax-sendsK**, **Ax-ifK**, **Ax-fairK**, *and* **Ax-refK** *hold for all interpretations I.*

Proof: Since the proofs for all axioms are similar in spirit, we prove only that **Ax-ifK** holds for all interpretations I'. Fix an interpretation I. Let Pg^{kb} be the program $@i\ \mathit{kind} = \mathit{local}(a)$ **only if** φ, where φ is an i-formula. Let Y^{kb} be an instance of **Ax-ifK**:

$$\lambda \mathit{Sys}.\quad i\text{-}\textsf{formula}(\varphi, I, \mathit{Sys}) \wedge$$
$$\forall \mathit{es} \in \mathit{Sys}.\ \forall e@i \in \mathit{es}.\ (\mathit{kind}(e) = \mathit{local}(a)) \Rightarrow I(\varphi)(\mathit{Sys}, \mathit{state\ before}\ e).$$

By Definition 3.2.3, $Pg^{kb} \mathrel{|\!\!\approx_I} Y^{kb}$ is true if and only if, for all systems $\mathit{Sys} \in S_I^{kb}(Pg^{kb})$, $Y^{kb}(\mathit{Sys})$ holds. That is, for all systems Sys such that $S_{I(\mathit{Sys})}((Pg^{kb})^t) = \mathit{Sys}$, the following holds:

$$\forall \mathit{es} \in \mathit{Sys}.\ i\text{-}\textsf{formula}(\varphi, I, \mathit{Sys}) \wedge \forall e@i \in \mathit{es}.\ (\mathit{kind}(e) = \mathit{local}(a)) \Rightarrow$$
$$I(\varphi)(\mathit{Sys}, \mathit{state\ before}\ e).$$

Let Sys be a system such that $S_{I(Sys)}((Pg^{kb})^t) = Sys$. By Definition 3.1.4, all event structures in Sys are consistent with the program $(Pg^{kb})^t$ with respect to interpretation $I(Sys)$. Recall that $(Pg^{kb})^t$ is the (standard) program $@i\ kind = local(a)$ **only if** $\varphi^{\mathbf{t}}$, where $I(Sys)(\varphi^t)(s) = I(\varphi)(Sys, s)$. We can thus apply axiom **Ax-if** and conclude that the following holds for all event structures es consistent with $I(Sys)((Pg^{kb})^t)$ with respect to $I(Sys)$ (i.e., for all $es \in Sys$):

i-*formula*$(\varphi^t, I(Sys)) \wedge \forall e@i \in es.\, (kind(e) = local(a)) \Rightarrow I(Sys)(\varphi^t)(state\ before\ e)$.

The first conjunct says that, for all global states s and s' in Sys, if $s_i = s'_i$ then $I(Sys)(\varphi^t)(s) = I(Sys)(\varphi^t)(s')$, which is equivalent to saying that $I(\varphi)(Sys, s) = I(\varphi)(Sys, s')$, that is, i-*formula*(φ, I, Sys) holds. The second conjunct is equivalent to

$$\forall e@i es.\, (kind(e) = local(a)) \Rightarrow I(\varphi)(Sys, state\ before\ e),$$

by the definition of φ^t and $I(Sys)$. Thus, $Y^{kb}(Sys)$ holds under interpretation I. ▮

The proof of Lemma 3.2.4 involves only unwinding the definition of satisfiability for knowledge-based specifications and the application of simple refinement rules, already implemented in Nuprl. In general, proofs of epistemic formulas will also involve reasoning in the logic of knowledge. Sound and complete axiomatizations of (nonintuitionistic) first-order logic of knowledge are well-known (see [29] for an overview) and can be formalized in Nuprl in a straightforward way. This is encouraging, since it supports the hope that Nuprl's inference mechanism is powerful enough to deal with knowledge specifications, without further essential additions.

Note that **Ax-$\oplus$K** is not included in Lemma 3.2.4. That is because it does not always hold, as the following example shows.

Example 3.2.5: Let $Y_i^{kb} =_{\text{def}} \Box(\neg K_{2-i}(x_i = i))$ for $i = 1, 2$, where $x_i \in X_i$, and let $I = \emptyset$. Let Pg_i, $i = 1, 2$ be the standard program for agent i such that $S_I(Pg_i)$ consists of all the event structures such that $x_i = i$ at all times; that is, Pg_i is the program

$$@i \textbf{ initially } x_i = i \;\oplus\; @i \textbf{ only } \emptyset \textbf{ affects } x_i.$$

Since Pg_i places no constraints on x_{2-i}, is straightforward to prove that $Pg_i \not\approx_I Y_{2-i}^{kb}$, for $i = 1, 2$. On the other hand, $S_I(Pg_1 \oplus Pg_2)$ consists of all the event structures where $x_i = i$ at all times, for $i = 1, 2$, so $Pg_1 \oplus Pg_2 \not\approx_I \neg Y_1^{kb} \wedge \neg Y_2^{kb}$. ∎

3.2.3 Example

Recall from Section 3.1.4 that the specification $\mathit{FairSend}(l) \Rightarrow \mathit{Fair}_I(\varphi, t, l)$ is satisfied by the program $\mathit{Fair\text{-}Pg}(\varphi, t, l, a)$, for all actions a. We now consider a knowledge-based version of this specification. If φ is an i-knowledge-based formula and t is an i-term in I, define

$$\mathit{Fair}_I^{kb}(\varphi, t, l) =_{\text{def}} \lambda \mathit{Sys}.\ \forall es \in \mathit{Sys}.\ \mathit{Fair}_{I(\mathit{Sys})}(\varphi^t, t, l)(es),$$

that is

$$\begin{array}{l}
\mathit{Fair}_I^{kb}(\varphi, t, l) =_{\text{def}} \\
\quad \lambda \mathit{Sys}.i\text{-}\textbf{\textit{formula}}(\varphi, I, \mathit{Sys}) \wedge i\text{-}\textbf{\textit{term}}(t, I, \mathit{Sys}) \wedge \\
\qquad \forall es \in \mathit{Sys}.((\forall e' \in es.\ (\mathit{kind}(e') = \mathit{rcv}(l)) \Rightarrow \\
\qquad\quad I(\varphi)(\mathit{Sys}, \mathit{state\ before\ send}(e')) \wedge \mathit{val}(e') = I(t)(\mathit{Sys}, \mathit{state\ before\ send}(e'))) \\
\qquad\quad \wedge ((\exists e@i \in es \wedge \forall e@i \in es.\ \exists e' \succeq_i e.\ I(\neg\varphi)(\mathit{Sys}, \mathit{state\ after\ } e')) \vee \\
\qquad\qquad (\exists e@i \in es \wedge \forall e@i \in es.\ \exists e'.\ \mathit{kind}(e') = \mathit{rcv}(l) \wedge \mathit{send}(e') \succeq_i e) \vee \\
\qquad\qquad (\neg(\exists e@i \in es) \wedge I(\neg\varphi)(\mathit{Sys}, \mathit{initstate}_i))).
\end{array}$$

For example, $\mathit{Fair}_I^{kb}(K_i\varphi, t, l)$ says that every message received on l is given by the term t interpreted at the local state of the sender i, and that i must have known fact φ when it sent this message on l; furthermore, if from some point on i knows that φ holds, then eventually a message is received on l.

As in Section 3.1.4, we assume that message communication satisfies a strong fairness condition. The knowledge-based version of the condition $\mathit{FairSend}(l)$ simply associates with each system Sys the specification $\mathit{FairSend}(l)$; that is, $\mathit{FairSend}^{kb}(l)$ is just $\lambda \mathit{Sys}.\, \forall es \in \mathit{Sys}.\mathit{FairSend}(l)(es)$.

Lemma 3.2.6: *For all interpretations I such that φ is an i-formula and t is an i-term in I, and all actions a, we have that*

$$Fair\text{-}Pg(\varphi,t,l,a) \models\!\!\approx_I \mathit{FairSend}^{kb}(l) \Rightarrow Fair_I^{kb}(\varphi,t,l).$$

The proof is similar in spirit to that of Lemma 3.2.4; by supplying a system Sys as an argument to the specification, we essentially reduce to the situation in Lemma 3.1.8. We leave details to the reader.

We can also prove the following analogue of Lemma 3.1.11.

Lemma 3.2.7: *For all interpretations I such that φ is an i-formula, φ' is a j-formula, t is an i-term, and t' is a j-term in I, all distinct links l and l', and all distinct actions a and a', we have that*

$$\begin{aligned}&Fair\text{-}Pg(\varphi,t,l,a) \oplus Fair\text{-}Pg(\varphi',t',l',a') \models\!\!\approx_I\\ &\quad(\mathit{FairSend}^{kb}(l) \wedge \mathit{FairSend}^{kb}(l')) \Rightarrow (Fair_I^{kb}(\varphi,t,l) \wedge Fair_I^{kb}(\varphi',t',l')).\end{aligned}$$

3.3 The sequence transmission problem (STP)

In this section, we give a more detailed example of how a program satisfying a knowledge-based specification X can be extracted from X using the Nuprl system. We do the extraction in two stages. In the first stage, we use Nuprl to prove that the specification is satisfiable. The proof proceeds by refinement: at each step, a rule or tactic (i.e., a sequence of rules invoked under a single name) is applied, and new subgoals are generated; when there are no more subgoals to be proved, the proof is complete. The proof is automated, in the sense that subgoals are generated by the system upon tactic invocation. From the proof, we can extract a knowledge-based program Pg^{kb} that satisfies the specification. In the second stage, we find standard programs that implement Pg^{kb}. This two-stage process has several advantages:

- A proof carried out to derive Pg^{kb} does not rely on particular assumptions about how knowledge is gained. Thus, it is potentially more intuitive and elegant than a proof based on certain implementation assumptions.
- By definition, if Pg^{kb} satisfies a specification, then so do all its implementations.
- This methodology gives us a general technique for deriving standard programs that implement the knowledge-based program, by finding weaker (non-knowledge-based) predicates that imply the knowledge preconditions in Pg^{kb}.

We illustrate this methodology by applying it to one of the problems that has received considerable attention in the context of knowledge-based programming, *the sequence transmission problem* (STP).

3.3.1 Synthesizing a knowledge-based program for STP

The STP involves a sender S that has an input tape with a (possibly infinite) sequence $X = X(\mathit{0}), X(\mathit{1}), \ldots$ of bits, and wants to transmit X to a receiver R; R must write this sequence on an output tape Y. (Here we assume that $X(n)$ is a bit only for simplicity; our analysis of the STP does not essentially change once we allow $X(n)$ to be an element of an arbitrary constructive domain.) A solution to the STP must satisfy two conditions:

1. (safety): at all times, the sequence Y of bits written by R is a prefix of X, and
2. (liveness): every bit $X(n)$ is eventually written by R on the output tape.

Halpern and Zuck [50] give two knowledge-based programs that solve the STP, and show that a number of standard programs in the literature, like Stenning's [81] protocol, the alternating bit protocol [8], and Aho, Ullman and Yannakakis's algorithms [2], are all particular instances of these programs.

If messages cannot be lost, duplicated, reordered, or corrupted, then S could simply send the bits in X to R in order. However, we are interested in solutions to the STP in contexts where communication is not reliable. It is easy to see that if undetectable corruption is allowed, then the STP is not solvable. Neither is it solvable if all messages can be lost. Thus, following [50], we assume (a) that all corruptions are detectable and (b) a strong fairness condition: for any given link l, if infinitely often a message is sent on l, then infinitely often some message is delivered on l. We formalize strong fairness by restricting to systems where $\mathit{FairSend}(l)$ holds for all links l.

The safety and liveness conditions for STP are run-based specifications. As argued by Fagin et al. [26], it is often better to think in terms of knowledge-based specifications for this problem. The real goal of the STP is to get the receiver to know the bits. Writing $K_R(X(n))$ as an abbreviation for $K_R(X(n) = 0) \vee K_R(X(n) = 1)$, we really want to satisfy the knowledge-based specification

$$\varphi_{\mathit{stp}}^{\mathit{kb}} =_{\mathrm{def}} \forall n \, \Diamond K_R(X(n)).$$

This is the specification we now synthesize.

Since we are assuming fairness, S can ensure that R learns the nth bit by sending it sufficiently often. Thus, S can ensure that R learns the n^{th} bit if, infinitely often, either

S sends $X(n)$ or S knows that R knows $X(n)$. (Note that once S knows that R knows $X(n)$, S will continue to know this, since local states encode histories.) We can enforce this by using an appropriate instantiation of $Fair^{kb}$.

Let $\mathbf{c}_S$ be a (nonrigid) constant that, intuitively, represents the smallest n such that S does not know that R knows $X(n)$, if such an n exists. That is, we want the following formula to be true:

$$\forall n.\, \forall k < n.\, K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S.$$

Let φ_S be the knowledge-based formula that holds at a consistent cut c if and only if there exists a smallest n such that, at c, S does not know that R knows $X(n)$:

$$\varphi_S =_{\text{def}} \exists n.\, \forall k < n.\, K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)).$$

Let t_S be the term $\langle \mathbf{c}_S, X(\mathbf{c}_S)\rangle$.[4] Let l_{SR} denote the communication link from S to R. Now consider the knowledge-based specification $Fair_I^{kb}(\varphi_S, t_S, l_{SR})$. $Fair_I^{kb}(\varphi_S, t_S, l_{SR})$ holds in a system Sys if, (1) whenever R receives a message from S, the message is a pair of the form $\langle n, X(n)\rangle$; (2) at the time S sent this message to R, S knew that R knew the first n elements in the sequence X, but S did not know whether R knew $X(n)$; and (3) R is guaranteed to either eventually receive the message $\langle n, X(n)\rangle$ or eventually know $X(n)$.

How does the sender learn which bits the receiver knows? One possibility is for S to receive from R a request to send $X(n)$. This can be taken by S to be a signal that R knows all the preceding bits. We can ensure that S gets this information by again using an appropriate instantiation of $Fair^{kb}$. Define $\mathbf{c}_R$ be a (nonrigid) constant that, intuitively, represents the smallest n such that R does not know $X(n)$, if such an n

[4]We are implicitly assuming here that the pairing function that maps x and y to $\langle x, y\rangle$ is in the language.

exists. In other words, we want the following formula to be true:

$$\forall n.\ \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)) \Rightarrow n = \mathbf{c}_R.$$

We take φ_R to be the knowledge-based formula

$$\varphi_R =_{\text{def}} \exists n.\ \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)),$$

which says that there exists a smallest n such that R does not know $X(n)$ (or, equivalently, such that $\mathbf{c}_R = n$ holds). Finally, let l_{RS} denote the communication link from R to S. $Fair_I^{kb}(\varphi_R, t_R, l_{RS})$ implies that whenever S receives a message n from R, it is the case that, at the time R sent this message, R knew the first n elements of X, but not $X(n)$. Note that, for all n, S is guaranteed to eventually receive a message n unless R eventually knows $X(n)$.

We can now use the system to verify our informal claim that we have refined the initial specification φ_{stp}^{kb}. That is, the system can prove

$$\begin{aligned}&(Fair_I^{kb}(\varphi_S, t_S, l_{SR}) \wedge Fair_I^{kb}(\varphi_R, \mathbf{c}_R, l_{RS}) \wedge \\ &\quad (\forall n.\ \forall k < n.\ K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S) \wedge \\ &\quad (\forall n.\ \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)) \Rightarrow n = \mathbf{c}_R)) \Rightarrow \varphi_{stp}^{kb}.\end{aligned}$$

No new techniques are needed for this proof: we simply unwind the definitions of the semantics of knowledge formulas and of the fairness specifications, and proceed with a standard proof by induction on the smallest n such that R does not know $X(n)$.

It follows from Lemma 3.2.7 that $Fair_I^{kb}(\varphi_S, t_S, l_{SR}) \wedge Fair_I^{kb}(\varphi_R, \mathbf{c}_R, l_{RS})$ is satisfied by the combination of two simple knowledge-based programs, assuming that message communication on links l_{SR} and l_{RS} satisfies the strong fairness conditions $FairSend^{kb}(l_{SR})$ and $FairSend^{kb}(l_{RS})$. That is, for any two distinct actions a_S and a_R, the following is true:

$$\begin{aligned}&Fair\text{-}Pg(\varphi_S, t_S, l_{SR}, a_S) \oplus Fair\text{-}Pg(\varphi_R, \mathbf{c}_R, l_{RS}, a_R) \models\!\!\approx_I \\ &\quad (FairSend^{kb}(l_{SR}) \wedge FairSend^{kb}(l_{RS})) \Rightarrow (Fair_I^{kb}(\varphi_S, t_S, l_{SR}) \wedge Fair_I^{kb}(\varphi_R, \mathbf{c}_R, l_{RS})).\end{aligned}$$

As explained in Section 3.1.4, $\mathit{FairSend}^{kb}(l_{SR}) \wedge \mathit{FairSend}^{kb}(l_{RS})$ says that if infinitely often a message is sent on l_{SR} then infinitely often a message is received on l_{SR}, and, similarly, if infinitely often a message is sent on l_{RS} then infinitely often a message is received on l_{RS}; as mentioned at the beginning of this section, we restrict to systems where these conditions are met. Furthermore, it is not difficult to show that we can use simple initialization clauses to guarantee that the constraints on the interpretation of $\mathbf{c}_S$ and $\mathbf{c}_R$ are satisfied:

$$@S \ \textbf{initially} \ \Box \, (\forall n.\, \forall k < n.\ K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S) \approx\!\!\!\models_I \\ \forall n.\, \forall k < n.\ K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S,$$

$$@R \ \textbf{initially} \ \Box \, (\forall n.\, \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)) \Rightarrow n = \mathbf{c}_R) \approx\!\!\!\models_I \\ \forall n.\, \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)) \Rightarrow n = \mathbf{c}_R.$$

Thus, $\mathit{Pg}_S^{kb}(\varphi_S, t_S, l_{SR}, a_S) \oplus \mathit{Pg}_R^{kb}(\varphi_R, \mathbf{c}_R, l_{RS}, a_R)) \approx\!\!\!\models_I \ \varphi_{stp}^{kb}$, where

$$\mathit{Pg}_S^{kb}(\varphi_S, t_S, l_{SR}, a_S) =_{\text{def}} \mathit{Fair}\text{-}\mathit{Pg}(\varphi_S, t_S, l_{SR}, a_S) \oplus \\ @S \ \textbf{initially} \ \Box \, (\forall n.\, \forall k < n.\ K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S),$$

$$\mathit{Pg}_R^{kb}(\varphi_R, \mathbf{c}_R, l_{RS}, a_R)) =_{\text{def}} \mathit{Fair}\text{-}\mathit{Pg}(\varphi_R, \mathbf{c}_R, l_{RS}, a_R) \oplus \\ @R \ \textbf{initially} \ \Box \, (\forall n.\, \forall k < n.\ K_R(X(k)) \wedge \neg K_R(X(n)) \Rightarrow n = \mathbf{c}_R).$$

From the definition of $\mathit{Fair}\text{-}\mathit{Pg}(\varphi_R, \mathbf{c}_R, l_{RS}, a_R)$ in Section 3.2.3, it follows that $\mathit{Pg}_S^{kb}(\varphi_S, t_S, l_{SR}, a_S)$ is the following composition:

$$@S \ \textbf{initially} \ \Box \, (\forall n.\, \forall k < n.\ K_S K_R(X(k)) \wedge \neg K_S K_R(X(n)) \Rightarrow n = \mathbf{c}_S) \oplus \\ @S \ \mathit{kind} = \mathit{local}(a_S) \ \textbf{only if} \ \exists n.\ (\forall k < n.\ K_S K_R(X(k))) \wedge \neg K_S K_R(X(n)) \oplus \\ @S \ \textbf{if} \ \mathit{kind} = \mathit{local}(a_S) \ \textbf{then} \ \textbf{msg}(l_{SR}) \ := t_S \oplus \\ @S \ \textbf{only events in} \ [a_S] \ \textbf{affects} \ \textbf{msg}(l_{SR}) \oplus \\ @S \ \textbf{if necessarily} \ \exists n.\ (\forall k < n.\ K_S K_R(X(k))) \wedge \neg K_S K_R(X(n)) \\ \textbf{then i.o.} \ \mathit{kind} = \mathit{local}(a_S).$$

Using the program notation of Fagin et al. [29], $Pg_S^{kb}(\varphi_S, t_S, l_{SR}, a_S)$ is essentially semantically equivalent to the following collection of programs, one for each value n:

```
if $K_S(K_R X(0) \wedge \ldots \wedge K_R X(n-1)) \wedge \neg K_S K_R X(n)$
    then $send_{l_{SR}}(\langle n, X(n) \rangle)$
    else $skip$.
```

In both of these programs, S takes the same action under the same circumstances, and with the same effects on its local state. That is, given a run r (i.e., a sequence of global states) consistent with the collection of knowledge-based programs, we can construct an event structure es consistent with $Pg_S^{kb}(\varphi_S, t_S, l_{SR}, a_S)$ such that the sequence of local states of S in es, with stuttering eliminated, is the same as in r. The converse is also true. More precisely, in a run r consistent with the collection of knoweldge-based programs, at each point of time, either S knows that R knows the value of $X(n)$ for all n, or there exists a smallest n such that $\neg K_S K_R(X(n))$ holds. In the first case, S does nothing, while in the second case S sends $\langle n, X(n) \rangle$ on l_{SR}. Similarly, in an event structure es consistent with $Pg_S^{kb}(\varphi_S, t_S, l_{SR}, a_S)$, if S knows that R knows $X(n)$ for all n, then S does nothing; if not, then it is impossible for S to know that R knows the first n bits, but never know that R knows $X(n)$, without eventually S taking an a_S action with value $\langle n, X(n) \rangle$. This means that for each run r consistent with the collection of knowledge-based programs, the event structure es in which S starts from the same initial state as in r and performs action a_S as soon as it is enabled has the same sequence of local states of S as r. For each event structure es consistent with $Pg_S^{kb}(\varphi_S, t_S, l_{SR}, a_S)$, in the run r of global states in es with stuttering eliminated, S takes action a_S as soon as enabled; subsequently, r is consistent with the collection of knowledge-based programs.

Similarly, $Pg_R^{kb}(\varphi_R, \mathbf{c}_R, l_{RS}, a_R)$ is essentially semantically equivalent to the following collection of programs, one for each value n:

```
if $K_R X(0) \wedge \ldots\ldots \wedge K_R X(n-1) \wedge \neg K_R X(n)$ then $send_{l_{RS}}(n)$ else $skip$.
```

Thus, the derived program is essentially one of the knowledge-based programs considered by Halpern and Zuck [50]. This is not surprising, since our derivation followed much the same reasoning as that of Halpern and Zuck. However, note that we did not first give a knowledge-based program and then verify that it satisfied the specification. Rather, we derived the knowledge-based programs for the sender and receiver from the proof that the specification was satisfiable. And, while Nuprl required "hints" in terms of what to prove, the key ingredients of the proof, namely, the specification $Fair_I^{kb}(\varphi, t, l)$ and the proof that $Fair\text{-}Pg(\varphi, t, l, a)$ realizes it, were already in the system, having been used in other contexts. Thus, this suggests that we may be able to apply similar techniques to derive programs satisfying other specifications in communication systems with only weak fairness guarantees.

3.3.2 Synthesis of standard programs for STP

This takes care of the first stage of the synthesis process. We now want to find a standard program that implements the knowledge-based program. As discussed by Halpern and Zuck [50], the exact standard program that we use depends on the underlying assumptions about the communications systems. Here we sketch an approach to finding such a standard program.

The first step is to identify the exact properties of knowledge that are needed for the proof. This can be done by inspecting the proof to see which properties of the knowledge operators K_S and K_R are used. The idea is then to replace formulas involving the knowledge operators by standard (non-epistemic formulas) which have the relevant properties.

Suppose that $\tilde{\varphi}_S^{kb}$ is a formula that mentions the function X, has a free variable m, and is guaranteed to be an S-formula in all interpretations I and systems Sys. (Recall that, as noted just before Lemma 3.2.1, there are simple syntactic conditions that guarantee that a formula is an i-formula for all I and Sys.) Roughly speaking, we can think of $\tilde{\varphi}_S^{kb}$ as corresponding to $K_S K_R(X(m))$. Let φ_S^{kb} be an abbreviation of

$$\exists n.\ ((\forall k < n.\ \tilde{\varphi}_S^{kb}[m/k]) \land \neg\tilde{\varphi}_S^{kb}[m/n]).$$

Similarly, suppose that $\tilde{\varphi}_R^{kb}$ is a formula that mentions X, has a free variable m, and is guaranteed to be an R-formula in all interpretations I; let φ_R^{kb} be an abbreviation of

$$\exists n.\ ((\forall k < n.\ \tilde{\varphi}_R^{kb}[m/k]) \land \neg\tilde{\varphi}_R^{kb}[m/n]).$$

Thus, φ_S^{kb} and φ_R^{kb} are the analogues of φ_S and φ_R in Section 3.3.1. While φ_S is a formula that says that there is a least n such that $K_S K_R X(n)$ does not hold, φ_S^{kb} says that there is a least n such that $\tilde{\varphi}_S^{kb}(n)$ does not hold. Similarly, while φ_R^{kb} says that there is a least n such that $K_R X(n)$ does not hold, φ_R^{kb} says that there is a least n such that $\tilde{\varphi}_R^{kb}(n)$ does not hold.

We also use we use constants $\tilde{\mathbf{c}}_S$, and $\tilde{\mathbf{c}}_R$ that are analogues to $\mathbf{c}_S$, $\mathbf{c}_R$; $\tilde{\varphi}_S^{kb}$ plays the same role in the definition of $\tilde{\mathbf{c}}_S$ as $K_S K_R(X(m))$ played in the definition of $\mathbf{c}_S$, and $\tilde{\varphi}_R^{kb}$ plays the same role in the definition of $\tilde{\mathbf{c}}_R$ as $K_R(X(m))$ played in the definition of $\mathbf{c}_R$. Thus, we take $\tilde{\mathbf{c}}_S$ to be a constant that represents the least n such that $\tilde{\varphi}_S^{kb}[m/n]$ does not hold (that is, we want $\exists n.\ \forall k < n.\ \tilde{\varphi}_S^{kb}[m/k] \land \neg\tilde{\varphi}_S^{kb}[m/n] \Rightarrow (n = \tilde{\mathbf{c}}_S)$ to be true), and define $\tilde{t}_S$ as the pair $\langle \tilde{\mathbf{c}_S}, X(\tilde{\mathbf{c}_S})\rangle$, Similarly, we take $\tilde{\mathbf{c}}_R$ to be a constant that represents the least n such that $\tilde{\varphi}_R^{kb}[m/n]$ does not hold (that is, we want $\exists n.\ \forall k < n.\ \tilde{\varphi}_R^{kb}[m/k] \land \neg\tilde{\varphi}_R^{kb}[m/n] \Rightarrow (n = \tilde{\mathbf{c}}_R)$ to be true).

Let $\varphi_{stp}^{kb}(\tilde{\varphi}_R^{kb})$ be the specification that results by using $\tilde{\varphi}_R^{kb}$ instead of K_R in φ_{stp}^{kb}:

$$\varphi_{stp}^{kb}(\tilde{\varphi}_R^{kb}) =_{\text{def}} \forall n.\ \Diamond\tilde{\varphi}_R^{kb}[m/n].$$

We prove the goal $\varphi^{kb}_{stp}(\tilde{\varphi}^{kb}_R)$ by refinement: at each step, a rule (or tactic) of Nuprl is applied, and a number of subgoals (typically easier to prove) are generated; the rule gives a mechanism of constructing a proof of the goal from proofs of the subgoals. Some of the subgoals cannot be further refined in an obvious manner; this is the case, for example, for the simple conditions on $\tilde{\varphi}^{kb}_S$ or $\tilde{\varphi}^{kb}_R$. The new theorem states that, under suitable conditions on $\tilde{\varphi}^{kb}_S$ and $\tilde{\varphi}^{kb}_R$, $\varphi^{kb}_{stp}(\tilde{\varphi}^{kb}_R)$ is satisfiable if both $Fair^{kb}_I(\varphi^{kb}_S, \tilde{t}_S, l_{SR})$ and $Fair^{kb}_I(\varphi^{kb}_R, \tilde{\mathbf{c}}_R, l_{RS})$ are satisfiable.

We now explain the conditions placed on the predicates $\tilde{\varphi}^{kb}_S$ and $\tilde{\varphi}^{kb}_R$. One condition is that $\tilde{\varphi}^{kb}_R$ be ***stable***, that is, once true, it stays true:

$$\begin{aligned} Stable(\tilde{\varphi}^{kb}_R) =_{\text{def}} \quad & \lambda Sys.\, \forall es \in Sys.\, \forall e_R@R \in es.\, \forall n. \\ & \quad I(\tilde{\varphi}^{kb}_R[m/n])(Sys, state\ before\ e_R) \Rightarrow \\ & \quad\quad I(\tilde{\varphi}^{kb}_R[m/n])(Sys, state\ after\ e_R). \end{aligned}$$

Assuming $Stable(\tilde{\varphi}^{kb}_R)$ allows us to prove φ^{kb}_R by induction on the least index n such that $\neg\tilde{\varphi}^{kb}_R[m/n]$ holds.

To allow us to carry out a case analysis on whether $\tilde{\varphi}^{kb}_R$ holds, we also assume that $\tilde{\varphi}^{kb}_R$ satisfies the principle of excluded middle; that is, we assume that $Determinate(\tilde{\varphi}^{kb}_R) =_{\text{def}} Determinate(\forall n.\ (\tilde{\varphi}^{kb}_R[m/n])^t)$. For similar reasons, we also restrict $\tilde{\varphi}^{kb}_S$ to being stable and determinate; that is, we require that $Stable(\tilde{\varphi}^{kb}_S)$ and $Determinate(\tilde{\varphi}^{kb}_S)$ both hold.

The third condition we impose establishes a connection between $\tilde{\varphi}^{kb}_S$ and $\tilde{\varphi}^{kb}_R$, and ensures that, for all values n, if $\tilde{\varphi}^{kb}_S[m/n]$ holds, then eventually $\tilde{\varphi}^{kb}_R[m/n]$ will also hold:

$$\begin{aligned} Implies(\tilde{\varphi}^{kb}_S, \tilde{\varphi}^{kb}_R) =_{\text{def}} \quad & \lambda Sys.\, \forall es \in Sys.\, \forall n.\, \forall e_S@S \in es. \\ & \quad I(\tilde{\varphi}^{kb}_S[m/n])(Sys, state\ before\ e_S) \Rightarrow \\ & \quad\quad \exists e_R \succ e_S@R \in es.\ I(\tilde{\varphi}^{kb}_R[m/n])(Sys, state\ after\ e_R). \end{aligned}$$

To explain the next condition, recall that $\tilde{\varphi}_R$ is meant to represent $K_R(X(m))$. With this interpretation, $I(\forall k \leq n.\ \tilde{\varphi}_R^{kb}[m/k])(Sys, \mathit{state\ before\ send}(e_S))$ says that R knows the first n bits before it sends a message to S. We would like it to be the case that, just as with the knowledge-based derivation, when S receives R's message, S knows that R knows the n^{th} bit. Since we think of $\tilde{\varphi}_S^{kb}$ as saying that $K_S K_R(X(m))$ holds, we expect $I(\tilde{\varphi}_S^{kb}[m/n])(Sys, \mathit{state\ after}\ e_S)$ to be true. Define $Rcv(\tilde{\varphi}_S^{kb}, \tilde{\varphi}_R^{kb}, l_{RS})$ to be an abbreviation of

$$\begin{aligned}&\lambda Sys.\forall es \in Sys.\ \forall e_S@S \in es.\ (kind(e_S) = rcv(l_{RS})) \Rightarrow\\ &\qquad \forall n.\ (\forall k \leq n.\ I(\tilde{\varphi}_R^{kb}[m/n])(Sys, \mathit{state\ after\ send}(e_S))) \Rightarrow\\ &\qquad I(\tilde{\varphi}_S^{kb}[m/n])(Sys, \mathit{state\ after}\ e_S).\end{aligned}$$

With this background, we can describe the last condition. Intuitively, it says that if n is the least value for which $\tilde{\varphi}_S^{kb}$ fails when S sends a message to R, then $\tilde{\varphi}_R^{kb}$ holds for n upon message delivery:

$$\begin{aligned}&Rcv(\tilde{\varphi}_R^{kb}, \tilde{\varphi}_S^{kb}, l_{SR}) \ \equiv\\ &\lambda Sys.\ \forall es \in Sys.\ \ \forall e_R@R \in es.\ (kind(e_R) = rcv(l_{SR})) \Rightarrow\\ &\ \forall n.\ (I(\varphi_S^{kb}[m/n])(Sys, \mathit{state\ before\ send}(e_R)) \Rightarrow\\ &\qquad I(\tilde{\varphi}_R^{kb}[m/n])(Sys, \mathit{state\ after}\ e_R)).\end{aligned}$$

We denote the conjunction of these conditions as $\psi^{kb}(\tilde{\varphi}_S^{kb}, \tilde{\varphi}_R^{kb}, \tilde{t}_S, \tilde{\mathbf{c}}_R, l_{SR}, l_{RS})$. The new theorem says

$$\begin{aligned}&\psi^{kb}(\tilde{\varphi}_S^{kb}, \tilde{\varphi}_R^{kb}, \tilde{t}_S, \tilde{\mathbf{c}}_R, l_{SR}, l_{RS}) \wedge\\ &Fair_I^{kb}(\varphi_S^{kb}, \tilde{t}_S, l_{SR}) \wedge Fair_I^{kb}(\varphi_R^{kb}, \tilde{\mathbf{c}}_R, l_{RS}) \wedge\\ &\forall n.\ \forall k < n.\ \tilde{\varphi}_S^{kb}[m/k] \wedge \neg\tilde{\varphi}_S^{kb}[m/n] \Rightarrow (n = \tilde{\mathbf{c}}_S) \wedge\\ &\forall n.\ \forall k < n.\ \tilde{\varphi}_R^{kb}[m/k] \wedge \neg\tilde{\varphi}_R^{kb}[m/n] \Rightarrow (n = \tilde{\mathbf{c}}_R)\\ &\qquad \Rightarrow \varphi_{stp}^{kb}(\tilde{\varphi}_R^{kb}).\end{aligned}$$

We can prove that the following is true for any two distinct actions a_S and a_R:

$$
\begin{aligned}
&Pg_S^{kb}(\varphi_S^{kb}, \tilde{t}_S, l_{SR}, a_S) \oplus Pg_R^{kb}(\varphi_R^{kb}, \tilde{\mathbf{c}}_R, l_{RS}, a_R) \mathrel{|\!\!\approx}_I \\
&\quad \psi^{kb}(\tilde{\varphi}_S^{kb}, \tilde{\varphi}_R^{kb}, \tilde{t}_S, \tilde{\mathbf{c}}_R, l_{SR}, l_{RS}) \wedge \mathit{FairSend}^{kb}(l_{RS}) \wedge \mathit{FairSend}^{kb}(l_{SR}) \Rightarrow \varphi_{stp}^{kb}(\tilde{\varphi}_R^{kb}),
\end{aligned}
$$

where

$$
\begin{aligned}
&Pg_S^{kb}(\varphi_S^{kb}, \tilde{t}_S, l_{SR}, a_S) =_{\text{def}} \\
&\quad \mathit{Fair\text{-}Pg}(\varphi_S^{kb}, \tilde{t}_S, l_{SR}, a_S) \oplus \\
&\quad @S \textbf{ initially } \Box\, (\forall n.\ \forall k < n.\ \tilde{\varphi}_S^{kb}[m/k] \wedge \neg\tilde{\varphi}_S^{kb}[m/n] \Rightarrow n = \tilde{\mathbf{c}}_S),
\end{aligned}
$$

$$
\begin{aligned}
&Pg_R^{kb}(\varphi_R^{kb}, \tilde{\mathbf{c}}_R, l_{RS}, a_R)) =_{\text{def}} \\
&\quad \mathit{Fair\text{-}Pg}(\varphi_R^{kb}, \tilde{\mathbf{c}}_R, l_{RS}, a_R) \oplus \\
&\quad @R \textbf{ initially } \Box\, (\forall n.\ \forall k < n.\ \tilde{\varphi}_R^{kb}[m/k] \wedge \neg\tilde{\varphi}_R^{kb}[m/n] \Rightarrow n = \tilde{\mathbf{c}}_R).
\end{aligned}
$$

In particular, for the terms t_S and $\mathbf{c}_R$ defined in the previous section, we can show that $\psi^{kb}(K_S K_R X(m),\ K_R X(m), t_S, \mathbf{c}_R, l_{SR}, l_{RS})$ is true. Thus, the new theorem is indeed a generalization of the previous results.

The formulas $K_S K_R X(m)$ and $K_R X(m)$ are not the only ones that satisfy these conditions. Most importantly for the purpose of extracting standard programs, the conditions are satisfied by non-epistemic formulas, that is, formulas whose interpretations do not depend on the entire system, just on the local states of the sender or the receiver agents, respectively. Note that Lemma 3.1.12 guarantees that the extracted program is consistent.

For example, we can take x_S to be an S-local variable that stores the largest n such that S has received requests from R for all of the first n bits in X; that is, initially x_S is set to to -1, and if S receives a request for $X(n)$ and $x_S = n - 1$, then x_S is set to n. We similarly take x_R to be an R-local variable that stores the largest n such that R has received the first $n - 1$ bits of X from S . That is, initially, x_R is set to 0; if

R receives a message of the form $\langle n, msg\rangle$ from S and $x_R = n-1$, then x_R is set to n. We have in mind a setting in which the receiver requests bits from the sender in order, that is, R starts by requesting $X(0)$, and does not request $X(n)$ before receiving $X(n-1)$. Similarly, the aim is to have the sender send bits in order; that is, S does not not send $X(n+1)$ before S knows that $X(n)$ has been received. For the generalized knowledge-based formulation of the STP problem, we use a formula $\tilde{\varphi}_S^{kb}$ that is meant to correspond to $K_S K_R X(m)$. Given the intuition above, at all times, S knows that R has received bits $0, \ldots, x_S - 1$; that is, $\tilde{\varphi}_S^{kb}(m)$ holds here iff $x_S > m$. Since φ_S^{kb} holds iff there is a least m such that $\neg\tilde{\varphi}(m)$ holds, φ_S^{kb} is vacuously *true* here. Moreover, the term $\tilde{\mathbf{c}}_s$ that represents the least such m is just x_S. Similarly, the formula $\tilde{\varphi}_R^{kb}$ is meant to correspond to $K_R X(m)$; here we can take $\tilde{\varphi}_R^{kb}$ to be $x_R > m$. Again, φ_R^{kb} becomes vacuously *true*, and the corresponding term $\tilde{c}_R$, which represents the least m such that $\tilde{\varphi}_R^{kb}$ does not hold, is just x_R.

It is not hard to show that $\psi^{kb}(x_S > m x_R > m, \langle x_S, X(x_S)\rangle, x_R, l_{SR}, l_{RS})$ holds in the system generated by $Pg(\textit{true}, \langle x_S, X(x_S)\rangle, l_{SR}, a_S) \oplus Pg(\textit{true}, x_R, l_{RS}, a_R)$, for any distinct actions a_S and a_R; this specification is not knowledge-based. Recall that $\varphi_{stp}^{kb}(\tilde{\varphi}_R^{kb})$ is $\forall n.\diamond\tilde{\varphi}_R^{kb}[m/n]$. In this context, it is the formula $\forall n.\diamond(x_R = n)$. In addition, $\varphi_{stp}^{kb}(x_R > m)$ implies φ_{stp}^{kb} (in the system generated by $Pg(\textit{true}, \langle x_S, X(x_S)\rangle, l_{SR}, a_S) \oplus Pg(\textit{true}, x_R, l_{RS}, a_R)$), so if message communication is fair,

$$Pg(\textit{true}, \langle x_S, X(x_S)\rangle, l_{SR}, a_S) \oplus Pg(\textit{true}, x_R, l_{RS}, a_R)$$

satisfies the STP specification, as long as a_S and a_R are distinct actions. We can easily give a justification for this result: If R follows $Pg(\textit{true}, x_R, l_{RS}, a_R)$ and S follows $Pg(\textit{true}, \langle x_S, X_S(x_S), l_{SR}, a_S)$, then R starts by sending message 0 to S; since communication is fair, eventually S receives this message, and x_S is set to 0; S starts sending $\langle 0, X(0)\rangle$ to R; since communication is fair, eventually R receives this message, so R sets x_R to 1, stops sending message 0 to S, and starts sending message 1 to S. It is not difficult to show that, for all values n, there is a time when x_R is set to n, which triggers R to send message n to S. S eventually receives this message and starts sending $\langle n, X(n)\rangle$ to R. This, in turn, ensures that R eventually receives this message and thus learns $X(n)$. Note that the program $Pg(\textit{true}, \langle x_S, X_S(x_S)\rangle, l_{SR}, a_S) \oplus Pg(\textit{true}, x_R, l_{RS}, a_R)$ is realizable. We have thus extracted a standard program that realizes the STP specification. In fact, the program turns out to be essentially equivalent to Stenning's [81] protocol.

The key point here is that by replacing the knowledge tests by weaker predicates that imply them and do not explicitly mention knowledge, we can derive standard programs that implement the knowledge-based program. We believe that other standard implementations of the knowledge-based program can be derived in a similar way.

3.4 Remarks

We have shown that the mechanism for synthesizing programs from specifications in Nuprl can be extended to knowledge-based programs and specifications and that axioms much in the spirit of those used for standard programs can be used to synthesize knowledge-based programs as well. We applied this methodology to the analysis of the sequence transmission problem and showed that the knowledge-based programs proposed by Halpern and Zuck for solving the STP problem can be synthesized in Nuprl. We also sketched an approach for deriving standard programs that implement the knowledge-based programs that solve the STP. A feature of our method is that the extracted standard programs are close to the type of pseudocode designers write their programs in, and can be translated into running code.

There has been work on synthesizing both standard programs and knowledge-based programs from knowledge-based specifications. In the case of synchronous systems with only one process, Van der Meyden and Vardi [83] provide a necessary and sufficient condition for a certain type of knowledge-based specification to be realizable, and show that, when it holds, a program can be extracted that satisfies the specification. Still assuming a synchronous setting, but this time allowing multiple agents, Engelhardt, van der Meyden, and Moses [23, 24] propose a refinement calculus in which one can start with an epistemic and temporal specification and use refinement rules that eventually lead to standard formulas. The refinement rules annotate formulas with preconditions and postconditions, which allow programs to be synthesized from the leaf formulas. A search up the tree generated in the refinement process suffices to build a program that satisfies the specification. The extracted programs are objects of a programming language that allows concurrent and sequential executions, variable assignments, loops and conditional statements. We see their work as complementary to ours. Since our work is based on Nuprl, we are able to take advantage of the library of tactics provided by Nuprl to be able to generate proofs. The expressive power of Nuprl also allows us to express all the high-level concepts of interest (both epistemic and temporal) easily. We see our method for synthesizing programs from knowledge-based specifications as an alternative to this approach. As in the Engelhart et al. approach, the programs extracted in Nuprl are close to realistic programming languages. Arguably, distributed I/O message automata are general enough to express most of the distributed programs of interest when communication is done by message passing. Our approach has the additional advantage of working in asynchronous settings. The refinement rules proposed by Engelhardt, van der Meyden, and Moses can be captured as tactics in Nuprl, though we have not formalized them in the system yet.

CHAPTER 4

A KNOWLEDGE-BASED ANALYSIS OF INFORMATION-FLOW PROPERTIES

4.1 Overview

Secrecy has often been cast as lack of information flow between domains at different levels of security [35, 36, 66, 75, 85]. Typically, a system is considered to exhibit an insecure information flow if a user at a lower level of security can gain information about the activity at higher levels of security. Many variations on this theme are possible, depending on what type of information is considered sensitive (e.g., should unauthorized users be prevented from learning whether some confidential event has ever occurred, or is it the non-occurrence of such events that has to be kept secret), and on the capabilities of the unauthorized users.

A framework for studying information-flow properties should allow (1) natural formalizations of properties classically accepted as building blocks for expressing more complex requirements, (2) analyzing whether a given system (typically thought of as arising from the executions of a protocol) satisfies a given information-flow restriction, as well as (3) the development of systems and protocols that provably preclude information flows. One such framework that has gained widespread acceptance is the Modular Assembly Kit (MAKS) proposed by Mantel [62]. MAKS has been shown to fulfill all three of these desiderata. It can express a number of well-known information-flow constraints [64]. Thanks to its modularity, complex information-flow properties can be built from basic security predicates. Thus, the problem of analyzing whether a system satisfies a complex security property reduces to verifying that the system satisfies all the basic security predicates that make up the complex property [51]. Proof techniques and automatic provers have been developed for the framework. Recent research has mainly focused on extending the framework to cryptographic systems [53], and on decomposing global confidentially requirements into local confidentiality requirements, in order to construct secure systems from secure components [79].

It has been long recognized that information-flow properties have simple intuitive reformulations as restrictions on what users at lower levels of security may know about users at higher levels of security. Formally, these restrictions can be expressed as epistemic formulas [25]. Such epistemic representations of information-flow properties have the advantage of simplicity and closeness to natural-language formulations [47]. Once we have a logical representation of information-flow properties, known techniques for proof refinement and program synthesis can be applied to deal with desiderata (2) and (3) in the context of information-flow analysis.

In this chapter, we focus on some of the building blocks of the MAKS framework and formally analyze the extent to which they match the informal explanations typically given for them. A case-by-case analysis shows that there are subtle differences. We show that simple properties that preclude lack of information about occurrences or non-occurrences of confidential events, like Backwards (Strict) Insertion of Confidential Events or Deletion of Confidential Events, are actually more involved than usually thought. In particular, they preclude a low-level user (better thought of as an *adversary*) from knowing whether confidential events have occurred, even if he is told about all events (including the confidential ones) in the past. Techniques for reasoning about counterfactual situations [43], and about multiple adversaries, extending the work of Halpern and Pucella [49], can be applied to formally capture the properties above as knowledge-based requirements.

The reformulations of MAKS-style information-flow properties as knowledge-based formulas can be naturally extended to cryptographic settings, where agents' inability to distinguish two encrypted messages is taken into account. Interestingly, the natural extensions of both Backwards (Strict) Insertion of Confidential Events or Deletion of Confidential Events to cryptosystems differ slightly from the notions proposed by Hutter and Schairer [53]. We show that the difference lies in the assumptions regarding the adversary's capabilities: Hutter and Schairer (HS from now on) implicitly assume a very weak adversary that does not keep track of the messages received in the past and does not perform any operation on the received messages in order to compute a set of available keys. A knowledge-based formulation of information-flow properties has the advantage of making explicit the connection between adversary's knowledge and his capabilities; this allows us to express a number of variants of HS definitions, making different assumptions about the adversary's capabilities.

Relating the MAKS approach and the epistemic approach in a principled way benefits both. Further work is needed to express all MAKS-style properties as epistemic formulas in our logic. This could potentially identify other subtle differences between the formal definitions and their informal explanations, and suggest variants of these properties that also make sense.

In Section 4.2 we present the syntax of first order epistemic logic and its semantics with respect to event systems. We proceed in Section 4.3 with a review of some of the basic event-system definitions of information-flow properties, and show how these properties can be expressed as knowledge-based formulas. In Sections 4.4.1 and 4.4.2 we explain the need for reasoning about different types of adversaries and counterfactual situations. More involved MAKS-style properties and their corresponding characterizations as knowledge-based formulas are discussed in Section 4.4.3. In Section 4.5 we extend the knowledge-based characterizations to cryptographic systems, and relate our formalism with the one proposed by Hutter and Schairer.

4.2 Epistemic logic interpreted in event systems

4.2.1 Event systems

We assume a fixed and finite set $\mathcal{A}$ of agents, typically the agents participating in a given protocol. As usual [55, 59, 66, 70], changes in the local states of agents, most often triggered by agents taking certain actions, are modeled as *events*. The behavior of each agent i in $\mathcal{A}$ is described as a set Tr_i of finite sequences (also called *traces*) of events over a set E_i. Events in a trace are recorded in the order in which they have occurred; there is, however, no precise information on the time each event in a trace has occurred. Among events in E_i, some are considered *confidential*, i.e., to be kept secret; some are considered *visible*, i.e., their occurrence can be detected by a passive observer; and the rest are considered *neutral*. Agent i's *view*, denoted $\mathcal{V}_i$, is simply a partition (C_i, V_i, N_i) of E_i into confidential events C_i, visible events V_i, and neutral events N_i. (We follow here the standard approach [62, 88] and consider a fixed, context-independent classification of events into confidential, neutral and visible events. In general, the classification may depend on context; for example, an event may be confidential at some points in a trace, and visible at other points of the trace, say for example after a declassification event occurs. We leave the context-dependent case to future research.)

Since we focus here on message-passing systems, that is, systems in which agents communicate by exchanging messages, E_i may contain events corresponding to agent i sending or receiving messages on a certain link. We assume that, when agent i sends a message to agent j, the message is placed in a buffer (or queue) on the communication link between i and j; agent j is said to receive the message once he consumes (dequeues) the message from the communication buffer. We denote the event of agent i sending message msg to agent j as $send_{i,j}(msg)$, and the event of agent i receiving message msg as $recv_i(msg)$ [1]. The set of queuing and dequeuing events on the communication link between i and j is denoted $Comm_{i,j}$. We assume that $Comm_{i,j}$ and $Comm_{i,k}$ are disjoint for all $j \neq k$. The set of all communication events involving agent i (i.e., $\bigcup_{j \in \mathcal{A}} Comm_{i,j}$) is denoted $Comm_i$. Note that we are not imposing the restriction that events in E_i and E_j for distinct agents i and j be disjoint.

Information-flow properties are typically studied in settings where there are two agents (see [63, 66, 67, 68]): a low-level user L and a high-level user H. We can model this by taking H's view of the system to be $\mathcal{V}_H = (C, V, N)$, and L's view of the system to be $\mathcal{V}_L = (\emptyset, V, \emptyset)$, formalizing the assumption that all confidential and neutral events are associated with the high-level user H, and all events visible to L are also visible to H.

The set of all events in a system with agents $\mathcal{A}$ is $E = \bigcup_{i \in \mathcal{A}} E_i$. We define the ***global view*** $\mathcal{V}$ (also called an $\mathcal{A}$-view) as the tuple $\langle \mathcal{V}_i : i \in \mathcal{A} \rangle$. A sequence of events in E is simply an element of the set E^*. The global behavior of the system is modeled as a set of traces Tr over events in E, best thought of as arising from the execution of a protocol. We use the notation $\tau' \leq \tau$ (resp., $\tau' < \tau$) to indicate that τ' is a prefix (resp., strict prefix) of trace τ. We assume that all traces are finite and Tr is closed under prefixes. For a given subset E' of events in E, the sequence of events in E' occurring in a sequence τ (preserving the order of occurrence) is denoted as $\tau|_{E'}$; for example, $\tau|_{V_i}$ is the sequence of agent i's visible events in the sequence τ.

[1]Note that the sender is not specified in $recv_i(msg)$; it is possible the receiver may not be able to determine the sender of the message. We can model cases when the receiver knows the sender of a message by assuming that msg is signed by the sender.

The set of traces Tr represents the *composition* of the sets of traces Tr_X and Tr_Y, for two disjoint sets X and Y of agents, if Tr is precisely the set of traces whose projection over the set of events E_X and E_Y associated with agents X, and Y respectively, are the traces in Tr_X, and Tr_Y respectively; that is, τ belongs to Tr if and only if there exist τ_X in Tr_X and τ_Y in Tr_Y such that $\tau|_{E_X} = \tau_X$ and $\tau_{E_Y} = \tau_Y$ (where E_X and E_Y represent the set of all events associated with agents in X and Y, respectively). Note that this definition ensures that, for all pairs of executions τ_X in Tr_X and τ_Y in Tr_Y, we can find an execution τ in Tr such that $\tau|_{E_X} = \tau_X$ and $\tau|_{E_Y} = \tau_Y$, as long as τ_X and τ_Y coincide on the common events, that is, $\tau_X|_{E_X \cap E_Y} = \tau_Y|_{E_X \cap E_Y}$.

4.2.2 Syntax

We consider a standard first-order logic of knowledge and time. Formulas are built up from a set P of predicate symbols on events, a set Var, and a set C of constants with one constant symbol $\mathbf{e}$ for each event e in E. (In the sequel, we often do not distinguish between the syntactic symbol $\mathbf{e}$ and the semantic e; it will be clear to which we refer.) For simplicity, we assume that all predicates have arity either 0 or 1, and that P contains unary predicates $conf_i$, one for each agent i in $\mathcal{A}$, and the predicate occ. To each agent i, we associate a knowledge operator K_i; we read $K_i\varphi$ as "agent i knows fact φ". To each set X of agents, we associate the symbols E_X and D_X, and read $E_X\varphi$ (resp., $D_X\varphi$) as "everybody in the group X knows φ" (resp., "φ is distributed knowledge among the agents in X"). Formulas are closed under negation $\neg$, conjunction $\wedge$, the knowledge operators K_i, E_X and D_X, universal and existential quantification, and the temporal operators $\bigcirc$ ("next"), $\Diamond$ ("eventually"), $\Box$ ("always"), $\ominus$ ("previously"), $\diamondminus$ ("some time in the past"), $\boxminus$ ("always in the past"), $\mathcal{S}$ ("since"), and $\mathcal{A}$ ("immediately after"). $\varphi\,\mathcal{S}\,\psi$ means that ψ held at some point in the past, and since the last time it held,

φ held also; $\varphi\ \mathcal{A}\ \psi$ means that ψ held at some point in the past, and φ held at the step after. As it is standard, we take $\Box\varphi =_{\text{def}} \neg\Diamond\neg\varphi$, and $\boxminus\varphi =_{\text{def}} \neg\diamondminus\neg\varphi$. We also take $P_i\varphi =_{\text{def}} \neg K_i\neg\varphi$, and read it as "agent i thinks φ possible". Note that $\diamondminus\varphi = \textit{true}\ \mathcal{S}\ \varphi$.

4.2.3 Semantics

Let ν be a valuation that associates to each variable x in Var an event in the set E. Let I be a function that associates to each constant symbol $\mathbf{e}$ an event in E. For simplicity, throughout this chapter we assume that I is fixed. We define $I_\nu(t)$ to be $I(t)$ if t is a constant symbol, or $\nu(t)$ if t is a variable. Let π be a mapping from views to functions that map a trace τ in Tr and a predicate symbol in P of arity k (where $k \in \{0,1\}$) to a function mapping E^k to $\{\textit{true}, \mathit{false}\}$. For an event $e \in E$ we take $\pi(\mathcal{V})(\tau, \mathit{conf}_i)(e)$ to be ***true*** if and only if, according to $\mathcal{V}$, e is one of i's confidential events, that is, e is an element of the set C_i recorded in $\mathcal{V}_i$. (Notice that the meaning of predicate conf_i does not depend on the trace τ.) We take $\pi(\mathcal{V})(\tau, \mathit{occ})(e)$ to be ***true*** if and only if e is the last event in τ, that is, there exists trace γ such that $\tau = \gamma \cdot \langle e\rangle$. In the following, we refer to π as an *interpretation*.

Let τ and τ' be two traces in Tr, and let i, j be agents in $\mathcal{A}$. Let $\nu\,[x/e]$ be a valuation just like ν, except that $\nu(x) = e$ holds. An *interpreted* system $\mathcal{I}$ is a tuple $(E, Tr, \mathcal{V}, \pi)$ consisting of a set E of events, a set Tr of traces with events in E, a view $\mathcal{V}$, and an interpretation π. We define what it means for formula φ to be true with respect to a system $\mathcal{I}$, a trace τ, and valuation ν, written $(\mathcal{I}, \tau, \nu) \models \varphi$, inductively:

- if p is a predicate symbol in P of arity 0, then $(\mathcal{I}, \tau, \nu) \models p$ iff $\pi(\mathcal{V})(\tau, p) = \mathit{true}$; if p is a predicate symbol in P of arity 1, and t is a term, then $(\mathcal{I}, \tau, \nu) \models p(t)$ iff $\pi(\mathcal{V})(\tau, p)(I_\nu(t)) = \mathit{true}$
- $(\mathcal{I}, \tau, \nu) \models \neg\varphi$ iff $(\mathcal{I}, \tau, \nu) \not\models \varphi$

- $(\mathcal{I}, \tau, \nu) \models \varphi_1 \wedge \varphi_2$ iff $(\mathcal{I}, \tau, \nu) \models \varphi_1$ and $(\mathcal{I}, \tau, \nu) \models \varphi_2$
- $(\mathcal{I}, \tau, \nu) \models \forall x.\, \varphi$ iff $(\mathcal{I}, \tau, \nu[x/e]) \models \varphi$ for all events e in E
- $(\mathcal{I}, \tau, \nu) \models \bigcirc\varphi$ iff there exists an event e such that $\tau\cdot\langle e\rangle \in Tr$ and $(\mathcal{I}, \tau\cdot\langle e\rangle, \nu) \models \varphi$; $(\mathcal{I}, \tau, \nu) \models \circleddash\varphi$ iff there exist a trace τ' and event e such that $\tau = \tau' \cdot \langle e\rangle$ and $(\mathcal{I}, \tau', \nu) \models \varphi$
- $(\mathcal{I}, \tau, \nu) \models \Diamond\varphi$ iff there exists a trace $\tau' \in Tr$ such that $\tau \leq \tau'$ and $(\mathcal{I}, \tau', \nu) \models \varphi$; $(\mathcal{I}, \tau, \nu) \models \diamondminus\varphi$ iff there exists τ' with $\tau' \leq \tau$ and $(\mathcal{I}, \tau', \nu) \models \varphi$
- $(\mathcal{I}, \tau, \nu) \models \varphi_2\, \mathcal{A}\, \varphi_1$ iff there exist τ_1, e, τ_2 such that $\tau = \tau_1 \cdot \langle e\rangle \cdot \tau_2$, $(\mathcal{I}, \tau_1, \nu) \models \varphi_1$, $(\mathcal{I}, \tau_1 \cdot \langle e\rangle, \nu) \models \varphi_2$ and $(\mathcal{I}, \tau_1 \cdot \langle e\rangle \cdot \tau_2', \nu) \not\models \varphi_1$ for all $\tau_2' \leq \tau_2$
- $(\mathcal{I}, \tau, \nu) \models \varphi_2\, \mathcal{S}\, \varphi_1$ iff there exist τ_1, τ_2 such that $\tau = \tau_1 \cdot \tau_2$, $(\mathcal{I}, \tau_1, \nu) \models \varphi_1$, and $(\mathcal{I}, \tau_1 \cdot \tau_2', \nu) \models \varphi_2 \wedge \neg\varphi_1$ for all τ_2' with $\tau_2' \leq \tau_2$ and $\tau_2' \neq \langle\rangle$.

Note that we use here a *branching time* operator (see [22]): since there is no explicit notion of time in the event systems formalism, $\bigcirc\varphi$ does not mean that φ holds at the next point in time; when interpreted with respect to a trace τ, $\bigcirc\varphi$ holds when there exists an event e that can occur next (i.e., immediately after the last event in τ), and φ holds after e occurs (i.e., φ holds when interpreted with respect to $\tau \cdot \langle e\rangle$).

Traces τ and τ' are *indistinguishable to agent* i, written $\tau \sim_i \tau'$, if and only if i's sequence of events is the same in τ and τ', that is, $\tau|_{E_i} = \tau'|_{E_i}$. We interpret knowledge-based formulas in the standard way, and say that agent i knows fact φ at trace τ precisely when the fact φ holds at all traces τ' agent i cannot distinguish from τ:

- $(\mathcal{I}, \tau, \nu) \models K_i\varphi$ if and only if $(\mathcal{I}, \tau', \nu) \models \varphi$ for all traces τ' in Tr such that $\tau \sim_i \tau'$.

As it is standard, we say that "everybody in group X knows φ" (written as $E_X\varphi$) if every agent i in X knows φ, and say that "φ is distributed knowledge among agents in group X" if φ is known by an agent that knows what each agent in the group X knows:

- $(\mathcal{I}, \tau, \nu) \models E_X\varphi$ iff $(\mathcal{I}, \tau, \nu) \models K_i\varphi$ for all $i \in X$, and
- $(\mathcal{I}, \tau, \nu) \models D_X\varphi$ iff $(\mathcal{I}, \tau', \nu) \models \varphi$ for all traces τ' such that $\tau' \sim_i \tau$ for all agents $i \in X$.

4.3 Formalizing basic information-flow properties

As mentioned before, information-flow properties are typically studied in settings where there are two agents, a low-level user L and a high-level user H, all confidential and neutral events are visible only to H, and all events visible to L are also visible to H. Recall that we model this by taking H's view of the system as $\mathcal{V}_H = (C, V, N)$, and L's view of the system as $\mathcal{V}_L = (\emptyset, V, \emptyset)$. In the following, let $\mathcal{V}^s = \langle \mathcal{V}_h, \mathcal{V}_L \rangle$.

4.3.1 Noninterference and simple variants of it

The well-known notion of *noninterference* [75] can be formalized as a richness condition on the set of executions of a system. It says that, for any trace in the system, there exists a trace in the system with the same sequence of visible events, except that no confidential events have occurred. We can write the condition that the set Tr of traces satisfies noninterference as follows:

$$NI(Tr) \equiv \forall \tau \in Tr.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (\tau'|_C = \langle\, \rangle).$$

It is intuitively clear that noninterference ensures that the low-level user never knows whether some confidential events have occurred, since it is always consistent with the low-level view of an execution that no confidential event has occurred. Following the approach of Fagin et al. [25], this intuition can be made precise:

$$\begin{array}{ll}
I_1(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \forall x.\ conf_H(x) \Rightarrow \neg K_L(occ(x)); \\
 & \forall \tau \in Tr.\, \forall c \in C.\, \exists \tau' \in Tr.\, \forall \beta \in E^*.\, (\tau'|_V = \tau|_V) \wedge (\tau' \neq \beta \cdot \langle c \rangle) \\[2ex]
I_2(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \forall x.\ conf_H(x) \Rightarrow \neg K_L(\diamondminus\, occ(x)); \\
 & \forall \tau \in Tr.\, \forall c \in C.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (c \notin \tau'|_C) \\[2ex]
I_3(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \neg K_L(\exists x.\ conf_H(x) \wedge occ(x)); \\
 & \forall \tau \in Tr.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (\forall \beta' \in E^*.\, \forall c \in C.\, \tau' \neq \beta \cdot \langle c \rangle) \\[2ex]
I_4(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \forall x.\ conf_H(x) \Rightarrow \neg K_L(\neg occ(x)); \\
 & \forall \tau \in Tr.\, \forall c \in C.\, \exists \tau' \in Tr.\, \exists \beta \in E^*.\, (\tau'|_V = \tau|_V) \wedge (\tau' = \beta \cdot \langle c \rangle) \\[2ex]
I_5(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \forall x.\ conf_H(x) \Rightarrow \neg K_L(\boxminus \neg occ(x)); \\
 & \forall \tau \in Tr.\, \forall c \in C.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (c \in \tau'|_C) \\[2ex]
I_6(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \neg K_L(\forall x.\ conf_H(x) \Rightarrow \neg occ(x)); \\
 & \forall \tau \in Tr.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (\exists \beta \in E^*.\, \exists c \in C.\, \tau' = \beta \cdot \langle c \rangle) \\[2ex]
I_7(Tr): & \mathcal{I}_{\mathcal{V}}(Tr) \models \neg K_L(\forall x.\ conf_H(x) \Rightarrow \boxminus \neg occ(x)); \\
 & \forall \tau \in Tr.\, \exists \tau' \in Tr.\, (\tau'|_V = \tau|_V) \wedge (\tau'|_C \neq \langle\, \rangle).
\end{array}$$

Figure 4.1: Simple information-flow restrictions.

Lemma 4.3.1: *Let Tr be a set of traces in a system for a high-level user H with view $\mathcal{V}_H = (C, V, N)$ and a low-level user L with view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{I} = (E, Tr, \mathcal{V}^s, \pi)$. Then $NI(Tr)$ is satisfied if and only if $\mathcal{I} \models \neg K_L(\exists x.conf_H(x) \wedge \diamondminus\, occ(x))$.*

Proof: Let τ be an arbitrary trace in Tr, and ν an arbitrary valuation. It suffices to note that $(\mathcal{I}, \tau, \nu) \models \neg K_L(\exists x.conf_H(x) \wedge \diamondminus\, occ(x))$ iff there exists a trace τ' in Tr such that $\tau' \sim_L \tau$ (i.e., $\tau'|_V = \tau|_V$) and $(\mathcal{I}, \tau', \nu) \models \forall x.\ conf_H(x) \Rightarrow \neg occ(x)$ (i.e., $\tau'|_C = \langle\, \rangle$). ∎

In Figure 4.3.1, we give a number of simple knowledge-based formulations of information-flow restrictions, together with their equivalent representation in the event-system formalism. $I_1(Tr)$ (resp. $I_2(Tr)$) says that, for each confidential event, the adversary considers it possible that the event has not occurred (resp., has never occurred). $I_3(Tr)$ says that the adversary considers it possible that no confidential event has just occurred. $I_4(Tr)$ (resp., $I_5(Tr)$) says that, for any confidential event, the adversary considers it possible that the event has just occurred (resp., occurred at some point in the past), while $I_6(Tr)$ (resp., $I_7(Tr)$) says that the adversary considers it possible that some confidential event has just occurred (resp., some confidential event occurred at some point in the past). While both the event-based and the knowledge-based formulations of these properties are simple, we believe that the knowledge-based formulation makes it easier for the reader to assign a meaning to these properties and manipulate them in more complex arguments.

4.3.2 Generalized noninterference and backwards insertion of events

As has been noted before, and is clear from the knowledge-based reformulation above, noninterference alone does not prohibit a low-level user from knowing that no confidential event has occurred. Arguably, knowing that no confidential event has occurred is a form of information flow. For example, in a protocol where high-level users perform a confidential action only if some secret bit is 0, a low-level user that knows that no confidential event has occurred actually knows that the secret bit is 1. A well-known property that strengthens noninterference to restrict this type of information flow is *generalized noninterference* (GNI) [66]. Informally, GNI says that "any interleaving of the high-level input (i.e., confidential) events of one trace with the low-level (i.e., visible) events of another trace can be made a possible trace by adapting the outputs". As shown by Mantel [62], GNI is equivalent to the conjunction of NI and a property known as *backwards strict insertion of admissible events* ($BSIA$). According to Hutter [51], $BSIA$ "demands that observing the visible part v of a system run does not imply that (confidential event) c has not happened at some point". Formally, $BSIA$ is defined as follows:

$$\begin{array}{rcl} BSIA(Tr) & \equiv & \forall \alpha, \beta \in E^*.\, \forall c \in C. \\ & & \quad (\beta \cdot \alpha \in Tr \wedge \alpha|_C = \langle\, \rangle) \wedge \\ & & \quad (\exists \gamma \in E^*.\, (\gamma|_C = \beta|_C) \wedge (\gamma \cdot \langle c \rangle \in Tr)) \Rightarrow \\ & & \quad \exists \alpha' \in E^*.\, (\beta \cdot \langle c \rangle \cdot \alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\, \rangle) \\ GNI(Tr) & \equiv & NI(Tr) \wedge BSIA(Tr). \end{array}$$

Essentially, $BSIA(Tr)$ says that we can insert a confidential event at any point in a trace in Tr, provided that (1) the confidential event is *admissible*, that is, a user that has access only to confidential events in a trace thinks it possible that the event occurred at the point where it is inserted, and (2) we insert confidential events only at points of the trace after which no confidential events occur.

The formal definition of $BSIA$ does not completely capture the intuition that $BSIA(Tr)$ precludes low-level users from knowing that no confidential event has occurred. Consider a system in which, for no confidential event c, is it the case that the low-level user knows that c has not occurred, that is, a system in which the formula $\forall x.\, \mathit{conf}_H(x) \Rightarrow \neg K_L(\boxminus \neg occ(x))$ is valid. We can show that such a system satisfies a much simpler property than $BSIA$, which we call $BSIA^s$:

$$\begin{array}{rcl} BSIA^s(Tr) & \equiv & \forall c \in C.\, \forall \tau \in E^*.\, (\tau \in Tr) \Rightarrow \\ & & \quad \exists \tau' \in Tr.\, \exists \beta', \alpha' \in E^*.\, (\tau' = \beta' \cdot \langle c \rangle \cdot \alpha') \wedge (\tau'|_V = \tau|_V). \end{array}$$

Lemma 4.3.2: *Let Tr be a set of traces in a system for high-level user H with view $\mathcal{V}_H = (C, V, N)$ and low-level user L with view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{I} = (E, Tr, \mathcal{V}^s, \pi)$.*

Then

$$\mathcal{I} \models \forall x.\ \mathit{conf}_H(x) \Rightarrow \neg K_L(\boxminus \neg \mathit{occ}(x))$$

if and only if $BSIA^s(Tr)$ is satisfied.

Proof: Let τ be an arbitrary trace in Tr and ν be an arbitrary valuation. The proof follows from the observation that $(\mathcal{I}, \tau, \nu) \models \forall x.\ \mathit{conf}_H(x) \Rightarrow \neg K_L(\boxminus \neg \mathit{occ}(x))$ iff, for all $c \in C$, there exists a trace τ' such that $\tau' \sim_L \tau$ (that is, $\tau'|_V = \tau|_V$), and $(\mathcal{I}, \tau', \nu[x/c]) \models \diamond\, \mathit{occ}(x)$ (that is, $c \in \tau'|_C$). ∎

It is not obvious how $BSIA^s(Tr)$ and $BSIA$ are related. One important difference between them is that $BSIA$ requires α and α' to coincide on $V \cup C$, while $BSIA^s(Tr)$ requires τ and τ' to coincide on V. Qualitatively, we can interpret this difference as saying that, while $BSIA^s$ assumes that the low-level user has access only to the visible events in a trace, $BSIA$ talks about precluding information flow from a low-level user that may have access to confidential events as well. In other words, $BSIA$ refers to a low-level user that, even when given access to *all* confidential events that have occurred in an execution, still considers it possible that one more confidential event has occurred.

4.4 More complex MAKS-style information-flow properties

4.4.1 Multiple adversaries

The distinction between $BSIA_{\mathcal{V}}$ and $BSIA^s_{\mathcal{V}}$ suggests that it is useful to distinguish between a low-level user that has access only to low-level events, and a low-level user that may have access to confidential events. We call the latter an *adversary*.

Following Halpern et al. [45, 49], we model the adversary as another agent in the system. For our analysis, we allow systems with more than one adversary, and focus in particular on the following adversaries, all passive observers of the system. For all sets X, Y of agents in $\mathcal{A}$,

- X's eavesdropper adv_X is the adversary that can intercept all (and only) visible events of agents in the group X;
- X's communication eavesdropper adv_X^{Comm} is the adversary that can intercept all (and only) communication events involving agents in the group X;
- (X, Y)'s communication eavesdropper $adv_{X,Y}^{Comm}$ is the adversary that can intercept all (and only) communication events between agents in the group X and agents in the group Y.

For ease of exposition, all agents in $\mathcal{A}$ are taken to be honest, and we consider only passive adversaries, with capabilities as defined above. In the following, $\mathcal{A}^{adv}$ represents the set of all adversaries adv_X, adv_X^{Comm}, and $adv_{X,Y}^{Comm}$, for all sets X, Y in $\mathcal{A}$.

The notion of two traces being indistinguishable to an agent extends naturally to adversaries: Let τ and τ' be two arbitrary traces in Tr, and let X and Y be sets of agents. We define

- $\tau \sim_{adv_X} \tau'$ if and only if $\tau|_{V_X} = \tau'|_{V_X}$, where $V_X = \bigcup_{i \in X} V_i$;
- $\tau \sim_{adv_X^{Comm}} \tau'$ if and only if $\tau|_{Comm_X} = \tau'|_{Comm_X}$, where $Comm_X = \bigcup_{i \in X} Comm_i$;
- $\tau \sim_{adv_{X,Y}^{Comm}} \tau'$ if and only if $\tau|_{Comm_{X,Y}} = \tau'|_{Comm_{X,Y}}$, where $Comm_{X,Y} = \bigcup_{i \in X, j \in Y} Comm_{i,j}$.

We extend our logic so that it can express an adversary's knowledge of the system by associating with each adversary adv the knowledge operator K_{adv}. As before, $K_{adv}\varphi$ says that "adversary adv knows φ", and $(\mathcal{I}, \tau, \nu) \models K_{adv}\varphi$ holds if and only if $(\mathcal{I}, \tau', \nu) \models \varphi$ holds for all traces τ' in Tr such that $\tau' \sim_{adv} \tau$.

A number of connections can be established between what observers of a group of agents know and what observers of each agent in the group know, individually or by sharing their knowledge with the other observers. While a thorough analysis is beyond the scope of this chapter, we present one such result, which shows that the passive observers of a group X of agents know precisely what is distributed knowledge among the group of individual passive observers of each agent in X, provided that no two distinct agents may share the same event:

Lemma 4.4.1: *Suppose that $E_i \cap E_j = \emptyset$ for all distinct agents i, j in $\mathcal{A}$. Then, for all sets of traces Tr, views $\mathcal{V}$, and formulas φ, the following formulas are all valid with respect to $\mathcal{I}$:*

- $K_{adv_X}\varphi \Leftrightarrow D_{\{adv_i \mid i \in X\}}\varphi$;
- $K_{adv_X^{Comm}}\varphi \Leftrightarrow D_{\{adv_i^{Comm} \mid i \in X\}}\varphi$;
- $K_{adv_{X,Y}^{Comm}}\varphi \Leftrightarrow D_{\{adv_{i,j}^{Comm} \mid i \in X, j \in Y\}}\varphi$.

Proof: For the first equivalence, it suffices to note that, by definition, $\tau \sim_X \tau'$ iff $\tau|_{\cup_{i\in X} V_i} = \tau'|_{\cup_{i\in X} V_i}$, which, as $E_i \cap E_{i'} = \emptyset$ for distinct agents i and i' in X, is equivalent to $\tau|_{E_i} = \tau'|_{E_i}$ for all i in X. For the second equivalence, we remark that $\tau \sim_{adv_X^{Comm}} \tau'$ iff $\tau|_{\cup_{i,j\in\mathcal{A}} Comm_{i,j}} = \tau'|_{\cup_{i,j\in\mathcal{A}} Comm_{i,j}}$, which, as $Comm_{i,j} \subseteq E_i$ and $E_i \cap E_{i'} = \emptyset$, is equivalent to $\tau|_{\cup_{j\in\mathcal{A}} Comm_{i,j}} = \tau'|_{\cup_{j\in\mathcal{A}} Comm_{i,j}}$ for all $i \in \mathcal{A}$, and also equivalent to $\tau|_{Comm_i} = \tau'|_{Comm_i}$ for all $i \in \mathcal{A}$. The proof for the last equivalence proceeds similarly, once we note that, since $Comm_{i,j} \cap Comm_{i,k} = \emptyset$ for all distinct j and k, and $Comm_{i,j} \cap Comm_{i',k} = \emptyset$ for all distinct i and i' in X, it follows that $\tau|_{\cup_{i\in X, j\in Y} Comm_{i,j}} = \tau'|_{\cup_{i\in X, j\in Y} Comm_{i,j}}$ is equivalent to $\tau|_{Comm_{i,j}} = \tau'_{Comm_{i,j}}$ for all i in X and j in Y. ∎

4.4.2 Reveal events and counterfactuals

When analyzing information flows in systems with adversaries as above, it makes sense to consider a number of counterfactual situations. Preserving the secrecy of certain facts even in such hypothetical situations provides strong security guarantees. We consider the following situations:

- All confidential events of group Y are revealed to group X's eavesdropper. We denote the event of revealing all confidential events of Y to adv_X as $rev_{X,Y}$; for simplicity, we also define the 0-ary predicate that stands for "event $rev_{X,Y}$ occurs" as $rev_{X,Y}$. It will be clear from context whether we refer to the reveal event or the corresponding formula. If $Y = X$, we simply write rev_X.

- All events of group Y (not just the confidential ones) are revealed to adv_X. We denote both the occurrence of such a reveal event and the corresponding predicate as $rev^{all}_{X,Y}$ (resp., rev^{all}_X if $Y = X$).

- All confidential events of group Y are revealed to adv^{Comm}_X, respectively, all events of group Y are revealed to adv^{Comm}_X. We denote these events (resp., predicates) as $rev^{Comm}_{X,Y}$ and $rev^{all,Comm}_{X,Y}$ (resp., rev^{Comm}_X and $rev^{all,Comm}_X$ if $Y = X$).

- All confidential (resp., all) events of group Z are revealed to $adv^{Comm}_{X,Y}$. We denote these events as $rev^{Comm}_{X,Y,Z}$, and $rev^{all,Comm}_{X,Y,Z}$, respectively.

We remark that, intuitively, the occurrence of rev_X (resp. rev^{all}_X) is not equivalent to the simultaneous occurrence of events rev_i (resp., rev^{all}_i) for all agents i in X. This is because rev_i (resp., rev^{all}_i) reveals i's confidential events (resp., all i's events) to the observer of agent i, while rev_X (resp., rev^{all}_X) reveals i's confidential events (resp., all i's events) to the observer of the *entire* group X.

In the following, let E^{rev} be the set of all events in E, together with the reveal events above. The set Tr^{rev} of *extended* traces corresponding to the set Tr is the set of all sequences τ' of events in E^{rev} such that $\tau|_E$ is a trace in Tr. For ease of exposition, we consider only extended traces with at most one reveal event. (It follows from the semantics of counterfactual implications, as defined later in the section, that we do not lose any generality by making this assumption.)

The indistinguishability relations for each adversary can be naturally extended to the set of extended traces in the obvious way. We give one example here. Consider two arbitrary extended traces τ and τ'. We take $\tau \sim_{adv_i} \tau'$ to hold if and only if one of the following is true: (1) neither τ nor τ' contains a confidential event and $\tau|_{V_i} = \tau|_{V_i}$, (2) adv_i learned i's confidential events at some point in the past, that is, $\tau = \beta \cdot \langle rev_i \rangle \cdot \alpha$, $\tau' = \beta' \cdot \langle rev_i \rangle \cdot \alpha'$, $\beta'|_{V_i \cup C_i} = \beta|_{V_i \cup C_i}$, and $\alpha'|_{V_i} = \alpha|_{V_i}$, (3) adv_i learned all of i's events at some time in the past, that is, $\tau = \beta \cdot \langle rev_i^{all} \rangle \cdot \alpha$, $\tau' = \beta' \cdot \langle rev_i^{all} \rangle \cdot \alpha'$, $\beta'|_{E_i} = \beta|_{E_i}$, and $\alpha'|_{V_i} = \alpha|_{V_i}$, (4) adv_i learned the confidential events of a group Y at some point in the past, that is, $\tau = \beta \cdot \langle rev_{i,Y} \rangle \cdot \alpha$, $\tau' = \beta' \cdot \langle rev_{i,Y} \rangle \cdot \alpha'$, $\beta'|_{V_i \cup C_Y} = \beta|_{V_i \cup C_Y}$, and $\alpha'|_{V_i} = \alpha|_{V_i}$, or (5) adv_i learned all events of group Y at some point in the past, that is, $\tau = \beta \cdot \langle rev_{i,Y}^{all} \rangle \cdot \alpha$, $\tau' = \beta' \cdot \langle rev_{i,Y}^{all} \rangle \cdot \alpha'$, $\beta'|_{V_i \cup E_Y} = \beta|_{V_i \cup E_Y}$, and $\alpha'|_{V_i} = \alpha|_{V_i}$.

We would like to express in our logic strong secrecy requirements that involve adversaries not gaining knowledge of certain facts, even when they have obtained confidential information, including information regarding all the events associated with some group of agents. For example, if φ is a fact to be kept secret, we want to express the condition that φ remains secret even if agent i's observer/adversary obtained all of i's confidential events at some time in the past. However, we cannot write this as $(\diamondminus\, rev_i) \Rightarrow \varphi$, since typically that would hold trivially, as we would not expect rev_i to hold at any point in a realistic security system.

As explained by Halpern and Moses [43], we need to replace the implication $\Rightarrow$ with a *counterfactual* implication, denoted $\succ$. Counterfactual implications $\varphi \succ \psi$ are interpreted as being true at a trace τ in Tr if ψ holds at all traces τ' just like τ, except that φ holds at τ'. As φ may not hold at any trace in Tr, traces τ' range over the set E^{rev*} of all possible sequences of events in E^{rev}. To make this precise, to each trace τ in Tr we associate a relation $<_\tau$ on the E^{rev*}, with $\tau' <_\tau \tau''$ standing for "trace τ is closer to τ' than τ''". To match this intuition, $<_\tau$ is taken to be a partial order on E^{rev*}, with τ its unique minimal point (since, intuitively, the closest sequence of events to a given trace is the trace itself). Let $\mathcal{J}$ be an *extended* interpreted system, that is, an interpreted system $\mathcal{I} = (E^{rev*}, Tr^{rev}, \mathcal{V}, \pi)$ equipped with relations $<_\tau$, one for each trace τ in Tr. The set of extended traces just like τ except that φ holds can now be formally defined as

$$\begin{aligned}&\texttt{closest}([\![\varphi]\!]_{\mathcal{J}}, \tau, \nu) = \\ &\quad \{\tau' \in Tr^{rev} \mid (\mathcal{J}, \tau', \nu) \models \varphi \texttt{ and, if } (\mathcal{J}, \tau'', \nu) \models \varphi, \texttt{ then } \tau' \leq_\tau \tau''\}.\end{aligned}$$

We define $(\mathcal{J}, \tau, \nu) \models \varphi \succ \psi$ iff $(\mathcal{J}, \tau', \nu) \models \psi$ for all $\tau' \in \texttt{closest}([\![\varphi]\!]_{\mathcal{J}}, \tau, \nu)$.

In this chapter, we consider only relations $<_\tau$ such that, for all τ, τ', τ'', if $\tau = \beta \cdot \alpha$, $\tau' = \beta \cdot \langle e \rangle \cdot \alpha$, $\tau'' = \beta'' \cdot \langle e \rangle \cdot \alpha''$, and e does not occur in either α, β, α'' or β'', then $\tau' \leq_\tau \tau''$ holds. This essentially says that the closest trace to τ, among traces in which a given event e occurs exactly once, is a trace τ' such that the same events have occurred in τ' and τ, except that event e occurs in τ'. In particular, in systems $\mathcal{J}$ where the relations $<_\tau$ for traces τ in Tr satisfy the restriction above, the following holds for all reveal events rev and valuations ν:

$$\begin{aligned}&\texttt{closest}([\![\diamondminus\, rev]\!]_{\mathcal{J}}, \tau, \nu) = \\ &\{\tau' \in Tr^{rev} \mid \texttt{there exist } \beta, \alpha \in E^* \texttt{ such that } \tau' = \beta \cdot \langle rev \rangle \cdot \alpha \texttt{ and } \tau = \beta \cdot \alpha\}.\end{aligned} \tag{4.1}$$

Thus, $\texttt{closest}([\![\diamondminus\, rev]\!]_{\mathcal{J}}, \tau, \nu) \subseteq Tr^{rev}$ holds for all traces τ in Tr and valuations ν.

4.4.3 Capturing backwards insertion and deletion of events

We can now express more involved information-flow properties in the MAKS framework as constraints on adversaries' knowledge.

We start our analysis by defining the following formula:

$$\begin{aligned}\varphi_{BI} \equiv (\diamondminus\, rev_H^{all}) \succ\ & ((\forall x.\ conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all})) \Rightarrow \\ & (\forall x.\ conf_H(x) \Rightarrow \neg K_{adv_H} \neg(occ(x)\ \mathcal{A}\ rev_H^{all}))).\end{aligned}$$

φ_{BI} says that, if, at some point in the past, the adversary observed all events and no confidential events have occurred since, then, for all confidential events c, the adversary would still think it possible that c occurred immediately after all events were revealed to him. We can show that φ_{BI} is valid with respect to a set of traces Tr iff the following condition on Tr is satisfied:

$$\begin{aligned}BI(Tr) \equiv\ & \forall \alpha, \beta \in E^*. \forall C \in C. (\beta \cdot \alpha \in Tr \wedge \alpha|_C = \langle\,\rangle) \Rightarrow \\ & \exists \alpha' \in E^*. (\beta \cdot \langle c \rangle \cdot \alpha' \in Tr \wedge \alpha'|_V = \alpha|_V).\end{aligned}$$

Lemma 4.4.2: *Let Tr be a set of traces in a system where the high-level user H has view $\mathcal{V}_H = (C, V, N)$ and the low-level user L has view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi, \{<_\tau \mid \tau \in Tr\})$. Then $BI(Tr)$ holds if and only if $\mathcal{J} \models \varphi_{BI}$.*

Proof: By definition, $(\mathcal{J}, \tau, \nu) \models \varphi_{BI}$ if and only if, for all $\tau' \in \texttt{closest}($ $[\![\diamondminus\, rev_H^{all}]\!]_{\mathcal{J}}, \tau, \nu)$, if

$$(\mathcal{J}, \tau', \nu) \models \forall x.\ conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all}), \tag{4.2}$$

then

$$(\mathcal{J}, \tau', \nu) \models \forall x.\ conf_H(x) \Rightarrow P_{adv_H}(occ(x)\ \mathcal{A}\ rev_H^{all}). \tag{4.3}$$

By (4.1), we can restrict to extended traces τ' for which there exist sequences of events β and α in E^* with $\tau' = \beta \cdot \langle rev_H^{all} \rangle \cdot \alpha$ and $\tau = \beta \cdot \alpha$. (4.2) is satisfied for τ' if and only if $\alpha|_C = \langle\,\rangle$. (4.3) says that, for all confidential events c, there exists an extended trace τ'' such that $\tau'' \sim_{adv_H} \tau'$ and

$$(\mathcal{J}, \tau'', \nu\,[c/x]) \models occ(x)\ \mathcal{A}\ rev_H^{all}. \tag{4.4}$$

Since $\tau' = \beta \cdot \langle rev_H^{all}\rangle \cdot \alpha$, $\tau'' \sim_{adv_H} \tau'$ if and only if there exist sequences β'', α'' in E^* such that $\tau'' = \beta'' \cdot \langle rev_H^{all}\rangle \cdot \alpha''$, $\beta''|_E = \beta|_E$ (by the semantics of the rev_H^{all} event), and $\alpha''|_V = \alpha|_V$. This means that (4.4) holds if and only if $\beta'' = \beta$ and the first event in α'' is c, that is, there exists α' such that $\alpha'' = \langle c\rangle \cdot \alpha'$. We can now take $\beta' = \beta''$ and conclude that $BI_{\mathcal{V}}(Tr)$ is satisfied. The reverse implication follows immediately. ▮

Note that, unlike $BI(Tr)$, $BSIA$ requires that α' and α agree on the confidential events. To move one step closer to $BSIA$, we refine φ_{BI} to say that the adversary thinks it possible that c is the *last* confidential event, where we abbreviate $conf_H(x) \wedge ((\forall y.\ conf_H(y) \Rightarrow \neg occ(y))\ \mathcal{S}\ \ occ(x))$ as $lastC(x)$:

$$\begin{aligned}\varphi_{BSI} \ \equiv (\diamond\!\!\!\!\cdot\ rev_H^{all}) \succ\ \ &((\forall x.\ conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all})) \Rightarrow \\ &\qquad (\forall x.\ conf_H(x) \Rightarrow \neg K_{adv_H} \neg(lastC(x)\ \mathcal{A}\ rev_H^{all}))).\end{aligned}$$

Lemma 4.4.3: *Let Tr be a set of traces in a system where the high-level user H has view $\mathcal{V}_H = (C, V, N)$ and the low-level user L has view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi(\mathcal{V}^s), \{<_\tau \ \mid \tau \in Tr\})$. Then $\mathcal{J} \models \varphi_{BSI}$ if and only if the following property holds:*

$$\begin{aligned}BSI(Tr) \ \equiv\ \ &\forall \alpha, \beta \in E^*.\, \forall c \in C.\ (\beta \cdot \alpha \in Tr \wedge \alpha|_C = \langle\,\rangle) \Rightarrow \\ &\qquad \exists \alpha' \in E^*.\ (\beta \cdot \langle c\rangle \cdot \alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \alpha|_C).\end{aligned}$$

Proof: The proof essentially follows from the argument for Lemma 4.4.2. ▮

Note that BSI is just like $BSIA$, except that the admissibility restriction has been dropped; that is why it is also known as *backwards strict insertion of events* [62]. There is no difficulty in expressing the admissibility restriction: it simply says that, if H^s is a user who has access only to confidential events (call this a *restricted* high-level user, to distinguish it from a standard high-level user, who is assumed to have access to visible events as well), then c is admissible precisely when H^s thinks it possible that c occurs immediately after the reveal event has occurred. With this observation, the knowledge-based formulation of $BSIA$ becomes

$$\begin{aligned}
&\varphi_{BSIA} =_{\mathrm{def}} \\
&(\diamondsuit\, rev_H^{all}) \succ \quad ((\forall x.\ conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all})) \Rightarrow \\
&\qquad\qquad\qquad \forall x.\ (conf_H(x) \wedge (P_{H^s}(\bigcirc occ(x)))\ \mathcal{A}\ rev_H^{all}) \Rightarrow \\
&\qquad\qquad\qquad\quad \neg K_{adv_H} \neg (lastC(x)\ \mathcal{A}\ rev_H^{all})).
\end{aligned}$$

Theorem 4.4.4: *Let Tr be a set of traces in a system where the high-level user H has view $\mathcal{V}_H = (C, V, N)$ and the low-level user L has view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi(\mathcal{V}^s), \{<_\tau \mid \tau \in Tr\})$. Then $BSIA(Tr)$ holds if and only if $\mathcal{J} \models \varphi_{BSIA}$.*

Proof: The proof is similar similar to that of Lemma 4.4.2. ∎

The exercise of reformulating NI and $BSIA$ in terms of what users with different views of the system think possible or never learn has exposed most of the necessary elements for similar interpretations of other known information-flow properties. For example, for systems in which only high input events are considered confidential, the formulas above capture *backwards insertion of (HI-admissible) inputs*; if we consider as *admissible* a confidential event considered possible by a user that has access to all events, not just the confidential ones, then we can characterize the *insertion of admissible inputs* (see [63]). Furthermore, if we interpret a *reveal* event as revealing not just the confidential events, but also all the neutral ones, the formulas above characterize *back-*

wards insertion of (admissible) events; that is, the formulas above with all occurrences of rev_H^{all} replaced by rev_H capture backwards insertion of (admissible) events.

The properties above restrict the flow of information about non-occurrences of events. It is not difficult to refine NI to capture restrictions on the flow of information about occurrences of events. For example, *strict deletion of events* [62] requires that, for any trace τ in the system, if c is the last confidential event in τ, then there exists a trace τ' in the system that is just like τ, modulo neutral events, except that c does not occur in τ':

$$\begin{aligned} D(Tr) \;\equiv\; & \forall \alpha, \beta \in E^*.\, \forall c \in C. \\ & \quad (\beta \cdot \langle c \rangle \cdot \alpha \in Tr \wedge \alpha|_C = \langle\,\rangle) \Rightarrow \\ & \quad \exists \alpha' \in E^*.\, (\beta \cdot \alpha' \in Tr \wedge \alpha'|_V = \alpha|_V \wedge \alpha'|_C = \langle\,\rangle). \end{aligned}$$

We can show that $D(Tr)$ holds if and only if, even if all the confidential events in a trace but the last one are revealed to the adversary, the adversary would still think it possible that no confidential event has occurred after the last *reveal* event; that is, the following is valid with respect to Tr:

$$\begin{aligned} & \varphi_D =_{\text{def}} (\diamondminus\, rev_H^{all}) \succ \\ & \forall x.\, ((lastC(x)\ \mathcal{A}\ rev_H^{all}) \Rightarrow P_{adv_H}(\forall y.\ conf_H(y) \Rightarrow (\neg occ(y)\ \mathcal{S}\ rev_H^{all}))). \end{aligned}$$

Theorem 4.4.5: *Let Tr be a set of traces in a system where the high-level user H has view $\mathcal{V}_H = (C, V, N)$ and the low-level user L has view $\mathcal{V}_L = (\emptyset, V, \emptyset)$. Let $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi(\mathcal{V}^s), \{<_\tau \mid \tau \in Tr\})$. Then $D(Tr)$ holds if and only if $\mathcal{J} \models \varphi_D$.*

Proof: By the semantics of counterfactual implication and (4.1), $(\mathcal{J}, \tau, \nu) \models \varphi_D$ if and only if, for all extended traces τ' in Tr^{rev} such that there exist β, α in E^* with $\tau' = \beta \cdot \langle rev \rangle \cdot \alpha$ and $\tau = \beta \cdot \alpha$, and for confidential events c, if

$$(\mathcal{J}, \tau', \nu\,[c/x]) \models lastC(x)\ \mathcal{A}\ rev_H^{all} \tag{4.5}$$

then

$$(\mathcal{J}, \tau', \nu\,[c/x]) \models P_{adv_H}(\forall y.\ conf_H(y) \Rightarrow (\neg occ(y)\ \mathcal{S}\ rev_H^{all})). \tag{4.6}$$

(4.5) is satisfied if and only if there exists α_0 in E^* such that $\alpha = \langle c \rangle \cdot \alpha_0$ (i.e., $\tau = \beta \cdot \langle c \rangle \cdot \alpha_0$ and $\tau' = \beta \cdot \langle rev_H^{all} \rangle \cdot \langle c \rangle \cdot \alpha_0$) and $\alpha_0|_C = \langle\ \rangle$. (4.6) is satisfied if and only if there exists an extended trace τ'' in Tr^{rev} such that $\tau'' \sim_{adv_H} \tau'$ and

$$(\mathcal{J}, \tau'', \nu\,[c/x]) \models \forall y.\ conf_H(y) \Rightarrow (\neg occ(y)\ \mathcal{S}\ rev_H^{all}). \tag{4.7}$$

Since $\tau' = \beta \cdot \langle rev_H^{all} \rangle \cdot \langle c \rangle \cdot \alpha_0$, it follows that $\tau'' \sim_{adv_H} \tau'$ holds if and only if there exists $\alpha' \in E^*$ such that $\alpha'|_V = (\langle c \rangle \cdot \alpha_0)|_V$, so $\alpha'|_V = \alpha_0|_V$. The theorem now follows from the observation that (4.7) is equivalent to the condition $\alpha'|_C = \langle\ \rangle = \alpha_0|_C$. ∎

We remark that the formula above with all occurrences of rev_H^{all} replaced by rev_H characterizes *backwards deletion of confidential events* [63].

Simple logical reasoning proves that backwards strict deletion of confidential events is stronger than deletion of confidential events, which in turn implies NI. Similarly, we can show that backwards strict insertion of HI-admissible inputs is stronger than backwards insertion of admissible confidential events. The classification of such properties has been studied extensively, both in the event-system formulation of the Assembly Kit and in the context of process algebras [30, 31]. While our reformulations allow alternative derivations of the relations among such information-flow properties, rather than rederiving known results, we focus on finding intuitive explanations for information-flow properties. In the process, we identify simple variants of known properties that have natural knowledge-based formulations and express basic restrictions likely to be desired in realistic settings. For instance, as shown in Figure 2, a number of reasonable requirements similar to $BSIA$ and D can be defined: I_8 says that, even if all confidential events were revealed, for all confidential events c, the adversary would still consider it

$$
\begin{aligned}
I_8(Tr): \quad & (\diamondminus\, rev_H^{all}) \succ \; ((\forall x.\, conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all})) \Rightarrow \\
& \qquad \forall x.\, conf_H(x) \Rightarrow P_{adv_H}((\Diamond occ(x))\ \mathcal{A}\ rev_H^{all})) \\
I_9(Tr): \quad & (\diamondminus\, rev_H^{all}) \succ \; ((\forall x.\, conf_H(x) \Rightarrow (\neg occ(x)\ \mathcal{S}\ rev_H^{all})) \Rightarrow \\
& \qquad P_{adv_H}(\exists y.\, conf_H(y) \wedge (\Diamond occ(y)\ \mathcal{A}\ rev_H^{all}))) \\
I_{10}(Tr): \quad & (\diamondminus\, rev_H^{all}) \succ \; \forall x.\, conf_H(x) \Rightarrow \; (((lastC(x)\ \mathcal{A}\ rev_H^{all}) \Rightarrow \\
& \qquad \forall y.\, conf_H(y) \Rightarrow P_{adv_H}((\neg occ(y)\ \mathcal{S}\ rev_H^{all}))).
\end{aligned}
$$

Figure 4.2: Variants of $BSIA$ and D.

possible that c will occur at some time in the future; I_9 is a weaker property, as it requires that, in similar circumstances, the adversary considers it possible that some confidential event will occur, without precluding the adversary from knowing that some other confidential event will not occur at any time in the future; I_{10} says that, if all but the last confidential event were revealed to the adversary, for all confidential events c, the adversary would still consider it possible that c has not occurred since the rev^{all} event. I_{10} is a weaker variant of D, as it allows executions in which different confidential events are ruled out by the low-level user at different points in time after the rev^{all} event.

4.5 Extension to cryptographic settings

Recently, there has been increased interest in adapting information-flow properties to cryptographic settings. One approach, due to Hutter and Schairer [51] (HS from now on), specifically addresses possibilistic information-flow properties in the MAKS framework. In this section, we shed light on the HS results by translating them into our framework.

4.5.1 The Hutter and Schairer approach

We start by presenting the HS results. The main assumption is that, in cryptographic systems, it is possible that an agent cannot distinguish between two different messages encrypted with unknown keys. Taking $\{\!|x|\!\}_k$ to stand for message x encrypted with key k, we express the fact the x and x' encrypted with possible different keys are indistinguishable as $\{\!|x|\!\}_k \approx \{\!|x'|\!\}_{k'}$. The indistinguishability relation $\approx$ on messages initially applies only to encrypted messages. But we can extend it to all messages, under the assumption that messages are either keys, plaintexts, encrypted messages or concatenations of such messages: if msg is a key or plaintext, we take $msg \approx msg'$ if and only if $msg' = msg$; if $msg = msg_1 \cdot msg_2$ (i.e., msg is the concatenation of msg_1 and msg_2), then we take $msg \approx msg'$ if and only if there exist msg'_1, msg'_2 such that $msg_1 \approx msg'_1$, $msg_2 \approx msg'_2$, and $msg' = msg'_1 \cdot msg'_2$. This extension is not present in the HS paper, but it will be useful in clarifying our comparison with the HS approach later on in this chapter. We extend $\approx$ to receive events be defining $recv_i(msg) \approx recv_i(msg')$ if $msg \approx msg'$.

The executions τ' the low-level user cannot distinguish from τ are defined such that the visible events at the same position in τ and in τ' are related by $\approx$. We write $\tau \approx \tau'$ iff $\tau|_V = e_1 \cdots e_n$, $\tau'|_V = e'_1 \cdots e'_n$, and $e_i \approx e'_i$ for $i = 1, \ldots, n$.

HS define the extension of deletion of events and backwards strict insertion of confidential events in a cryptographic setting as follows:

$$
\begin{aligned}
D^{HS}_{\approx}(Tr) \;\equiv\; & \forall \alpha, \beta \in E^*.\, \forall c \in C. \\
& (\beta \cdot \langle c \rangle \cdot \alpha \in Tr \wedge \alpha|_C = \langle\, \rangle) \Rightarrow \\
& \exists \alpha' \in E^*.\, (\beta \cdot \alpha' \in Tr \wedge \alpha'|_V \approx \alpha|_V \wedge \alpha'|_C = \langle\, \rangle);
\end{aligned}
$$

$$
\begin{aligned}
BSIA^{HS}_{\approx}(Tr) \;\equiv\; & \forall \alpha, \beta \in E^*.\, \forall c \in C. \\
& (\beta \cdot \alpha \in Tr \wedge \alpha|_C = \langle\, \rangle) \wedge (\exists \gamma \in E^*.\, (\gamma|_C = \beta|_C) \wedge \\
& (\gamma \cdot \langle c \rangle \in Tr)) \Rightarrow \\
& \exists \alpha' \in E^*.\, (\beta \cdot \langle c \rangle \cdot \alpha' \in Tr \wedge \alpha'|_V \approx \alpha|_V \wedge \alpha'|_C = \langle\, \rangle).
\end{aligned}
$$

We do not believe that these definitions are always appropriate. Whether the event of agent i sending $\{\!|x|\!\}_k$ on a link l is indistinguishable from the event of agent i sending $\{\!|x'|\!\}_{k'}$ on link l should depend on what an agent knows. For example, from the point of view of a sender i the events are distinct, as i is the one choosing both the message and the key to encrypt with; from the point of view of one of i's observers, the two events are indistinguishable. Thus, rather than using a single indistinguishability relation $\approx$, we define an equivalence relation $\approx_i$ for each agent i in the system or observer. From the point of view of i's observer, the events of i sending two indistinguishable messages on the same link are indistinguishable; thus, we define $send_{i,j}(msg_1) \approx_{adv_i} send_{i,j}(msg_2)$ for all messages $msg_1 \approx msg_2$. To connect with the HS notion of indistinguishability, we take $recv_i(msg_1) \approx_{adv_i} recv_i(msg_2)$ if and only if $recv_i(msg_1) \approx recv_i(msg_2)$. We assume that the relations $\approx_i$ can be extended to all events that agent i can observe; how this is done depends on the specifics of the system we are analyzing. In particular, we assume that an adversary adv can always distinguish the events of being told confidential information from all other events; that is, $rev_{adv} \approx_{adv} e$ if and only if $e = rev_{adv}$ and $rev^{all}_{adv} \approx_{adv} e$ if and only if $e = rev^{all}_{adv}$.

In order to extend the indistinguishability relations $\approx_{adv}$ from events to traces, we must make clear what we assume about an adversary's capabilities. The first type of adversary we consider is quite weak: to decide whether it is possible for trace $\tau = e_1 \cdot \ldots \cdot e_n$ to be the current trace, the adversary looks at the sequence of events visible to him, looks at the sequence of events that would be visible to him if the trace were τ, and then checks that corresponding events look the same. Note that if a reveal event occurs in τ, the events that are intuitively visible to the adversary include the reveal event and all the events it reveals. To formalize this, we use $V^{\mathcal{A}}_{adv_a}(\tau)$ to denote the events visible to the adversary in trace τ and define

- $V^{\mathcal{V}}_{adv_a}(\tau) = \tau|_{V_a}$ if no reveal event occurs in τ,
- $V^{\mathcal{V}}_{adv_a}(\tau) = \alpha|_{V_a \cup C_a} \cdot rev_{adv_a} \cdot \beta|_{V_a}$ if $\tau = \alpha \cdot rev_{adv_a} \cdot \beta$, and
- $V^{\mathcal{V}}_{adv_a}(\tau) = \alpha \cdot rev^{all}_{adv_a} \cdot \beta|_{V_a}$ if $\tau = \alpha \cdot rev^{all}_{adv_a} \cdot \beta$.

Definition 4.5.1: Given an $\mathcal{A}$-view $\mathcal{V}$, two extended traces τ and τ' are indistinguishable to the first type of adversary, written $\tau \approx^{HS}_{adv_a} \tau'$, if and only if, if $V^{\mathcal{V}}_{adv_a}(\tau) = e_1 \cdot \ldots \cdot e_n$ and $V^{\mathcal{V}}_{adv_a}(\tau') = e'_1 \cdot \ldots \cdot e'_{n'}$, then $n = n'$ and $e_i \approx_{adv_a} e'_i$ for all $i = 1, \ldots, n$. ∎

It is straightforward to extend the semantic model to include such indistinguishability relations on traces. The new semantic model is a tuple $(\mathcal{J}, \{\approx_{adv}\}_{adv})$, where $\mathcal{J}$ is a model as defined in Section 4.4.2 and $\approx_{adv}$ is defined as above. We define $(\mathcal{J}, \{\approx_{adv}\}_{adv}, \tau, \nu) \models \varphi$ in the same way as $(\mathcal{J}, \tau, \nu) \models \varphi$, except that $(\mathcal{J}, \{\approx_{adv}\}_{adv}, \tau, \nu) \models K_{adv}\varphi$ if and only if $(\mathcal{J}, \{\approx_{adv}\}_a, \tau', \nu) \models \varphi$ for all traces τ' with $\tau \approx_{adv} \tau'$. Note that the semantics of knowledge formulas is unchanged, as it is still the case that adversary knows a fact φ if and only if φ holds in all traces that he cannot distinguish from the current trace; we have just changed the indistinguishability relation.

We can now prove that the HS extensions of deletion of events and backwards strict insertion of events implicitly assume the first type of adversary:

Proposition 4.5.2: *Let* $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi, \{<_\tau \mid \tau \in Tr\})$*. The following both hold:*

(a) $D^{HS}_{\approx}(Tr)$ *holds if and only if* $(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}) \models \varphi_D$*;*

(b) $BSIA^{HS}_{\approx}(Tr)$ *holds if and only if* $(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}) \models \varphi_{BSIA}$*.*

Proof: We prove part (a); the proof for (b) proceeds similarly. The argument closely follows the reasoning for Theorem 4.4.5. By the semantics of counterfactual implication and (4.1), $(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}, \tau, \nu) \models \varphi_D$ if and only if, for all extended traces τ' in Tr^{rev} such that there exist β, $\alpha \in E^*$ with $\tau' = \beta \cdot \langle rev \rangle \cdot \alpha$ and $\tau = \beta \cdot \alpha$, and for all confidential events c, if

$$(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}, \tau', \nu\,[c/x]) \models \mathit{lastC}(x)\ \mathcal{A}\ \mathit{rev}^{all}_H \tag{4.8}$$

then

$$(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}, \tau', \nu\,[c/x]) \models P_{adv_H}(\forall y.\ \mathit{conf}_H(y) \Rightarrow (\neg \mathit{occ}(y)\ \mathcal{S}\ \mathit{rev}^{all}_H)). \tag{4.9}$$

(4.8) is satisfied if and only if there exists $\alpha_0 \in E^*$ such that $\alpha = \langle c \rangle \cdot \alpha_0$, (i.e., $\tau = \beta \cdot \langle c \rangle \cdot \alpha_0$ and $\tau' = \beta \cdot \langle \mathit{rev}^{all}_H \rangle \cdot \langle c \rangle \cdot \alpha_0$), and $\alpha_0|_C = \langle\ \rangle$. (4.9) is satisfied if and only if there exists an extended trace τ'' in Tr^{rev} such that $\tau'' \approx^{HS}_{adv_H} \tau'$ and

$$(\mathcal{J}, \{\approx^{HS}_{adv}\}_{adv}, \tau'', \nu\,[c/x]) \models \forall y.\ \mathit{conf}_H(y) \Rightarrow (\neg \mathit{occ}(y)\ \mathcal{S}\ \mathit{rev}^{all}_H). \tag{4.10}$$

Since $\tau' = \beta \cdot \langle \mathit{rev}^{all}_H \rangle \cdot \langle c \rangle \cdot \alpha_0$, it follows that $\tau'' \approx^{HS}_{adv_H} \tau'$ holds if and only if there exists α' such that $\alpha'|_V \approx (c \cdot \alpha_0)|_V$, i.e., $\alpha'|_V \approx \alpha_0|_V$. The theorem follows from the observation that (4.10) is equivalent to the condition $\alpha'|_C = \langle\ \rangle = \alpha_0|_C$. ∎

Technically, note that the difference between $D_{\approx}^{HS}(Tr)$ and $D(Tr)$, as well as the difference between $BSIA_{\approx}^{HS}(Tr)$ and $BSIA(Tr)$, is in relaxing the constraint on the equality of the visible events in α and α' to equivalence according to the indistinguishability relation $\approx$. But what exactly does this mean? One possible interpretation is that the change from equality to equivalence is due to the change from the adversary's view of a trace $\beta \cdot \langle rev_{adv_H}^{all} \rangle \cdot \alpha$ being the same as his view of any trace $\beta \cdot \langle rev_{adv_H}^{all} \rangle \cdot \alpha'$ with $\alpha'|_V = \alpha|_V$ (in a system where cryptography is not taken into account) to being the same as his view at any trace $\beta \cdot \langle rev_{adv_H}^{all} \rangle \cdot \alpha'$ with $\alpha'|_V \approx \alpha|_V$ (in a cryptographic setting). However, it is not clear why the prefix of the trace should be β in the cryptographic case. Suppose, for example, that $D_{\approx}^{HS}(Tr)$ holds and $\langle send_{H,L}(\{\!|x|\!\}_k) \cdot c \rangle \cdot rev_{adv_H}^{all}$ is a trace in Tr^{rev}, where c is a confidential event. Let $\beta = send_{H,L}(\{\!|x|\!\}_k)$ and $\alpha = \langle\ \rangle$. $D_{\approx}^{HS}(Tr)$ requires that $send_{H,L}(\{\!|x|\!\}_k)$ be a trace in the system, and essentially says that, even if adv_H learnt all but the last (in our case, the only) confidential event, he would still consider it possible that no confidential event occurred and that the trace is $send_{H,L}(\{\!|x|\!\}_k)$. It is not clear why it is not enough to require that adv_H would consider it possible that no confidential event occurred, but other than that cannot rule out any of encrypted messages $\{\!|y|\!\}_k$ indistinguishable from $\{\!|x|\!\}_k$ as the message he intercepted on the communication link between H and L. Requiring β' and β to coincide not only on the confidential and neutral events, but also on the visible events, suggests that the implicit assumption is that the HS extensions of $BSIA(Tr)$ and $D(Tr)$ guard against leaking information to an adversary that, when told all the confidential events up to some point, is also able to "crack" the encryption scheme. An alternative explanation is that the HS definition guards against the adversary gaining knowledge about certain confidential facts even if he is told, not only all confidential events up to some point in the past, but also the content of the encrypted messages and the keys used to encrypt them on all communication links on which the adversary eavesdrops.

Now suppose that $D^{HS}_{\approx}(Tr)$ holds and $c \cdot send_{H,L}(k^{-1}) \cdot send_{H,L}(\{\!|x|\!\}_k)$ is a trace in Tr in which a confidential event c occurs, after which H sends two messages to L, a decryption key k^{-1} and a message x encrypted with key k. $D^{HS}_{\approx}(Tr)$ requires that there exist messages msg_1 and msg_2 such that $send_{H,L}(msg_1) \approx send_{H,L}(k^{-1})$ and $send_{H,L}(msg_2) \approx send_{H,L}(\{\!|x|\!\}_k)$. Since the decryption key k^{-1} is sent in the clear, it must be the case that $msg_1 = k^{-1}$. However, we would expect that the adversary, after intercepting a decryption key k^{-1}, could use it for decrypting subsequent intercepted message; in other words, we expect the adversary to decrypt $\{\!|x|\!\}_k$ and compute that x is the message H sent to L. The requirement above that some msg_2 exists such that $send_{H,L}(msg_2) \approx send_{H,L}(\{\!|x|\!\}_k)$ (as opposed to requiring msg_2 to be precisely $\{\!|x|\!\}_k$) seems too weak. More generally, the definition of two sequences $e_1 \ldots e_n$ and $e'_1 \ldots e'_n$ of visible events as equivalent from adv_H's point of view when the corresponding events are $\approx$-equivalent implicitly assumes a very weak adversary, either forgetful (not able to keep track of any message intercepted in the past) or not able to perform any cryptographic operations on intercepted messages. Arguably, this is not a realistic adversary.

4.5.2 A more realistic adversary

To model more realistic adversaries, we have to consider scenarios in which agents keep track of the messages they have intercepted and of the set of keys they have obtained from these messages. We define a notion of event indistinguishability from the point of view of an agent with respect to a set of keys. We write $e \approx^{Keys}_{a} e'$ to denote the fact that events e and e' are indistinguishable from the point of view of agent a with respect to a set $Keys$ of keys. The details of the definition of $\approx^{Keys}_{a}$ depend on the system we are analyzing. However, if a is either an agent in the system or an adversary, we require

$e \approx_a^{Keys} e'$ to hold if $e = recv_a(msg)$, $e' = recv_a(msg')$, and messages msg and msg' are indistinguishable to an agent who has obtained the set $Keys$, written $msg \approx^{Keys} msg'$. We define $msg \approx^{Keys} msg'$ as follows:

(1) if there exists a key $k^{-1} \in K$ such that $msg = \{\!|x|\!\}_k$, then $msg \approx^{Keys} msg'$ if and only if $msg' = \{\!|x'|\!\}_k$ for some x' such that $x \approx^{Keys} x'$;

(2) if there exist messages msg_1, msg_2 such that $msg = msg_1 \cdot msg_2$, then $msg \approx^{Keys} msg'$ if and only there exist messages msg'_1 and msg'_2 such that $msg' = msg'_1 \cdot msg'_2$, $msg_1 \approx^{Keys} msg'_1$, and $msg_2 \approx^{Keys} msg'_2$;

(3) if msg is neither a message encrypted with a key in $Keys$ nor a concatenation of messages, then $msg \approx^{Keys} msg'$ if and only if $msg \approx msg'$.

In addition, we require that $send_{a,b}(msg) \approx_{adv_a}^{Keys} send_{a,b}(msg')$ if and only if $msg \approx^{Keys} msg'$, as a's adversary cannot distinguish a's sending two undistinguishable messages.

Generalizing the approach of Halpern and Pucella [49], we would like to talk about the set of keys agent a obtains in a trace τ, denoted as $keysof_a(\tau)$. Intuitively, $keysof_a(\tau)$ is the result of agent a applying all possible decompositions, decryptions and encryptions to the messages received, starting with all the keys observed in the clear. We start by defining the set $keysof_a(e, Keys)$ of keys agent a can compute from an event e using a set $Keys$ of keys *in one pass*. (Note that a can be an adversary, as defined in Section 4.4.1, and that e can be an event that agent a observes or is revealed to a.) We proceed by induction on the structure of e:

- $keysof_a(recv_a(k), Keys) = \{k\} \cup Keys$ (for $k \in Keys$)
- $keysof_a(recv_a(\{\!|msg'|\!\}_{k^{-1}}), Keys) = keysof_a(recv_a(msg'), Keys) \cup Keys$ if $k \in Keys$

- $\mathit{keysof}_a(\mathit{recv}_a(\mathit{msg}\cdot\mathit{msg}'),\mathit{Keys}) = \mathit{keysof}_a(\mathit{recv}_a(\mathit{msg}),\mathit{Keys}) \cup \mathit{keysof}_a(\mathit{recv}_a(\mathit{msg}'),\mathit{Keys})$
- $\mathit{keysof}_a(\mathit{rev}_{\mathit{adv}_a},\mathit{Keys}) = \mathit{keysof}_a(\mathit{rev}_{\mathit{adv}_a^{full}},\mathit{Keys}) = \mathit{Keys}.$

To compute $\mathit{keysof}_a(\tau)$, the set of keys a can compute from trace τ, in general we need more than one pass. Consider for example the case when $\tau = \mathit{recv}_a(k^{-1})\cdot\mathit{recv}_a(\{\!|\mathit{msg}|\!\}_k)$, where k is a key. In one pass over τ, a can observe key k^{-1} from $\mathit{recv}_a(k^{-1})$ and no key from $\mathit{recv}_a(\{\!|\mathit{msg}|\!\}_k)$. However, it is clear that, once he knows k^{-1}, a can observe msg. Furthermore, msg may also contain keys that a can observe. The definition of $\mathit{keysof}(\tau)$ deals with this problem by iterating.

Definition 4.5.3: For all traces $\tau \in Tr^{rev}$ and agents a (including adversaries), define $\mathit{keysof}_a(\tau)$ to be the result Keys of the following computation:

```
$\mathit{Keys}' := \{\,\}$
$\mathit{repeat}$
    $\mathit{Keys} := \mathit{Keys}'$
    $\mathit{Keys}' := \cup_{\{e\ \mathit{event\ in}\ \tau\}}\, \mathit{keysof}_a(e,\mathit{Keys})$
$\mathit{until}\ \mathit{Keys} = \mathit{Keys}'.$
```

▮

Note that, without loss of generality, we have assumed that no keys are initially known by any agent. It is straightforward to model scenarios where it is possible for agents to have access to a set of keys before participating in any kind of protocol: we simply assume that the first event associated with a given agent in any trace in the system is a receive event for a message consisting of the concatenation of all such keys, in the clear.

With this notation, we can refine the indistinguishability relations $\approx_{adv}^{HS}$ on traces to take into account the set of keys an adversary can compute. If traces τ and τ' do not include reveal events, the keys are computed based only on the sequence of events visible to the adversary. If the adversary learns confidential information, the confidential or neutral events revealed also play a role in defining the set of keys the adversary can compute.

Definition 4.5.4: For any two traces $\tau, \tau' \in Tr^{rev}$, $\tau \approx_{adv}^{crypto} \tau'$ if and only if, if $V_{adv_a}^{\mathcal{V}}(\tau) = e_1 \cdot \ldots \cdot e_n$ and $V_{adv_a}^{\mathcal{V}}(\tau') = e'_1 \cdot \ldots \cdot e'_{n'}$, then $n = n'$, and for all $i = 1, \ldots, n$, if $Keys_i = keysof_{adv_a}(e_1 \cdot \ldots \cdot e_i)$ and $Keys'_i = keysof_{adv_a}(e'_1 \cdot \ldots \cdot e'_i)$, then $Keys_i = Keys'_i$ and $e_i \approx_{adv_a}^{Keys_j} e'_i$ for all $j = i, \ldots, n$. ∎

We give some examples to provide more intuition for this definition. Consider the simple case when trace τ contains no reveal events rev_{adv_a} or $rev_{adv_a}^{all}$. If only one event e_1 visible to adv_a occurs in τ, then, intuitively, adv_a cannot distinguish τ from any trace τ' such that only one event, say e'_1, visible to adv_a has occurred in τ', adv_a can compute the same set of keys $Keys_1$ from e_1 as from e'_1, and adv_a cannot distinguish e_1 and e'_1 given the keys $Keys_1$. If $\tau|_{V_a} = e_1 \cdots e_n$, then adv_a cannot distinguish τ from any trace τ' such that the same number of events are visible to adv_a in τ' as in τ (i.e., $\tau'|_{V_a}$ has the form $e'_1 \cdots e'_n$), and, at *any* point in τ', agent adv_a cannot distinguish τ' from τ. We need to be careful when making this intuition formal: Clearly, we want the set $Keys_i$ of keys adv_a has in τ after e_i occurs to be the same as the set $Keys'_i$ of keys adv_a has in τ' after e'_i occurs. Moreover, we want e_1 and e'_i to be indistinguishable by adv_a given this set of keys; that is, $e_i \approx_{adv_a}^{Keys_i} e'_i$. But we require more: e_i should remain indistinguishable from e'_i at all times after e_i occurs. Thus, we require that $e_i \approx_{adv_a}^{Keys_j} e'_i$ for all $j = i, \ldots, n$.

Example 4.5.5: Consider the traces $\tau_1 = send_{H,L}(k^{-1}) \cdot send_{H,L}(\{\!|x|\!\}_k)$ and $\tau_2 =$

$send_{H,L}(k^{-1}) \cdot send_{H,L}(\{\!|y|\!\}_k)$. These traces are indistinguishable by the HS adversary. However, $keysof_{adv_H}(\tau_1) = \{k^{-1}\}$ and $\{\!|x|\!\}_k \not\approx^{\{k^{-1}\}} \{\!|y|\!\}_k$. It follows that $\tau_1 \not\approx^{crypto}_{adv_H} \tau_2$; that is, adv_H can distinguish between τ_1 and τ_2. Similarly, traces $\tau_3 = send_{H,L}(\{\!|k|\!\}_{k'} \cdot \{\!|x|\!\}_{k^{-1}}) \cdot send_{H,L}(k'^{-1})$ and $\tau_4 = send_{H,L}(\{\!|k|\!\}_{k'} \cdot \{\!|y|\!\}_{k^{-1}}) \cdot send_{H,L}(k'^{-1})$ are indistinguishable by the HS adversary, but are distinguishable by the crypto adversary if $x \neq y$, since the adversary can apply the key k'^{-1} received in the second message to the first message received.

This example supports the claim that the relations $\approx^{crypto}_{adv}$ model a more realistic adversary than $\approx^{HS}_{adv}$.

Define

$$
\begin{aligned}
D^{HS}_{\approx^{crypto}}(Tr) \;\equiv\; & \forall \alpha, \beta \in E^*.\ \forall c \in C. \quad (\beta \cdot \langle c \rangle \cdot \alpha \in Tr \wedge \alpha|_C = \langle\,\rangle) \Rightarrow \\
& \quad \exists \beta', \alpha' \in E^*.\ (\beta' \cdot \alpha' \in Tr \wedge \beta' \approx^{crypto}_{adv_H} \beta \wedge \alpha'|_V \approx^{crypto}_{adv_H} \alpha|_V \wedge \alpha'|_C = \langle\,\rangle);
\end{aligned}
$$

$$
\begin{aligned}
& BSIA^{HS}_{\approx^{crypto}}(Tr) \;\equiv \\
& \quad \forall \alpha, \beta \in E^*.\ \forall c \in C.\ (\beta \cdot \alpha \in Tr \wedge \alpha|_C = \langle\,\rangle) \wedge \\
& \quad (\exists \gamma \in E^*.\ (\gamma|_C = \beta|_C) \wedge (\gamma \cdot \langle c \rangle \in Tr)) \Rightarrow \\
& \qquad \exists \beta', \alpha' \in E^*.\ (\beta' \cdot \langle c \rangle \cdot \alpha' \in Tr \wedge \beta' \approx^{crypto}_{adv_H} \beta \wedge \alpha'|_V \approx^{crypto}_{adv_H} \alpha|_V \wedge \alpha'|_C = \langle\,\rangle).
\end{aligned}
$$

We can show that the properties above represent the translation of backwards deletion and insertion of events information-flow properties with respect to our more realistic type of adversary:

Proposition 4.5.6: *Let* $\mathcal{J} = (E^{rev*}, Tr, \mathcal{V}^s, \pi, \{<_\tau \mid \tau \in Tr\})$.

(a) $(\mathcal{J}, \{\approx^{crypto}_{adv}\}_{adv}) \models \varphi_D$ *if and only if* $D_{\approx^{crypto}}(Tr)$ *holds;*

(b) $(\mathcal{J}, \{\approx^{crypto}_{adv}\}_{adv}) \models \varphi_{BSIA}$ *if and only if* $BSIA_{\approx^{crypto}}(Tr)$ *holds.*

Propostion 4.5.6 shows that assuming a more realistic adversary yields information-flow properties slightly different from the ones proposed by HS. One difference between $D_{\approx^{crypto}}$ and $BSIA_{\approx^{crypto}}$ and their HS counterparts comes from relaxing the requirement of exact equality between the sequences of events revealed to the adversary. A second difference comes from strengthening the constraints on the events visible to the adversary: once cryptography is taken into account, the events visible to H's observer are constrained to be equivalent with respect to the set of all keys H's observer is able to compute based on all the events intercepted in the past.

4.6 Remarks

The main contribution of this chapter is to provide a framework for characterizing some of the best-known MAKS-style information-flow requirements as knowledge-based formulas in a standard first-order logic of knowledge with counterfactual statements. This allows us to make contributions in a number of areas. First, knowledge-based characterizations can be used to identify subtle differences between the generally accepted explanations of information-flow requirements in the MAKS framework and their actual meaning. Second, a knowledge-based characterization allows for a more systematic approach for writing useful variants of MAKS properties. Lastly, we have shown that knowledge-based characterizations of information-flow properties can be naturally generalized to settings where cryptography is taken into account.

CHAPTER 5

CONCLUSION

This dissertation aimed at providing further evidence of the usefulness of knowledge-based programs in distributed settings. We focused on a number of aspects of knowledge-based programs: their applicability in solving the global function computation problem in a way that, in a precise sense, no unnecessary messages are sent; extending existing techniques for program synthesis from specifications in the Nuprl proof development system to knowledge-based specifications and programs; and formalizing information-flow properties as restrictions on what an adversary should never know about the system, allowing for a principled way of extending existing definitions to settings where cryptography and adversarial capabilities are taken into account.

We have identified a number of open problems and extensions to the analysis in this dissertation, which we now discuss.

5.1 Counterfactual belief-based programs

We believe that counterfactual knowledge-based programs are suited for writing programs that minimize the number of messages sent. Our use of counterfactual-based programs for solving global function computation in Chapter 2, as well as the results of Halpern and Moses [43] are evidence to this claim. Recently, Halpern and Moses [44] showed that different solution concepts (Nash equilibrium, correlated equilibrium etc.) in game theory can be understood as resulting from agents following the same knowledge-based program, but under different assumptions (such as whether players' strategies are independent or nor, or uncertainty is represented by a standard probability measure or not). The knowledge-based program says that each player chooses a certain strategy (move) if he believes that, if he were to choose any other strategy (move), the expected utility would not be higher. Technically, this program is similar in spirit to the program derived for solving the global function computation problem, as it relies on the notions of counterfactual implication and belief to express the fact that agents take "optimal" actions given their knowledge of the system. The difference is that, while in the context of global function computation problem, "optimality" is understood as sending only necessary messages (i.e., messages that, if not sent, their intended recipient would either never know the function value or the contents of the message), in the case of game theory, "optimality" is seen as choosing strategies of highest expected utility among all strategies players can choose from.

One area of possible interest comes from *routing*, that is, choosing paths in a network for sending data. Different approaches have been proposed for optimizing the path selection, in order to minimize either the number of hops or the cost associated with the data transmission along different links. In principle, one can define the goal of the problem so that eventually the intended recipient will know the data, and analyze to what extent different algorithms ensure that agents choose the next link to forward the data on in an optimal way given their knowledge of the network.

In a different vein, note that when considering global function computation, we made a number of simplifying assumptions. For example, we have assumed a fixed set of agents running the protocol, and that agents are reliable. This rules out systems in which agents may leave or join the network at different times. Moreover, the semantics of the knowledge and belief-based programs, as defined here and elsewhere in the literature, implicitly assumes a fixed number of agents in the system and that these agents run the protocol at all times. More work is needed in order to extend the framework to dynamic settings. Such an extension will not be trivial; for example, one subtlety comes from reasoning about agents that have participated in the protocol for some time, went off-line and later on rejoin the system.

5.2 Distributed program extraction from proofs

With respect to distributed program extraction from proofs about distributed systems, a number of questions, both theoretical and more applied, still remain open. While synthesis of distributed programs from epistemic and temporal specifications is undecidable in general, recent results [84] show that, under certain assumptions about the setting in which agents communicate, the problem is decidable. It would be worth understanding the extent to which these assumptions apply to our setting. To prove a result of this type, we need a better understanding of how properties of a number of knowledge-based programs relate to the properties of their composition; this would also allow us to prove stronger composition rules than the one presented in Section 3.2.2.

As we said, we believe that the approach that we sketched for extracting a standard program from the knowledge-based specification for the STP problem can be extended into a general methodology. As pointed out by Engelhart et al. [23], the key difficulty in extracting standard programs from abstract specifications is in coming up with good standard tests to replace the abstract tests in a program. However, it is likely that, by reducing the complexity of the problem and focusing only on certain classes of knowledge-based specifications, “good” standard tests can be more easily identified.

5.3 Knowledge-based analysis of information-flow

We have made the case that knowledge-based formulations of some of the building blocks of the Modular Assembly Kit framework are intuitive. However, more work is needed to extend MAKS-style security properties for more realistic adversaries. This effort ties in nicely with more recent results on modeling adversaries through their observational capabilities [78].

There is also potential in using the logical language for expressing information-flow properties, such as by syntactic manipulations of formulas based on a set of rules. Of particular interest is the recent work of Schairer et al. [52, 79] that focus on decomposing *global* confidentiality requirements into *local* confidentiality requirements, in order to construct secure systems from secure components. Roughly speaking, any fact about an agent a's confidential information being kept secret from a's observer can be seen as a *local* confidentiality requirement for agent a. Similarly, a *local* requirement for agent a with respect to agent b is a fact about confidential events of agent a being maintained secret from the observer of agent b. By contrast, a *global* confidentiality requirement is a confidentiality requirement that is not local, i.e., any requirement for maintaining secret confidential events of two or more agents.

We believe that we can gain insight by expressing the rules in a logic of knowledge. In the longer run, the goal is to put all the pieces together in a framework that allows us to (a) express global secrecy requirements as epistemic specifications, (b) manipulate the decomposition rules for refining the global requirement into local secrecy requirements, (c) extract knowledge-based programs that satisfy the secrecy specifications, and (d) suggest assumptions about the setting that allow knowledge-based tests in the extracted programs to be replaced with standard tests on agents' local states (and thus implementable in C/Java-like programming languages). The steps above should essentially remain the same when assumptions about the cryptographic model and the observational algorithms available to the adversaries vary.

CHAPTER 6

PROOF OF THEOREM ??

In this chapter we prove Theorem 2.6.1, which says that the flooding algorithm, LCR$'$, P1$'$, and P2$'$ de facto implement Pg_{cb}^{GC}. The most interesting cases are for the leader election in a ring, as it is not difficult to notice that the argument for the flooding protocol is mostly straightforward. 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.

6.1 The argument for LCR$'$

The pseudocode for LCR and LCR$'$ is given in Figures 6.1 and 6.2 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

```
$status := nonleader;\ maxid := id;\ val_R := \perp;\ done := 0$
$send_L(id)$
do until $done = 1$
   if $RQ \neq \perp$ then
      $val_R := dequeue(RQ)$
      if $(val_R = id)$ then
         $status := leader;\ send_L(\text{"}id\ is\ the\ leader\text{"});\ done := 1$
      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 6.1: The LCR protocol.

```
do until $(id \in val_R) \wedge (sent\ leader\ message \vee maxid = id_L)$
   if $some\_new\_info$ then
      if $((id \notin val_R \wedge max(val_R) > maxid) \vee (id \in val_R)$ then $send_L(new\_info)$
```

Figure 6.2: The LCR$'$ protocol.

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 his left neighbor, and whether 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 his 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^{GC}_{cb} 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, his neighbor will not eventually learn the information that it has or the function value. Since LCR$'$ solves the leader election problem, when agents do not send a message, they believe (correctly) that their neighbor will indeed learn the function value. So consider a situation where an agent i sends a message according to LCR$'$. (We do not want to set these variables explicitly, since then LCR' would not implement Pg^{GC}_{cb}, which does not have these assignment statements.) That means that either val_R is a message that says which id is the leader's id, or val_R contains an id larger than the value of $maxid$ so far. In the first case, if he does not forward his new information, then it is clear that all the agents 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. In the latter case, it is possible that the largest id seen so far is indeed the maximum in the ring. Then it is easy to see that, if i does not send his new information, then again i will never receive any further messages, and no agent 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^{GC}_{cb}, 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.

6.2 The argument for P2$'$

We start by describing P2. Since P2 works in undirected 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 his left. When an active agent receives val_R, it compares val_R to his id. If $val_R = id$ (which can happen only if i is active) then, as in the LCR protocol, i declares himself 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 his 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 6.3. Note that, since with P2$'$ agents keep track of all the information they have learned from the beginning of the protocol execution, the status of an agent can be inferred from the agent's local state: if the local state encodes an id larger than the agent's id, then the agent is passive; otherwise, the agent is active. Just as with LCR$'$, we have not made the assignments to val_R and wl explicit, since they can also be inferred from the local states (they are determined by the messages received by the agent).

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 give such a characterization despite the fact that we do not assume synchrony, nor that messages are received in FIFO order. As usual, we

```
$status := active;\ val_L := \perp;\ val_R := \perp;\ done := 0;\ wl = 0$
$send_L(id)$;
**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 6.3: Peterson's protocol P2.

use $(a_1, \ldots, a_k)^*$ to denote 0 or more repetitions of a sequence of actions $a_1, \ldots, a_k$. 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 6.2.1: *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 his last action, and this action was PR, then i's next action is PL; (ii) if i was active when it performed his last action, and this action was PL, then i's next action is PR; (iii) if i was passive when it performed his last action, and this action was SR, then i's next action is PR; and (iv) if i was passive when it performed his 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 his right, and this message was sent by R_i at time m', then

$$I_R(i,r,m+1) = I_R(i,r,m) \cup I_R(R_i,r,m')$$

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 his left, and this message was sent by L_i at time m', then

$$I_L(i,r,m+1) = I_L(i,r,m) \cup I_L(L_i,r,m')$$

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 $\mathit{heard\_from\_all}$, if $I_L(i,r,m) \cup I_R(i,r,m)$ contains all agents in the ring. More formally, $(\mathcal{J},r,m,i) \models \mathit{heard\_from\_all}$ 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 $\mathit{heard\_from\_all}$ 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 $\mathit{has\_all\_info}$ to be true 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 $\mathit{has\_all\_info}$ 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 6.4. (We describe what an agent does when he knows 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, just as with LCR′. 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 6.2.1 also applies to all runs r of P2′, times m, and agents i in N_r such that i did not know that he had all the information at time $m - 1$ in r.

```
$send_L(new\_info)$
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 6.4: The initial part of protocol P2′.

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 6.2.2: *For all runs r of P2′ and times m the following hold:*

(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 he 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 he 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 he had all the information when he last sent a message to i, then*

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

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

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

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 his 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 6.2.1 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 his left. It easily follows from Lemma 6.2.1 (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 his 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 $m_3 + 1 \leq m'$, 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 his 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

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

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 $val_R(i, r, m)$. By 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 $\max_{\{m' \leq m : val_R(i,r,m') \neq \perp\}} val_R(i, r, m')$. 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 m (as long as it does not contain all the ids). Thus, eventually at least one agent must know he has all the information. (Recall that we have not yet given the pseudocode for P2$'$ for the case that an agent knows that he has all the information.)

Corollary 6.2.3: *In all runs r consistent with P2$'$, eventually at least one agent knows that he 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'. This is not quite right, since the message sent by j is not msg, unless j is i's neighbor. Thus, we define what it means for msg *originated with j at time m'* by induction on the time m that msg was received by i. 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 if msg' was the last message received by R_i from its right before sending msg to i, and msg' was received by R_i at time m'', then msg 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 6.2.4: *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 6.2.1, 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 his pth message from the right, it follows from Lemma 6.2.1 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 6.2.1, we can think of P2$'$ as proceeding in phases while agents do not know that they have 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 6.2.5: *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 that he has 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 his $(p - l + 1)$st message, for $l = 2, \ldots, k$, provided that j_l processes his $(p-l+1)$st message before knowing that he has all the information; (iii) if $j_1 = i$, then j_l will be passive after having processed his $(p-l+3)$rd message, for $l = 2, \ldots, k$, provided that j_l processes his $(p-l+3)$rd message before knowing that he has all the information; and (iv) if $j_1 \neq i$, 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 his $(p - l + 1)$st message, for $l = 2, \ldots, k'$, provided that h_l processes his $(p - l + 1)$st message before knowing that he has all the information; (iii) if $h_1 = i$, then h_l will be passive after having processed his $(p-l+3)$rd message, provided that he processes his $(p-l+3)$rd message before knowing that he has all the information; and (iv) if $h_1 \neq i$, 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 6.2.4, 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 6.2.1 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 his $(p-2)$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 that he has 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 that he has all the information at some time m and no agent knows that he has 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.[1] 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 6.2.6: *If $N_r = N_{r'} = N$, no agent knows that he has 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 6.2.1 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 6.2.1 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

[1]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.

immediately. ∎

Lemma 6.2.7: *If an agent that can be first to learn all the information is active when he learns all the information, then he must be the agent with the largest id. Moreover, there are at most two agents that can be first to learn all the information in network N.*

Proof: We begin with the first claim. Suppose that i can be first to learn all the information and is active when he learns all the information. That means that there must be a run r with $N_r = N$ such that i learns all the information in r at some time m, and no agent has all the information at point $(r, m-1)$. By Lemma 6.2.5, if $j_1, \ldots, j_k$ are the active agents at time m in $I_R(i,r,m)$, listed in the order of closeness to i's right, then $id_{j_1} > \ldots > id_{j_k}$; since i is active at time m, this means that i has the largest id in $I_R(i,r,m)$. Similarly, we can show that i has the largest id in $I_L(i,r,m)$. Since i has all the information at time m, $I_L(i,r,m) \cup I_R(i,r,m)$ must be the full ring. Thus, i has the largest id in the network.

For the second claim, suppose, by way of contradiction, that more than two agents can be the first to learn all the information in a run r of P2$'$. Let i^* be the agent in N_r with the largest id. If i is an agent that is among the first to learn all the information, suppose that it was p_ith message that i received that resulted in i knowing that he has all the information. It easily follows from Lemma 6.2.5 that if $i \neq i^*$, then either the p_ith message or the $(p_i - 1)$st message that i processed must have originated with i^*.

First suppose that i^* is not among the first agents to learn all the information. Then there must be two agents who are among the first to learn all the information that receive messages that originated with i^* from the same side. Suppose, without loss of generality, that these two agents are i_1 and i_2, and that they receive the message from i^* from the right. Now a simple case analysis shows that either i_1 or i_2 cannot be the first to learn all the information in r. For suppose that the message that originated with i^* is the p'_hth message that i_h processed, for $h = 1, 2$. (By Lemma 6.2.6, p'_h is same in all runs r' such that $N_r = N_{r'}$.) If $p'_1 > p'_2$, then it follows from Lemma 6.2.1 that $p'_1 \geq p'_2 + 2$. It easily follows that $p_{i_1} < p_{i_2}$, and that i_i must receive any message that originates from i^* and comes from the right before i_2 does. It follows that i_1 must learn all the information before i_2 does. 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 $p'_1 = p'_2$. Without loss 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 $p'_1 = p_{i_1}$, so that i_1 knows he has all the information after processing the message from i^*, then i_1 knows he has all the information before i_2 in all runs r' with $N_r = N_{r'}$, while if $p_{i_1} = p'_1 + 1$, then i_1 must learn it after i_2 in all runs r' with $N_{r'} = N_r$ (since the p_1th message processed by i_1 must originate with an agent farther to the left of i^* than i_2, and there can be no active agents between i_1 and i_2 at this point). Thus, it cannot be the case that both i_1 and i_2 can be first to learn all the information in r, a contradiction.

Thus, we can assume that i^* is among the first agents to learn all the information. Let i_1 and i_2 be two agents other than i^* that are among the first agents to learn all the information. 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 these are the p'_1th and p'_2th messages processed by i_1 and i_2, respectively, and that i^* processes his p_{i^*}th message from the left. We cannot have $p'_2 = p_{i^*}$. If $p'_2 < p_{i^*}$, then it is easy to show that i_2 must learn all the information before i^* in all runs r' with $N_{r'} = N_r$, while if $p_{i^*} < p'_2$, then i^* must learn all the information before i_2 in all runs r' with $N_{r'} = N_r$ (indeed, i^* must know that he has all the information before he sends his message to i_2). Either way, we have a contradiction. ▮

The next result shows that whether an agent is active or passive after having received p messages in run r depends only on N_r.

Lemma 6.2.8: *Suppose that r and r' are two runs such that $N_r = N_{r'}$. For all agents i, i has the same state and status after processing the first p messages in r as $f(i)$ in r'.*

Proof: A straightforward induction on p shows that, for all agents i, the first p messages that i receives in r and r' are the same, and they come from the same direction in these two runs. Since i's state and status depends only on the messages it receives and the direction they come from, the lemma immediately follows. We leave details to the reader. ▮

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 would be more enlightening.

- Suppose that the only agent that can be first to learn all the information is the agent i^* with the largest id. We now do essentially what is done in Peterson's algorithm. Suppose that the message that resulted in i^* 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 i^*. It follows from parts (a)(ii) and (b)(ii) of Lemma 6.2.5 (taking i to be i^* and using the fact that $I_L(i^*, r, m) \cup I_R(i^*, r, m)$ consists of all agents in the ring if i^* learns all the information at the point (r, m)) that after processing the pth message, all agents other than i^* will be passive. Agent i^* then sends his $(p + 1)$st message to the left. After an agent processes the i^*'s $(p + 1)$st message, he will then know that he has all the information. We require it to send a message to the left with all the information unless he is i^*'s right neighbor. (Of course, once he knows that he has all the information, i^*'s right neighbor will realize that the neighbor to the left is the leader and that i^* already knows all the information, so he does not need to forward the information.) After this process is completed, all the agents know that they have all the information.

- Suppose that agent i is the only agent that can be first to learn 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 that they have all the information.(Recall that it follows from Lemma 6.2.8 that, with P2$'$, whether an agent is active or passive after processing the pth message in run r depends only on N_r.)

- Suppose that two agents, say i and i', can be first to know that they have all the information, and that i are i' are passive when they 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 6.2.5, 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 two agents, say i and i^*, can be first to learn all the information, and that i is passive when he learns all the information, while i^* is active (and is thus the agent with the highest id). 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 his 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$'$.

We can now finally prove that P2$'$ de facto implements Pg_{cb}^{GC} in the interpreted context $(\gamma^{br,u}, \pi)$ where the networks encoded in the initial states are the undirected 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 6.2.9: *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

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

Since R is the unique name that i's left neighbor calls i in a ring, we have that $(\mathcal{J}, r, m, i) \models Calls(L, I, R)$. By the definitions in Section 2.5.2, $(\mathcal{J}, r, m, i) \models \varphi_{send_L(new\_info)}$ 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) the following holds

$$(\mathcal{I}, r', m', i') \models \neg[\neg do(send_L(new\_info)) > \Diamond(B_L(Acc_R new\_info) \vee (\exists v.\ B_L(f = v)))],$$

so there exists a situation

$$(r'', m'', i'') \in \mathtt{closest}([\![\neg do(send_L(new\_info))]\!]_{\mathbf{I}(P2', \chi^{br,u})}, r', m', i')$$

such that

$$(\mathcal{J}, r'', m'', i'') \models \Box(\neg B_L(Acc_R new\_info) \wedge \neg B_L(f)).$$

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 that he has all the information, and sends his 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 he learn who the leader is.
- Case 2: i is active, does not know all the information, and does not send his first message to the left at time m. In this case, L_i must be passive. Suppose that i is about to send his kth message left at the point (r,m). By Lemma 6.2.1, 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 his kth message left, and is waiting to process his 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 $\texttt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r', m, i)$. 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 he 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 he is passive, i must have last processed a message from his right; without loss of generality, assume that i has processed p messages from his right, and so must have processed $(p-1)$ messages from his left by time m. It easily follows from Lemma 6.2.1 that $p > 1$. Suppose that the $(p-1)$st message that i processed from his left originated with k. Since i does not have all the information at time m, k did not have all the information when he sent this message to the right. After receiving his $(p-1)$st message from the left, i must consider it possible that the ring is sufficiently large that, even after k processes his $(p-1)$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 his $(p-1)$st message from the left by this point. Let (r'', m'', i'') be an arbitrary situation in $\texttt{close}(\overline{do(send_L(new\_info))}, P2', \gamma^{br,u}, r', m', i')$. 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 his $(p-1)$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 agents 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 his 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 he learns all the information as a result of processing a message 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 his 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 know that they 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

$$(\mathcal{J}, r, m, i) \models \neg do(send_L(new\_info)) > \Diamond((B_L(Acc_R new\_info)) \vee (\exists v.\, B_L(f = v))).$$

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

BIBLIOGRAPHY

[1] F. Afrati, C. H. Papadimitriou, and G. Papageorgiou. The synthesis of communication protocols. In *PODC '86: Proceedings of the Fifth Annual ACM Symposium on Principles of dDistributed Computing*, pages 263–271, 1986.

[2] A. V. Aho, J. D. Ullman, A. D. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Computers and Mathematics with Applications*, 8(3):205–214, 1982. This is a later version of [3].

[3] A. V. Aho, J. D. Ullman, and M. Yannakakis. Modeling communication protocols by automata. In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pages 267–273. 1979.

[4] D. Angluin. Local and global properties in netwroks of processors. pages 82–93, 1980.

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

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

[7] R. J. Back. On the correctness of refinement steps in program development. PhD thesis, Report A-1978-4, Department of Computer Science, University of Helsinki, 1978.

[8] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12:260–261, 1969.

[9] J. L. Bates and R. L. Constable. Proofs as programs. *ACM Transactions on Programming Languages and Systems*, 7(1):53–71, 1985.

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

[11] M. Bickford and R. L. Constable. A logic of events. Technical Report TR2003-1893, Cornell University, 2003.

[12] M. Bickford and R. L. Constable. A causal logic of events in formalized computational type theory. Report, Cornell University, 2005.

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

[14] L. E. J. Brouwer. On the significance of the principle of excluded middle in mathematics, especially in function theory. *J. für die Reine und Angewandte Mathematik*, 154:1–7, 1923.

[15] K. M. Chandy and J. Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[16] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, Mass., 1988.

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

[18] R. L. Constable. Naive computational type theory. In U. Berger and H. Schwichtenberg, editors, *Proof and System-Reliability, Proceedings of International Summer School, vol. 62 of NATO Science Series III*, pages 213–260, Amsterdam, 2002. Kluwer Academic Publishers.

[19] R. L. Constable, S. F. Allen, H. M. Bromley, W. R. Cleaveland, J. F. Cremer, R. W. Harper, D. J. Howe, T. B. Knoblock, N. P. Mendler, P. Panangaden, J. T. Sasaki, and S. F. Smith. *Implementing Mathematics in the NuPRL Proof Development System*. Prentice-Hall, 1986.

[20] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System*. Prentice-Hall, NJ, 1986.

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

[22] E. A. Emerson and J. Y. Halpern. "Sometimes" and "not never" revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[23] K. Engelhardt, R. van der Meyden, and Y. Moses. A program refinement framework supporting reasoning about knowledge and time. In J. Tiuryn, editor, *Proc. Foundations of Software Science and Computation Structures (FOSSACS 2000)*, pages 114–129. Springer-Verlag, Berlin/New York, 1998.

[24] K. Engelhardt, R. van der Meyden, and Y. Moses. A refinement theory that supports reasoning about knowledge and time for synchronous agents. In *Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, pages 125–141. Springer-Verlag, Berlin/New York, 2001.

[25] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

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

[27] R. Fagin, J. Y. Halpern, and M. Y. Vardi. What can machines know? On the properties of knowledge in distributed systems. *J. ACM*, 39(2):328–376, 1992.

[28] R. Fagin and M. Y. Vardi. Knowledge and implicit knowledge in a distributed environment: preliminary report. In *Theoretical Aspects of Reasoning about Knowledge: Proc. 1986 Conference*, pages 187–206. 1986.

[29] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. Massachusetts Institute of Technology, 1995.

[30] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3:5–33, 1995.

[31] R. Focardi and R. Gorrieri. Classification of security properties (Part I: Information flow). In *Foundations of Security Analysis and Design*, pages 331–396. Springer, 2001.

[32] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, N. J., 1962.

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

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

[35] J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symposyum on Security and Privacy*, pages 75–86. 1984.

[36] J. W. Gray, III and P. G. Syverson. A logical approach to multilevel security of probabilistic systems. *Distributed Computing*, pages 73–90, 1998.

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

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

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

[40] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989. A preliminary version appeared in *Proc. 4th ACM Symposium on Principles of Distributed Computing*, 1985, with the title "A formal model of knowledge, action, and communication in distributed systems: preliminary report".

[41] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989.

[42] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[43] J. Y. Halpern and Y. Moses. Using counterfactuals in knowledge-based programming. 17(2):91–106, 2004.

[44] J. Y. Halpern and Y. Moses. Characterizing solution concepts in games using knowledge-based programs. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1300–1307, 2007.

[45] J. Y. Halpern, Y. Moses, and M. Y. Vardi. Algorithmic knowledge. In *Proc. 5th Conference on Theoretical Aspects of Reasoning about Knowledge (TARK'94)*, pages 255–266. Morgan Kaufmann, 1994.

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

[47] J. Y. Halpern and K. O'Neill. Secrecy in multiagent systems. In *Proc. 15th IEEE Computer Security Foundations Workshop*, pages 32–46, 2002. To appear, *ACM Transactions on Information and System Security*.

[48] J. Y. Halpern and S. Petride. A knowledge-based analysis of global function computation. In *Proceedings of the 20th International Symposium on Distributed Computing*, pages 136–150, 2006.

[49] J. Y. Halpern and R. Pucella. Modeling adversaries in a logic for reasoning about security protocols. In *Proc. Workshop on Formal Aspects of Security (FASec'02)*, volume 2629 of *Lecture Notes in Computer Science*, pages 115–132, 2002.

[50] J. Y. Halpern and L. D. Zuck. 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, 1992.

[51] D. Hutter. Possibilistic information flow control in MAKS and action refinement. In *Proceedings of the International Conference on Emerging Trends in Information and Communication Security, ETRICS*, pages 268–281, 2006.

[52] D. Hutter, H. Mantel, I. Schaefer, and A. Schairer. Security of multi-agent systems: A case study on comparison shopping. *Journal of Applied Logic*, 5(2):303–332, 2007.

[53] D. Hutter and A. Schairer. Possibilistic information flow control in the presence of encrypted communication. In *Proc. 9th European Symposium on Research in Computer Security (ESORICS'04)*, volume 3193 of *Lecture Notes in Computer Science*, pages 209–224. Springer-Verlag, 2004.

[54] R. E. Johnson and F. B. Schneider. Symmetry and similarity in distributed systems. pages 13–22, 1985.

[55] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[56] G. Le Lann. Distributed systems—towards a formal approach. In *IFIP Congress*, volume 7, pages 155–160, 1977.

[57] D. K. Lewis. *Counterfactuals*. Harvard University Press, Cambridge, Mass., 1973.

[58] P. M. Löf. Constructive mathematics and computer programming. *Royal Society of London Philosophical Transactions Series A*, 312:501–518, 1984.

[59] N. Lynch and M. Tuttle. An introduction to Input/Output automata. *Centrum voor Wiskunde en Informatica*, 2(3):219–246, September 1989.

[60] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. Also available as MIT Technical Memo MIT/LCS/TM-373.

[61] N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, 1997.

[62] H. Mantel. Possibilistic definitions of security—an assembly kit. In *Proc. 13th IEEE Computer Security Foundations Workshop*, pages 185–199. 2000.

[63] H. Mantel. A uniform framework for the formal specification and verification of information flow security. PhD Thesis, Universitat des Saarlandes, 2003.

[64] H. Mantel. The framework of selective interleaving functions and the modular assembly kit. In *FMSE '05: Proceedings of the 2005 ACM workshop on Formal methods in security engineering*, pages 53–62. ACM, 2005.

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

[66] D. McCullough. Specifications for multi-level security and a hook-up property. In *Proc. IEEE Symposium on Research on Security and Privacy*, pages 161–166. 1987.

[67] J. McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Security and Privacy*, pages 79–93. 1994.

[68] John McLean. Security models and information flow. In *Proc. IEEE Symposium on Security and Privacy*, pages 180–187, 1990.

[69] R. Milner. *Communication and Concurrency*. Prentice Hall, Hertfordshire, 1989.

[70] B. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Trans. on Software Engineering*, 7:417–426, 1981.

[71] T. Mizrahi and Y. Moses. Continuous consensus via common knowledge. *Distributed Computing*, 20(5):305–321, 2008.

[72] Y. Moses and G. Roth. On reliable message diffusion. In *Proc. 8th ACM Symp. on Principles of Distributed Computing*, pages 119–128, 1989.

[73] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[74] G. Neiger and R. A. Bazzi. Using knowledge to optimally achieve coordination in distributed systems. *Theoretical Computer Science*, 220(1):31–65, 1999.

[75] C. O'Halloran. A calculus of information flow. In *Proc. of European Symposium on Research in Information Security*, pages 147–159. 1990.

[76] P. Panangaden and S. Taylor. Concurrent common knowledge: defining agreement for asynchronous systems. *Distributed Computing*, 6(2):73–93, 1992.

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

[78] S. Petride and R. Pucella. Perfect cryptography, S5 knowledge, and algorithmic knowledge. In *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2007)*, pages 239–247, 2007.

[79] A. Schairer. Towards using possibilistic information flow control to design secure multiagent systems. *Security in Pervasive Computing, LNCS*, 2802:101–115, 2004.

[80] R. C. Stalnaker. A semantic analysis of conditional logic. In N. Rescher, editor, *Studies in Logical Theory*, pages 98–112. Oxford University Press, 1968.

[81] M. V. Stenning. A data transfer protocol. *Comput. Networks*, 1:99–110, 1976.

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

[83] R. van der Meyden and M. Y. Vardi. Synthesis from knowledge-based specifications. In *Proc. Ninth International Conference on Concurrency Theory (CONCUR'98)*, pages 34–49, 1998.

[84] R. van der Meyden and T. Wilke. Synthesis of distributed systems from knowledge-based specifications. Technical Report UNSW-CSE-TR-0504, University of New South Wales, 2005.

[85] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 144–161. 1990.

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

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

[88] A. Zakinthinos and E.S. Lee. A general theory of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 94–102. 1997.