Knowledge and Communication Complexity in Distributed Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Daniel Pfleger, BSc
Matrikelnummer 1125864

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Ulrich Schmid

Wien, 8. März 2018

Daniel Pfleger　　Ulrich Schmid

FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Knowledge and Communication Complexity in Distributed Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Daniel Pfleger, BSc
Registration Number 1125864

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Ulrich Schmid

Vienna, 8th March, 2018

Daniel Pfleger   Ulrich Schmid

Erklärung zur Verfassung der Arbeit

Daniel Pfleger, BSc
Neilreichgasse 85/8/10, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 8. März 2018

Daniel Pfleger

Acknowledgements

First of all I want to express my deepest thanks to Prof. Ulrich Schmid for introducing me to the interesting topics in distributed algorithms and for his excellent guidance, counsel and support throughout my master study and especially during the work on this thesis. I also want to thank the research assistants Kyrill Winkler and Manfred Schwarz for their refreshing support during the work on this thesis and for poking on weak points in my arguments during various interesting and enlightening discussions, which opened my eyes to take alternative ways at some points. My family and friends deserve special thanks for backing me and for adding variety to stressful times. Last but not least I want to thank Manuela Hofer for her inexhaustive love and the patience in stressful times and sleepless nights full of work.

This thesis has been supported by the Austrian Science Fund FWF under the projects ADynNet (P28182) and RiSE/SHiNE (S11405).

Abstract

This thesis is concerned with the connection between *knowledge* and *communication complexity* in distributed systems. To find a lower bound on communication complexity for a problem $\mathcal{P}$, we pursue a two-step approach: First we determine the necessary knowledge the processes must locally acquire to solve $\mathcal{P}$. From this required knowledge and the processes' a priori knowledge, we can infer what the processes have to learn throughout the execution. From this learning process, we can determine a lower bound on communication complexity.

More specifically, we bridge the gap between *Action Models*, used in *Dynamic Epistemic Logic* to update the *epistemic state* of the system, and communication complexity and propose a way to determine a lower bound on the number of bits that need to be sent applying an action model in the general setting.

As this thesis shall also support the chase for a *strongest message adversary* for the problem of solving *consensus in directed dynamic networks*, we apply our method in the case of two processes to find a lower bound on the communication complexity for this problem. It turns out, however, that there is no such bound.

Nevertheless, as a by-product of our analysis, we provide necessary and sufficient conditions for a message adversary that allows to solve consensus in a directed dynamic network of two processes.

Kurzfassung

Diese Arbeit behandelt die Verbindung zwischen *Wissen* und *Kommunikations-Komplexität* in verteilten Systemen: Wir möchten eine untere Schranke für die Komplexität der Kommunikation ermitteln, die benötigt wird um ein Problem $\mathcal{P}$ zu lösen. Dazu verfolgen wir einen zweistufigen Ansatz: Zuerst ermitteln wir das Wissen, das die Prozesse benötigen, um $\mathcal{P}$ zu lösen. Ausgehend von dem a priori Wissen der Prozesse schließen wir darauf, was die Prozesse lernen müssen, um dieses benötigte Wissen zu erlangen. In einem zweiten Schritt wird die Kommunikation-Komplexität ermittelt, die für diesen Lernvorgang mindestens erforderlich ist.

Insbesondere untersuchen wir die Lücke zwischen *Action Models*, die in *Dynamic Epistemic Logic* verwendet werden, um Änderungen im *epistemischen Zustand* eines Systems zu formalisieren, und Kommunikations-Komplixität, konkret, wir schlagen eine Methode vor, um eine untere Schranke für die Anzahl der Bits zu finden, die gesendet werden müssen, um ein Action Model in einer allgemeinen Situation anzuwenden.

Darüberhinaus soll diese Arbeit die Suche nach einem *stärksten Message Adversary* für *Konsensus in gerichteten dynamischen Netzwerken* unterstützen. Wir wenden daher unsere Methode auf ein Zwei-Prozess-System an, um eine untere Schranke für die Kommunikations-Komplexität dieses Problems zu finden. Allerdings stellt sich heraus, dass die Kommunikations-Komplexität für Konsensus in gerichteten dynamischen Netzwerken unbeschränkt ist.

Dennoch liefern wir, als Nebenprodukt unserer Analyse, Bedingungen an den Message Adversary, die notwendig und hinreichend dafür sind, dass zwei Prozesse in einem gerichteten dynamischen Netzwerk Konsensus lösen können.

Contents

CHAPTER 1

Introduction

The main problem we will consider in this theses is the problem of *Consensus in Directed Dynamic Networks*. In the consensus problem, all the n processes of a distributed system have to decide on a certain value, given some local input at every process. This problem is well known and has been solved in various variants of distributed message-passing systems with reliable communication [LSP82, DFF$^+$82, ADG84, BT83, Bra84, FLP85]. In dynamic networks, on the other hand, the problem is more complex, as the processes cannot rely on the delivery of sent messages [CFQS12]. There is hence on-going research in this area [SWK09, KOM11, BRS12, BRS$^+$15].

An important abstraction in synchronous dynamic networks are *message adversaries*, which determine which messages are lost and which are delivered in every round. One of the most interesting questions in this area of research is to find a *strongest message adversary* that still allows to solve consensus. A strongest message adversary is such that it admits a correct consensus algorithm whereas any further relaxation of its guarantees does not allow to find such an algorithm.

A conceivable way to approach this question could be the following:

1. Determine the *knowledge* a process has to gather to be able to decide on a value according to the consensus specification.
2. Using this knowledge, determine the minimal number of bits a process has to receive to gather this knowledge.
3. Find a message adversary that allows to send exactly this minimal number of bits.
4. Check whether this message adversary is a strongest one, by (i) finding a correct consensus algorithm for it, and (ii) proving consensus impossibility for every relaxation.

As a preliminary step for such an approach, part of this thesis is concerned with the idea to connect the communication complexity for a general problem $\mathcal{P}$ with the necessary and sufficient knowledge that must be attained by the processes to solve $\mathcal{P}$, using *epistemic logic*. Our idea is to do this in three steps:

1. What do the processes have to *know* to solve $\mathcal{P}$?
2. What does a single process have to *learn* from other processes to gain its part of this knowledge?
3. How many bits have to be sent between the processes during this learning phase?

We will investigate the connection between knowledge gain and communication complexity in order to bridge the gap between steps 2 and 3. Subsequently, we will apply our approach to find a lower bound on communication complexity for the consensus problem in directed dynamic networks consisting of two processes.

1.1 Short Overview of Related Work

[Yao79] found methods to get communication complexity lower bounds for deterministic two-player games. We will take a look at the *fooling set* method, which has been implicitly used by [Yao79] and made more explicit in [LS81]. Another such method is the *rank lower bound* introduced in [MS82]. Among others, [DKW09] and [DF89] generalized the two-player setting to multi-party communication complexity. Concerning communication complexity, our presentation is based on [KN97], which gives a good overview of different methods and models.
[Ger97] invented *Dynamic Epistemic Logic* as a combination of epistemic logic and dynamic semantics. Epistemic logic [Hin62] allows to formally reason about knowledge and belief in multi-agent systems. Dynamic Epistemic Logic is used to argue about knowledge and communication-induced knowledge gain. We will especially use *Action Models* to reason about knowledge gain, by updating the epistemic state of the system. In a nutshell, action models allow to model the actions of players or the environment, which can alter the knowledge of other players. [vDvdHK08] gives an outline of different methods for reasoning about knowledge.
We will use small problems like the *Cheating Husbands* problem [MDH86] to illustrate our explanations. [FHMV95] gave a deep look into reasoning about knowledge, including the *Muddy Children* problem, which is very closely related to the Cheating Husbands Problem.

Similar to our work is [CK08], which used dynamic epistemic logic and action models in a combinatorial way to find a lower bound on communication complexity for the Russian Cards problem. [ALNR09] investigated bounds for a system of reasoning agents, where agents may have different knowledge and inferential capabilities and have to draw conclusions from received messages, which contain formulas. They established a framework to verify time, memory and communication bounds in such a system. The communication complexity in this work is defined as the number of sent formulas rather than sent bits, however.

An overview of the related work on solving consensus will be presented in Section 5.2.

1.2 Thesis Structure and Major Contributions

This thesis has been supported by the Austrian Science Fund FWF under the projects ADynNet (P28182) and RiSE/SHiNE (S11405).

The thesis is structured as follows:

- Section 1.3 defines the model of computation used throughout the thesis, including the notions of a *synchronous message passing system*, *message adversaries* and *indistinguishability*.
- Chapter 2 gives an overview of the topic of *communication complexity*. We will present the model given by [Yao79], followed by techniques [Yao79, LS81, MS82] to find lower bounds on the communication complexity of a problem.
- Chapter 3 introduces the notion of *knowledge* in distributed systems. We will introduce two small problems: *Buy or Sell?* and *Cheating Husbands*, which will be used to illustrate the content of Chapter 3 and Chapter 4. To reason about knowledge and knowledge gain, we will also present *epistemic logic* and *action models*, a well-known method to reason in *dynamic epistemic logic* [Ger97].
- Chapter 4 investigates the connection between knowledge gain and communication complexity. In particular, we show that an algorithm $\mathcal{A}$ is defined by a sequence of action models $\mathtt{AM}_1, \dots, \mathtt{AM}_m$ and show that the application of the composed actions model $\mathtt{CAM}_m = (\mathtt{AM}_1; \dots; \mathtt{AM}_m)$ implies a lower-bound on the communication complexity of the algorithm $\mathcal{A}$.
- Chapter 5 is concerned with the problem of solving *consensus* in *directed dynamic networks* consisting of 2 processes. Note that this is not at all a toy problem, as the rich literature on consensus with lossy links [SW89, CBS09, SWK09, CG13] reveals.
  - Section 5.3 tries to apply the approach of Chapter 4 to this problem in order to find a lower bound on its communication complexity. It turns out, however, that the latter is unbounded (Theorem 5.3.1).
  - Since this thesis shall also contribute to the chase for necessary and sufficient conditions to solve consensus in directed dynamic networks, we further investigated this problem in 2 process systems. Section 5.4 defines a message adversary, which is necessary and sufficient to solve consensus in this setting, as proved in Theorem 5.4.4.

- Chapter 6 concludes this thesis, by giving a summary and a perspective of open questions.

1.3 The Model

We consider synchronous message passing systems only. Such systems are modeled as a set Π of n processes with unique identifiers, which are modeled as state machines. The processes themselves are reliable and operate in lock-step rounds. At the beginning of round r, all processes are allowed to send out messages to each other (and to themselves). The actual communication in round r is controlled by an omniscient *message adversary*, which determines which messages are delivered and which messages get lost in this round. I.e., in each round r, the message adversary determines (we often also say chooses) a communication graph $\mathcal{G}^r = (V, E^r)$, such that: (i) each vertex in V corresponds to exactly one process in Π and (ii) iff the message adversary would permit the delivery of a messages sent from p to q in round r, the edge from p to q is present in E^r. Note that each $\mathcal{G}^r$ contains all self-loops, i.e., each process p always sends a message to itself in each round r and always receives this message. Rounds are *communication closed*, in the sense that each message sent in some round r may either be delivered in r or not at all. After this message exchange, each process does an *instantaneous local computation step*.

For our analysis, we need the notions of the *local state* of a process. The *local state* L_p is the set of all the local variables of process p, L_p^r denotes the local state of process p at the end of round r, L_p^0 denotes the initial state of p. x_p^r denotes the value of some variable x in the local state L_p^r. At the beginning of round r (before the message exchange), process p has the local state L_p^{r-1}. Using this local state and the contents of the received messages, p computes the new local state L_p^r during its computation step of round r, via a *state-transition function* that encodes the algorithm executed by p. Similarly, it determines the contents of the messages it tries to send in round $r+1$, via an appropriate *message-sending function*.

We also denote the vector of the local states of all processes at the end of round r as the *configuration* C^r; C^0 is the initial configuration. The sequence of configurations and communication graphs is called an *execution* or *run*. As we only consider deterministic algorithms, such an execution (C^0, σ) is completely determined by a given initial configuration C^0 together with an infinite sequence of communication graphs σ that is controlled by the message adversary. We say that two configurations C and C' are *indistinguishable* for a process p, denoted by $C \sim_p C'$, if its local state is the same in both configurations, $L_p = L'_p$. Two prefixes σ_r and σ'_r of graph sequences are called indistinguishable, if the configurations $C = \langle C^0, \sigma_r \rangle$, $C' = \langle C^0, \sigma'_r \rangle$ reached by applying σ_r, σ'_r to C^0 are indistinguishable.

We denote a sequence of graphs from round a to round b by $(\mathcal{G}^r)_{r=a}^b$; an infinite graph sequence (also simply called graph sequence in the following) is denoted by $(\mathcal{G}^r)_{r=1}^\infty$, typically abbreviated by σ or ε. A prefix of length k of a graph sequence $\sigma = (\mathcal{G}_\sigma^r)_{r=1}^\infty$ is written as $\sigma_k = (\mathcal{G}_\sigma^r)_{r=1}^k$. A single graph in round r of the sequence σ is denoted by $\mathcal{G}_\sigma^r$.

A message adversary MA is defined by a set of properties P_{MA}, which must be fulfilled by each of the graph sequences. Thus, MA can be specified via the set of graph sequences $MA := \{(\mathcal{G}^r)_{r=1}^{\infty} \mid (\mathcal{G}^r)_{r=1}^{\infty} \text{ satisfies } P_{MA}\}$ it may choose. A problem is *impossible to solve* under a message adversary MA, if there does not exist any deterministic algorithm that solves the problem for each graph sequence $(\mathcal{G}^r)_{r=1}^{\infty} \in MA$. It is easy to see that no problem that needs at least some communication can be solved under a message adversary that includes the graph sequence where no communication graph contains even a single edge.

We will assume that all algorithms considered in this thesis are *full-graph-history*, i.e., processes keep track of all received messages and append the current history to every message they send.

CHAPTER 2

Communication Complexity

Whenever two or more parties — be it humans, computers, processes or something else — need to solve a problem jointly, they have to communicate with each other. This communication may be implicit — e.g. via communication by time [Lam78, BZM14] — or explicit — by sending a message or "talking to each other". Once there is a solution (a distributed algorithm) for such a problem that needs to be solved jointly, one usually thinks about the "goodness" of the solution in order to compare it to another solution for the same problem: We want to be able to state that solution A is better or equal to solution B. Thus we need some measure of the *complexity* of a solution.
The complexity of solutions can also be used to compare problems to each other: Is problem A more complicated than problem B? If we have optimal algorithms (in terms of complexity) solving A respectively B, we can use their complexity as measures of the problem complexity of A respectively B.

One often uses the *amount of communication*, which is known as *communication complexity*, as a measure of the complexity. If considered as a complexity measure of a solution for problem $\mathcal{P}_r$, we will restrict our attention to the *number of bits/messages solution A sends to solve problem $\mathcal{P}_r$*. Since many problems in distributed computing (like consensus) are parameterized via some input (the vector of the processes input values), we will focus on the worst-case complexity here, taken over all possible inputs. Talking about the complexity of the problem itself, we use the *minimal number of bits/messages that any solution for problem $\mathcal{P}_r$ has to send to solve $\mathcal{P}_r$ for the worst-case input.*

The simplest scenario is the two-party model defined by [Yao79]. It consists of two processes only, which have some local input data. The problem $\mathcal{P}_r$ at hand is to locally compute a certain function of both inputs, and the only resource we care about is communication. Although the model allows to handle issues like randomization and non-determinism, in this work, we restrict ourselves to the deterministic case.

In this chapter, we will give an overview of this model and the lower-bound techniques related to it. The content and structure is based on [KN97].

2.1 The Scenario and Model

Let X, Y and Z be arbitrary finite sets and $f\colon X \times Y \to Z$ a non-constant function. There are two processes p_0 and p_1, which jointly solve the problem of evaluating $f(x,y)$, for some inputs $x \in X$ and $y \in Y$: x is only known to p_0, while only p_1 knows y. Thus p_0 and p_1 have to communicate with each other in order to solve the problem. Our communication model assumes that the processes send information to each other alternatingly: one bit is sent by p_0 then one bit is sent by p_1 and so on, according to some protocol $\mathcal{P}$. The communication complexity for the problem (function) f is the least number of bits (= the number of messages) that need to be exchanged between p_0 and p_1 by any deterministic protocol $\mathcal{P}$ in order to determine $f(x,y)$ at p_0 or p_1. We assume that the process that can compute $f(x,y)$ for the first time sends a special $\perp$ message to the other process and stops. For simplicity, we will assume that $\perp$ requires 0 bits, e.g., by using communication-by-time.

Note that, in this model introduced in [Yao79], it is assumed that the processes know both the identity (id) of themselves and the other process. This allows a protocol to specify the process sending the first bit (e.g., the process with the minimal identity), making it an *asymmetric* model. In distributed computing, the processes usually know their id, but do not know the id of any other process, giving raise to a *symmetric* model. Note that a leader election algorithm can be used to enable an asymmetric model abstraction on top of a symmetric model. In the symmetric model, one cannot fix a priori a process that will send the first bit in the communication. [DMR08] studies the communication complexity of leader election, consensus and maximum finding in ring and chain networks in the symmetric model.

Example 2.1.1. Consider the example $f(x,y) = x \wedge y$, where $f(x,y) = 0$ iff at least one bit in x or y is 0. In the trivial protocol, p_0 sends its input to p_1 bit by bit, p_1 replies with any bit. p_1 computes $f(x,y)$ and sends $\perp$ to p_0, resulting in $2 \cdot log_2|X|$ bits sent in total. But there is also a solution using exactly 1 communicated bit: p_0 determines the AND of all the bits of x and sends this bit to p_1, p_1 determines the AND of all the bits of y and the received bit. Since there cannot be any protocol solving the problem with less than one bit of communication, the communication complexity for this function is 1.

2.1.1 The Model

We use a synchronous distributed system model, as defined in Section 1.3. As we are only interested in the number of bits sent between p_0 and p_1, we consider the communication link from p_0 to p_1 and vice versa reliable — each message sent in round r will be delivered in the same round.

We are only interested in deterministic protocols:

Definition 2.1.1. A *deterministic protocol* $\mathcal{P}$ (for function $f\colon X \times Y \to Z$) must specify:

- the process which sends the first bit in round 1, w.l.o.g. p_0,
- a deterministic function a_k (resp. b_k) for p_0 (resp. p_1) to determine the value of the bit s_k^0 (resp. s_k^1) sent in round k by p_0 (resp. p_1), such that s_k^i only depends on the own input x (resp. y) and the received bits $(s_0^1, \dots, s_{k-1}^1)$ (resp. $(s_0^0, \dots, s_{k-1}^0)$),
- and a way to determine for both processes whether the result $f(x,y)$ has been determined already by the end of the current round, w.l.o.g. implemented by sending $\perp$ to the other process in the next round.

Since we do not want to execute any protocol $\mathcal{P}$, but rather analyze it, we give an equivalent alternative definition of a protocol.

Definition 2.1.2 ([KN97], Definition 1.1)**.** A *protocol* $\mathcal{P}$ over $X \times Y$ with range Z is a binary tree, where each internal node v is labeled either by a function $a_v : X \to \{0,1\}$ or by a function $b_v : Y \to \{0,1\}$, and each leaf is labeled with an element z of Z. The root r is labeled by $a_1 : X \to \{0,1\}$. Intuitively, a_k (resp. b_k) gives the bit sent by p_0 (resp. p_1) in round k.

The *value of the protocol* $\mathcal{P}$ on input (x,y) is the label of the leaf reached by starting from the root, and walking on the tree, as follows: At each node v labeled by a_v (resp. b_v) walk left if $a_v(x) = 0$ (resp. $b_v(y) = 0$) and right if $a_v(x) = 1$ (resp. $b_v(y) = 1$).

The *cost of the protocol* $\mathcal{P}$ on input (x,y) is the length of the path taken on input (x,y). As the longest such path is the height of the protocol tree, the maximal cost over all inputs is the height of this protocol tree.

Every root-leaf path in this tree corresponds to an execution of $\mathcal{P}$ on some input (x,y): At each internal node the process that is the next to send a bit is computing a_v, resp. b_v, to determine the value of the next bit.

Figure 2.1 shows the definition of a function $f\colon \{x_0, x_1, x_2, x_3\} \times \{y_0, y_1, y_2, y_3\} \to \{0,1\}$ which is computed in the protocol tree of Figure 2.2. The dashed path in Figure 2.2 corresponds to the input (x_3, y_3), and the sequence of sent bits is $(0,1,0)$.

	y_0	y_1	y_2	y_3
x_0	1	1	0	1
x_1	0	0	1	1
x_2	0	0	1	1
x_3	0	0	0	0

Figure 2.1: The function f computed in Figure 2.2.

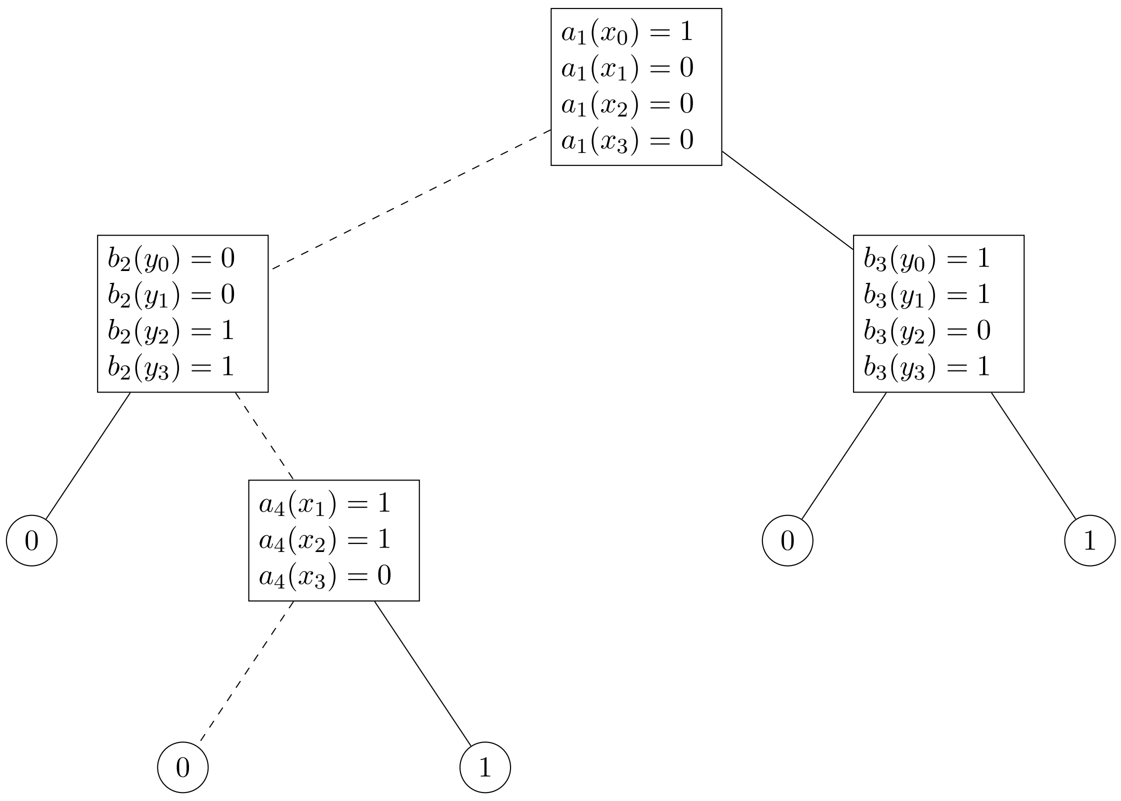


Figure 2.2: A protocol tree for function f defined in Figure 2.1. The dashed path corresponds to the input (x_3, y_3).

Definition 2.1.3. The *cost* of a protocol $\mathcal{P}$ *on input* (x, y) is the number of bits communicated by $\mathcal{P}$ on input (x, y).
The *cost of a protocol* $\mathcal{P}$ is the maximal cost of $\mathcal{P}$ over all inputs (x, y), denoted by $D(\mathcal{P})$.
The *cost of a problem* $\mathcal{P}_r(f)$ is the minimal cost of any protocol $\mathcal{P}$ that computes f. We denote this cost by $D(f)$.

The trivial protocol in Example 2.1.1 works for any function f, thus we can state an upper bound on the cost for any f:

$$D(f) \leq 2 \cdot log_2(min\{|X|, |Y|\})$$

Since the problem statement does not allow any solution which does not send a single bit, the trivial (but, for arbitrary f, not tight) lower bound is 1, hence:

$$1 \leq D(f) \leq 2 \cdot log_2(min\{|X|, |Y|\})$$

2.2 Rectangles

Since we want the communication complexity lower bound on a function f to be tight, the trivial lower bound is not of interest. To prove tight lower bounds, [Yao79] took a *combinatorial* view on protocols: A protocol $\mathcal{P}$ can be seen as a way to partition the set of possible inputs $X \times Y$ to multiple sets with the same communication pattern. E.g. in the protocol tree in Figure 2.2, the communication pattern for all the input pairs in $\{x_3\} \times \{y_2, y_3\}$ is the same, as for all those pairs the protocol follows the dashed path. Another partition in Figure 2.2 is $\{x_0\} \times \{y_0, y_1\}$, for which the protocol follows the path to the rightmost leaf. This leads to the notion of a rectangle:

Definition 2.2.1 ([KN97], Proposition 1.13)**.** A partition $R \subseteq X \times Y$ is a rectangle if and only if:

$$(x_1, y_1) \in R \text{ and } (x_2, y_2) \in R \Rightarrow (x_1, y_2) \in R$$

Note that rectangles are symmetric, even though Definition 2.2.1 appears to be asymmetric: From $(x_1, y_1) \in R$ and $(x_2, y_2) \in R$ follows $(x_1, y_2) \in R$, and from $(x_2, y_2) \in R$ and $(x_1, y_1) \in R$ follows $(x_2, y_1) \in R$. Hence, we actually have:

$$(x_1, y_1) \in R \text{ and } (x_2, y_2) \in R \Rightarrow (x_1, y_2) \in R \text{ and } (x_2, y_1) \in R$$

Furthermore, Definition 2.2.1 directly leads us to the realization that each partition created by the leaves of a tree, i.e., the set of inputs that lead to the same leaf, is a rectangle. This can be shown by a simple inductive indistinguishability argument: Suppose both inputs (x_1, y_1) and (x_2, y_2) lead to leaf ℓ, then also (x_1, y_2) and (x_2, y_1) lead to ℓ. Thus the communication pattern for them has to be exactly the same, i.e., in each node v the sent bit is the same for all of the four input pairs. We start the induction at the root node at tree-level $k = 0$: Since (x_1, y_1) and (x_2, y_2) both lead to leaf ℓ, $a_1(x_1) = a_1(x_2)$. Thus, the bit sent by p_0 is the same for each of the considered input pairs, (x_1, y_1), (x_1, y_2), (x_2, y_1) and (x_2, y_2). By the induction hypothesis the processes sent the same sequence of bits in rounds 1 to $k - 1$, i.e., (x_1, y_1), (x_2, y_2), (x_1, y_2) and (x_2, y_1) reach the same node v at level k. W.l.o.g. suppose that p_1 has to send the bit in round k. Since all the considered input pairs reached node v, p_1 cannot hence distinguish (x_1, y_1) from (x_2, y_1) and (x_1, y_2) from (x_2, y_2) and since $b_v(y_1) = b_v(y_2)$ by our assumption, the bit sent by p_1 in v is the same for (x_1, y_1), (x_2, y_2), (x_1, y_2) and (x_2, y_1).

There is a slightly stronger notion of rectangles, called *f-monochromatic rectangles*, in which the result of f is the same. Clearly, the partition created by the leaves of a protocol tree for function f consists of f-monochromatic rectangles. In fact, if we take any correct protocol $\mathcal{P}$ for a function f, $\mathcal{P}$ must actually split $X \times Y$ in several partitions which are f-monochromatic rectangles. We have to distinguish the f-monochromatic rectangles induced by f, and the ones equivalent to the partition created by a given protocol tree, however: A set of f-monochromatic rectangles induced by f is a partition of $X \times Y$

into maximal-size f-monochromatic rectangles. Clearly, a f-monochromatic rectangle created by some protocol tree is a subset of such a rectangle induced by f. Since each leaf of $\mathcal{P}$ creates a unique rectangle, the number of rectangles created by $\mathcal{P}$ is the number of leaves in $\mathcal{P}$ and is at least the number of elements in any set of f-monochromatic rectangles induced by f.

Interpreting $\mathcal{P}$ as an assignment from inputs (x, y) to communication patterns, we see that the maximum number of bits sent in $\mathcal{P}$ is equal to the height $h_{\mathcal{P}}$ of the tree. Thus, $D(\mathcal{P}) = h_{\mathcal{P}}$. Clearly, there are multiple protocols solving f, and we are interested in some protocol $\mathcal{P}_{min}$ such that $\forall \mathcal{P}\colon D(\mathcal{P}_{min}) \leq D(\mathcal{P})$. $\mathcal{P}_{min}$ satisfies $D(\mathcal{P}_{min}) = D(f)$ and has the lowest number of f-monochromatic rectangles among all protocols $\mathcal{P}$ solving f. This property can be used to find a lower bound on communication cost for f:

Corollary 2.2.1 ([KN97], Corollary 1.17)**.** If any set of f-monochromatic rectangles induced by f contains at least t rectangles, then

$$log_2\, t \leq D(f).$$

If there is a protocol $\mathcal{P}$ that partitions $X \times Y$ into t rectangles (i.e., $\mathcal{P}$ has t leaves), then $\mathcal{P}$ is *optimal* and $D(\mathcal{P}) = log_2\, t$.

2.3 Lower-Bound Techniques

Corollary 2.2.1 states a lower bound for $D(f)$ depending on the minimum number of f-monochromatic rectangles any partition of $X \times Y$ requires. This can be used in several techniques to determine a lower bound of $D(f)$ for an arbitrary function f.

2.3.1 Fooling Sets

The lower bound technique of “fooling sets” is used implicitly by [Yao79]; it first appeared in [LS81]. The idea is to find a lower bound for the number of induced rectangles t and use it to find a lower bound for $D(f)$, according to Corollary 2.2.1.

We start by specializing a f-monochromatic rectangle to a z-monochromatic rectangle R, characterized by $(x_1, y_1) \in R$ and $(x_2, y_2) \in R \Rightarrow f(x_1, y_1) = f(x_2, y_2) = z \in Z$ and $(x_1, y_2) \in R$ and $(x_2, y_1) \in R$. Thus if $f(x_1, y_2) \neq z$ or $f(x_2, y_1) \neq z$, then $(x_1, y_1) \notin R$ or $(x_2, y_2) \notin R$. A fooling set S_z is the set of pairs (x, y) leading to the same value $f(x, y) = z$ that are not in the same z-monochromatic rectangle (Definition 2.3.1). As a z-monochromatic rectangle cannot contain more than one element of S_z by maximality, $|S_z|$ is a lower bound for the number of z-monochromatic rectangles, which in turn is obviously smaller than t.

Definition 2.3.1. A set $S_z \subset X \times Y$ is a *fooling set* for f if:

- $\forall (x, y) \in S_z\colon f(x, y) = z$

- $\forall (x_1, y_1), (x_2, y_2) \in S_z \colon f(x_1, y_2) \neq z \vee f(x_2, y_1) \neq z$

Denote the number of z-monochromatic rectangles of f by N_z. If f has a fooling set S_z of size t_z, then $t_z \leq N_z$, thus $log_2\ t_z \leq log_2\ N_z \leq D(f)$. This bound can be tightened by taking into account the fooling sets for every $z \in Z$ on f, which are of course disjoint. Consequently, we obtain $\sum\limits_{z \in Z} t_z \leq \sum\limits_{z \in Z} N_z \leq t$ and thus

$$log_2 \sum_{z \in Z} t_z \leq log_2 \sum_{z \in Z} N_z \leq D(f).$$

For an example, consider the function given in Figure 2.1. We see that $S_0 = \{(x_1, y_0), (x_0, y_2), (x_3, y_3)\}$ and $S_1 = \{(x_0, y_0), (x_1, y_2)\}$. Thus $t_0 = |S_0| = 3$ and $t_1 = |S_1| = 2$, hence, $3 \leq D(f)$. Note that e.g., (x_0, y_0) and (x_0, y_1) cannot be both in S_1, since $f(x_0, y_1) = f(x_0, y_0) = 1$.

2.3.2 Rank Lower Bound

[MS82] came up with a different technique called the "Rank Lower Bound", using a matrix M_f describing the output of f ($M_f(x,y) = f(x,y)$); Figure 2.1 shows an example of such a matrix. The idea is to estimate the number of z-monochromatic rectangles by the rank $rank(M_f)$ of M_f, since it turns out that $rank(M_f) \leq N_z$.

We will prove this for an arbitrary non-constant function $f : X \times Y \to Z$. Given any protocol $\mathcal{P}$ for f, define a matrix M_{ℓ_z} for each z-valued leaf $\ell_z \in L_z$ of the protocol tree of $\mathcal{P}$, for every $z \in Z$, as follows: $M_{\ell_z}(x,y) = 1$ if the protocol reaches ℓ_z on the input (x,y), $M_{\ell_z}(x,y) = 0$ otherwise. Obviously, for each (x,y) such that $f(x,y) \neq z$, $M_{\ell_z}(x,y) = 0$ for all $\ell_z \in L_z$, while $M_{\ell_z}(x,y) = 1$ for some $\ell_z \in L_z$. It is easy to see that $M_f = \sum\limits_{z \in Z} \sum\limits_{\ell_z \in L_z} M_{\ell_z}$ and $rank(M_{\ell_z}) = 1$ for all $\ell_z \in L_z$. Thus $\sum\limits_{\ell_z \in L_z} rank(M_{\ell_z})$ equals the number of z-valued leaves of $\mathcal{P}$. By obvious rank properties we find

$$rank(M_f) \leq \sum_{z \in Z} \sum_{\ell_z \in L_z} rank(M_{\ell_z}) = \sum_{z \in Z} N_z \leq t$$

Since the protocol $\mathcal{P}$ is chosen arbitrarily, this lower bound holds for each $\mathcal{P}$. Thus

$$log_2\ rank(M_f) \leq log_2 \sum_{z \in Z} N_z \leq D(f)$$

Taking again the function given in Figure 2.1, we see that $rank(M_f) = 3$, which gives the (non-tight) lower bound $2 \leq D(f)$.

CHAPTER 3

Knowledge & Epistemic Logic

3.1 Introduction

Usually the tasks a distributed system has to solve are primarily referring to the global behavior of the system. The actions of a single process in such a system depend solely on its local information, though, and the global behavior emerges from those local actions. Thus, defining and proving the correctness of distributed systems often involves arguments about the behavior and interaction between individual processes. In such proofs it is often argued that: "Once the synchronous round r begins, all processes *know* that all the sent messages have been delivered.", for example.

To formalize such arguments, Fagin, Halpern, Moses and Vardi [FHMV95], but also others, investigated methods to reason about knowledge. One of them is *Dynamic Epistemic Logic* invented by Plaza [Pla07] and Gerbrandy [Ger97]. This logic is used to argue about knowledge and knowledge gain. At the end of this chapter, we will discuss *Action Models*, which are used to formalize the *change of knowledge* due to certain global actions.

Since we will work mainly with two examples in this chapter, we define them in the following subsections.

3.1.1 Definition: Buy or Sell? (adapted from [vDvdHK08], Example 4.1)

Example 3.1.1. Consider two stockbrokers Alice and Bob having a break in a Wall Street bar. A messenger comes in and tells Alice, that Carol will fetch her this day to tell her something important about the company "United Agents". Alice and Bob go back to work and meet again for dinner.

What can we state about the knowledge of Alice and Bob over time? Clearly, in the evening, Alice knows either that United Agents is doing well or not. But did Bob also have any change in his knowledge? Yes! Before the break he knows that United Agents does well or not, i.e., knows nothing. After the break he knows that Alice *will know* whether United Agents does well or not. At the dinner he knows that Alice *knows* whether United Agents does well or not.

3.1.2 Definition: Cheating Husbands ([MDH86])

Example 3.1.2. The queens of the matriarchal city-state of Mamajorca, on the continent of Atlantis, have a long record of opposing and actively fighting the male infidelity problem. Ever since the technologically-primitive days of queen Henrietta I, women in Mamajorca have been required to be in perfect health and pass an extensive logic and puzzle-solving exam before being allowed to take a husband. The queens of Mamajorca, however, were not required to show such competence.

It has always been common knowledge among the women of Mamajorca that their queens are truthful and that the women are obedient to the queens. It was also common knowledge that all women hear every shot fired in Mamajorca. Queen Henrietta I woke up one morning with a firm resolution to do away with the male infidelity problem in Mamajorca. She summoned all of the women heads-of-households to the town square and read the following statement:

There are (one or more) unfaithful husbands in our community. Although none of you knew before this gathering whether your own husband was unfaithful, each of you knows which of the other husbands are unfaithful. I forbid you to discuss the matter of your husband's fidelity with anyone. However, should you discover that your husband is unfaithful, you must shoot him on the midnight of the day you find out about it.

Thirty nine silent nights went by, and on the fortieth night, shots were heard.

This example is quite interesting since life and death depend on the reasoning about knowledge of other parts of the system. Lets take a look at what would have happened if there had been exactly one unfaithful husband: Clearly his wife did not know of any cheating husband. Since the queen announced truthfully that there is at least one such husband, this wife would have shot her own in the first night. All the other women would have known 1 cheating husband. Since his wife shot him in the first night, all the other know that she did not knew about any other husband, or she would not have known that her husband was cheating.

In general: If there had been n unfaithful husbands, they would all have been shot on the midnight of the n^{th} day ([MDH86], Theorem 1).
We already sketched the proof for the induction basis on the number of husbands $n \geq 1$ above. Now we sketch the proof of the induction step:
If there are n unfaithful husbands, then there are some wives knowing of n unfaithful husbands (those who are not cheated on) and some wives knowing of $n-1$ unfaithful husbands. If there would have been $n-1$ unfaithful husbands they would have been shot

on the midnight of day $n-1$, by the induction hypothesis. Since no shots fall in this night, all the women knowing $n-1$ unfaithful husbands know that their own husbands are unfaithful and shoot them.

We will get a more formal look on this example later on.

3.2 Epistemic Logic

In this section we will introduce basic epistemic logic and some different notions of knowledge and group knowledge. Defining and arguing about the notions of *Basic Epistemic Logic* (Section 3.2.1), *Everybody knows* and *Common Knowledge* (Section 3.2.2) and *Dynamic Epistemic Logic* (Section 3.3), we follow the textbook [vDvdHK08].

3.2.1 Basic Epistemic Logic

Hintikka [Hin62] formalized modern epistemic logic and gave it a possible world semantics. Epistemic logic is based on modal logic and formalized using Kripke structures. It is investigated in very different areas of research, e.g. artificial intelligence [Moo80], game theory [Aum76] and philosophy [HH89], the area it originated from. [Aum76] also formalized common knowledge the first time, the notion of which was first discussed by [Lew69]. Later on also computer scientists started to use epistemic logic as a tool for argumentation, [FHMV95] gives a survey of many papers investigating this topic.

We start by defining the language and semantics of the basic epistemic logic. To do so we need the atomic propositions $p, q, \ldots$, which describe some state, e.g., p may stand for "United Agents is doing well". To refer to processes (or in general members of the system) we use agent-symbols $a, b, \ldots$.

Definition 3.2.1 (Basic epistemic language ([vDvdHK08], Definition 2.4))**.** Let P be a set of atomic propositions, and A a set of agent-symbols. The language $\mathcal{L}_{\mathcal{K}}$ is generated by:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi$$

where $p \in P$ and $a \in A$.

As usual, we use also standard abbreviations like $(\varphi \vee \psi) = \neg(\neg\varphi \wedge \neg\psi)$, $(\varphi \to \psi) = (\neg\varphi \vee \psi)$ and $(\varphi \leftrightarrow \psi) = (\varphi \to \psi) \wedge (\psi \to \varphi)$. For each process a, we define the knowledge operator K_a, such that $K_a\varphi$ can be interpreted as "process a knows fact φ". Using this operator we also can describe nested knowledge like $K_aK_b\varphi$: "a knows that b knows φ".

In the following we will define the semantics of epistemic formulas. Such formulas are interpreted on states in Kripke models (M, s).

Definition 3.2.2 (Kripke model, see [vDvdHK08], Definition 2.6)**.** A *Kripke model* is a tuple $\langle S, R, V\rangle$ on a set of processes A, where:

- $S \neq \emptyset$ is a set of states.
- R is a set of accessibility relations: $R = \{R_a \mid a \in A\}$, with $R_a \subseteq S \times S$. A state $t \in S$ is accessible for process $a \in A$ from state $s \in S$, iff sR_at.
- $V : P \to 2^S$ is a valuation function for each proposition p. For any proposition p, $V(p) \subseteq S$ is exactly the set of states in which p is true.

Usually in our context, the accessibility relation R_a for some process a is interpreted as an indistinguishability relation: "process a cannot distinguish between states s and t", thus denoted by $\sim_a$. In general, however R_a need not be symmetric. In such a case, process a in state s would not know whether it is in s or in t, but process a in t would know that it is in t and not in s. An indistinguishability relation has all the properties of an equivalence relation, thus each node contains a self-loop for each process and we usually omit the arrows on the relation between the nodes.

For an example look at the Kripke model M given in Figure 3.1, which is $M = \langle S, \sim, V\rangle$, with:

- $S = \{s_0, s_1, s_2\}$
- $\sim_a = \{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_0, s_1), (s_1, s_0)\}$
- $\sim_b = \{(s_0, s_0), (s_1, s_1), (s_2, s_2), (s_1, s_2), (s_2, s_1)\}$
- $V(p) = \{s_0, s_1\}$

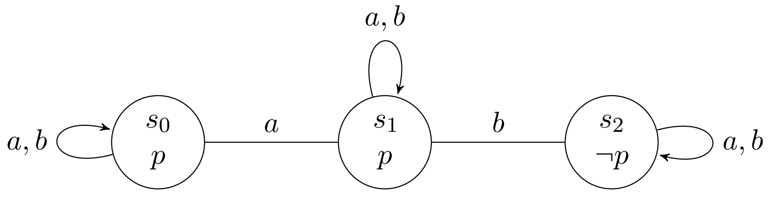


Figure 3.1: First example of a Kripke model. Since $\sim_a$ and $\sim_b$ are indistinguishability relations, the model contains self loops.

The following Definition 3.2.3 gives the semantics of formulas in $\mathcal{L}_\mathcal{K}$ in a Kripke model.

Definition 3.2.3 ([vDvdHK08], Definition 2.7)**.** Given a Kripke model $M = \langle S, \sim, V\rangle$ and a state $s \in S$, a formula φ is true in (M, s), denoted by $M, s \models \varphi$, iff:

$M, s \models p$	iff	$s \in V(p)$
$M, s \models (\varphi \wedge \psi)$	iff	$M, s \models \varphi$ and $M, s \models \psi$
$M, s \models \neg\varphi$	iff	not $M, s \models \varphi$, denoted $M, s \not\models \varphi$
$M, s \models K_a\varphi$	iff	for all $t \in S$ such that $s \sim_a t$: $M, t \models \varphi$

The states in a Kripke model can be seen as possible worlds. For an example look at Figure 3.2, the states represent the weather in Vienna and Dublin via the atomic propositions p, the sun is shining in Vienna, and q, the sun is shining in Dublin. There are four possible worlds in the corresponding Kripke model: s_0 it is sunny both in Vienna and Dublin, s_1 it is sunny only in Vienna, s_2 it is sunny only in Dublin, s_3 it is not sunny both in Vienna and Dublin. Think of the persons Anne and Bob, both in Vienna. Both of them observe whether it is sunny in Vienna or not. Thanks to his smart-phone, Bob has access to the internet and looks up the actual weather in Dublin. As Anne's phone is broken, she does not have such possibility. Thus, if it is sunny in Vienna, Anne considers the worlds s_0 and s_1 possible, since she cannot observe the weather in Dublin. In both of the worlds it is sunny in Vienna, thus she *knows* that it is sunny in Vienna and $K_{Anne}p$ holds (see Definition 3.2.3).
Since Bob observes the weather in both of the cities, he is able to distinguish any of the states from any other state, thus only considers the actual state possible. Thus, if it is sunny in both Vienna and Dublin, this holds in all of the worlds he considers possible, thus $K_{Bob}(p \wedge q)$.

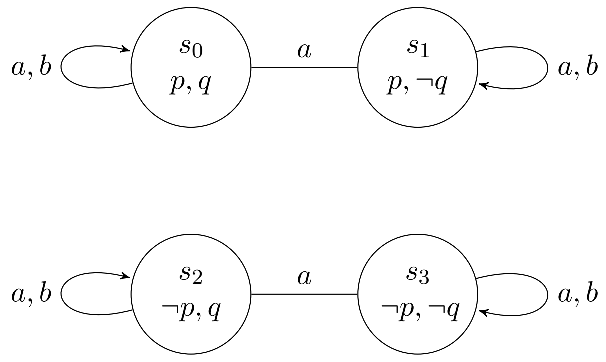


Figure 3.2: Example for a Kripke model. p denotes the fact that it is sunny in Vienna, while q denotes sunny in Dublin. Agent a (Anne) only knows whether or not it is sunny in Vienna, while agent b (Bob) knows the weather in both of the cities.

Here are some examples for the interpretation of epistemic formulas in the Kripke model of Figure 3.1:

- $M, s_0 \models p$, $M, s_1 \models p$ and $M, s_2 \not\models p$
  These we get directly from $V(p)$.

- $M, s_0 \models K_a p$ and $M, s_0 \models K_b p$
  All states t indistinguishable from s_0 for a, namely $t \in \{s_0, s_1\}$, satisfy $M, t \models p$.
  All states t indistinguishable from s_0 for b, namely $t \in \{s_0\}$, satisfy $M, t \models p$.

- $M, s_1 \models K_a p$ and $M, s_1 \not\models K_b p$
  All states $t \in \{s_0, s_1\}$ indistinguishable from s_1 for a satisfy $M, t \models p$.
  There is a state $t = s_2$ indistinguishable from s_1 for b, such that $M, t \not\models p$.

- $M, s_0 \models K_b K_a p$
  All states $t \in \{s_0\}$ indistinguishable from s_0 for b satisfy $M, t \models K_a p$ as established above.

- $M, s_0 \not\models K_a K_b p$
  There is a state $t = s_1$ indistinguishable from s_0 for a, such that $M, t \not\models K_b p$, as established above.

3.2.2 "Everybody knows" and Common Knowledge

We now introduce the notions related to "everybody knows φ". Intuitively "everybody in group B knows fact φ" — denoted by $E_B\varphi$ for some non-empty set of processes B — has the meaning that for each process $a \in B$ it holds that $K_a\varphi$. Therefore we introduce this notion as syntactical equivalence:

$$E_B\varphi = \bigwedge_{a \in B} K_a\varphi$$

To introduce nested "everybody knows" we use:

$$E_B^0\varphi = E_B\varphi$$
$$E_B^k\varphi = E_B E_B^{k-1}\varphi$$

We introduce the notion of *Common Knowledge*, as the limit of nested "everybody knows" via the infinite conjunction that "everybody knows that everybody knows that ... everybody knows that φ":

$$C_B\varphi = \bigwedge_{n=0}^{\infty} E_B^n\varphi$$

To demonstrate the difference between everybody knows and common knowledge, we use an example:

Example 3.2.1. Consider three people a, b, c sitting in a train. a and b are talking to each other while c listens to some music with her headphones. There is an announcement via the speakers, stating that the train is half an hour late (denote that fact by φ). a and b stop their conversation and listen to the announcement. Since they both know that the other heard the announcement as well it holds that "both know that both know that ... both know φ" ad infinitum. Thus, $C_{\{a,b\}}\varphi$.

Since the volume of c's earphones is just low enough that c understands all the announcements, she also knows φ. Thus $E_{\{a,b,c\}}\varphi$ holds. As c heard a and b complaining about the delay, $K_c E_{\{a,b,c\}}\varphi$ and even $K_c C_{\{a,b\}}\varphi$ holds. But since a and b did not recognize that c heard the announcement, $E^2_{\{a,b,c,\}}\varphi$ and hence $C_{\{a,b,c\}}\varphi$ do not hold.

To define common knowledge we need to extend our language $\mathcal{L}_\mathcal{K}$ by $C_B\varphi$ for any non-empty set of processes B.

Definition 3.2.4 (similar to [vDvdHK08], Definition 2.26)**.** Let P be a set of atomic propositions, A a set of process-symbols, B a non-empty subset of A and a some process in A. The language $\mathcal{L}_{\mathcal{KC}}$ is generated by:

$$\varphi ::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid C_B\varphi$$

To define the semantics of common knowledge, we first have to introduce the *reflexive transitive closure* of a relation R (Definition 3.2.5) before we can extend the interpretation given in Definition 3.2.3 towards Definition 3.2.6.

Definition 3.2.5. The *reflexive transitive closure* of a relation R is the smallest relation R^* such that:

- $R \subseteq R^*$,
- for all x, y, and z: $xR^*y \wedge yR^*z \Rightarrow xR^*z$ (transitivity)
- for all x, xR^*x (reflexivity)

Note that if y is reachable from x using only pairs of elements related by R, then xR^*y.

Definition 3.2.6. Given a Kripke model $M = \langle S, \sim, V\rangle$ and a state $s \in S$, an epistemic formula of $\mathcal{L}_{\mathcal{KC}}$ φ is true in (M, s), iff:

$M, s \models p$	iff	$s \in V(p)$
$M, s \models (\varphi \wedge \psi)$	iff	$M, s \models \varphi$ and $M, s \models \psi$
$M, s \models \neg\varphi$	iff	$M, s \not\models \varphi$
$M, s \models K_a\varphi$	iff	for all $t \in S$ such that $s \sim_a t$: $M, t \models \varphi$
$M, s \models E_B\varphi$	iff	for all $t \in S$ such that $s \sim_{E_B} t$: $M, t \models \varphi$
$M, s \models C_B\varphi$	iff	for all $t \in S$ such that $s \sim^*_{E_B} t$: $M, t \models \varphi$,

with $\sim_{E_B} = \bigcup\limits_{b\in B} \sim_b$.

Since $\sim_{E_B} = \bigcup\limits_{b\in B} \sim_b$, everybody in a group B of processes knows a fact φ in state s, if φ is true in each state t that can be reached from s using a single step $\sim_b$. I.e. there is no state t, in which φ does not hold, which cannot be distinguished from s by some process $b \in B$.
As $x \sim^*_{E_B} y$ only if y can be reached from x using only steps from $\sim_{E_B}$, a group B commonly knows a fact φ in state s, if φ is true in each state t that can be reached from s by using only indistinguishability relations $\sim_b$ corresponding to some process $b \in B$. In the graphical representation of a Kripke structure this means that a group B commonly knows φ in state s, only if there is no path consisting solely of edges corresponding to any $b \in B$ to any state t in which φ does not hold.

3.2.3 Buy or Sell?

We will now use epistemic logic to describe the knowledge of Alice and Bob, denoted by a respectively b in Example 3.1.1. At the start of the coffee break we clearly have the Kripke model M_{BoS} depicted in Figure 3.3. The fact that United Agents is doing well is denoted by proposition p.

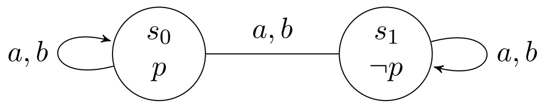


Figure 3.3: Example Buy or Sell: Kripke model M_{BoS} at the start of the coffee break. Proposition p denotes that United Agents is doing well.

Since neither Alice nor Bob knows anything about United Agents, for any state s it holds that $M_{BoS}, s \models K_a(p \vee \neg p) \wedge K_b(p \vee \neg p)$. It even holds that $M_{BoS}, s \models C_{\{a,b\}}(p \vee \neg p)$, it is common knowledge that both consider it possible that p or $\neg p$.

The situation in the evening is of greater interest. Carol already talked to Alice and Bob knows that she did, thus the corresponding Kripke model M'_{BoS} is as depicted in Figure 3.4.

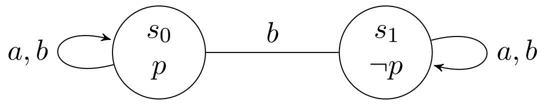


Figure 3.4: Example Buy or Sell: Kripke model M'_{BoS} in the evening. Proposition p denotes that United Agents is doing well.

If Carol told Alice that United Agents is doing well, the processes are in state s_0. From Figure 3.4 we see that $M'_{BoS}, s_0 \models K_a p$. We also see that still $M'_{BoS}, s_0 \models K_b(p \vee \neg p)$, but also $M'_{BoS}, s_0 \models K_b((p \rightarrow K_a p) \wedge (\neg p \rightarrow K_a \neg p))$, i.e., that Bob knows that Alice knows whether United Agents is doing well or not.

3.2.4 Cheating Husbands

We now look at the Kripke models involved in the Cheating Husbands Problem in Example 3.1.2. In order to simplify the figures, we will only consider three wives (a, b and c) and will omit the self-loops in the Kripke models. Each state in the model in Figure 3.5 is labeled with (xyz), $x, y, z \in \{0, 1\}$, $x = 1(0)$ as "husband of a is unfaithful (faithful)". $y = 1(0)$ depicts "husband of b is unfaithful (faithful)" and $z = 1(0)$ depicts "husband of c is unfaithful (faithful)". Figure 3.5 depicts the initial Kripke model M_{CH}.

This is another excellent example to show the difference between $E_{\{a,b,c\}}\varphi$ and $C_{\{a,b,c\}}\varphi$. Denote by φ the fact that at least one husband is unfaithful and consider any state in which at least two husbands are unfaithful. As it happens,
$\forall s \in \{(011), (110), (101), (111)\}\colon M_{CH}, s \models E_{\{a,b,c\}}\varphi$ is valid, since in all neighboring

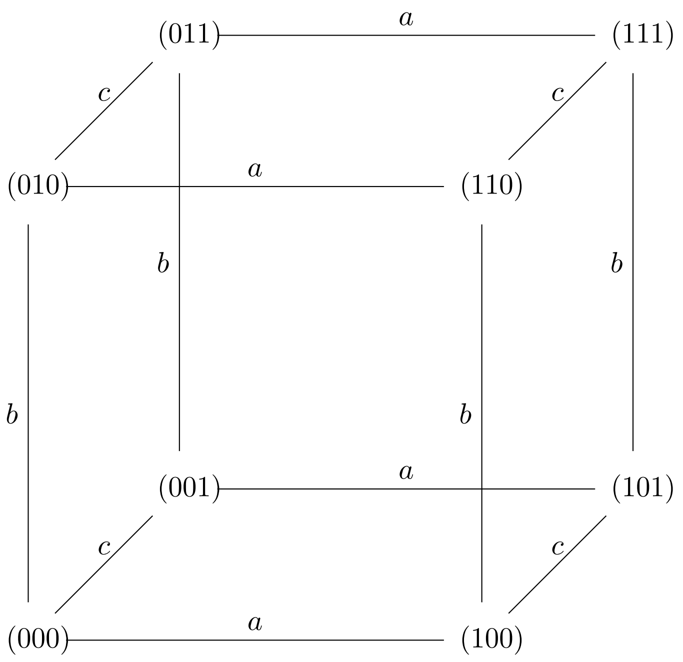


Figure 3.5: Example Cheating Husbands: Initial Kripke model M_{CH} for three wives a, b, c.

states φ holds. On the other hand $\forall s \in S\colon M_{CH}, s \not\models C_{\{a,b,c\}}\varphi$, since from each state s there is a path to (000) in which φ does not hold.

Next we will have a look at the situation after queen Henrietta publicly announced the situation on town square, depicted in Figure 3.6 as model $\mathtt{M}'_{CH} = (S'_{CH}, \sim'_{CH}, V'_{CH})$.

As the queens announcement is publicly made on the town square, all of the women hear it and know that all of the other women also hear it. Thus each woman can distinguish state 000 from the other states and knows that all other women also can. Now, since there is no path from any other state s to state (000), $\forall s \in S'\colon M'_{CH}, s \models C_{\{a,b,c\}}\varphi$.

3.3 Dynamic Epistemic Logic & Action Models

Up to now we considered *Epistemic Logic* only, which is concerned with the knowledge associated with a certain situation. We will now take a further step towards *Dynamic Epistemic Logic*, which is concerned with the dynamic change of knowledge caused by certain events (actions).

Consider the *Buy or Sell?* example. Previously we argued about the situation at the coffee break and about the situation in the evening, but we did not formalize the change of knowledge during the day, i.e., how the situation changed.

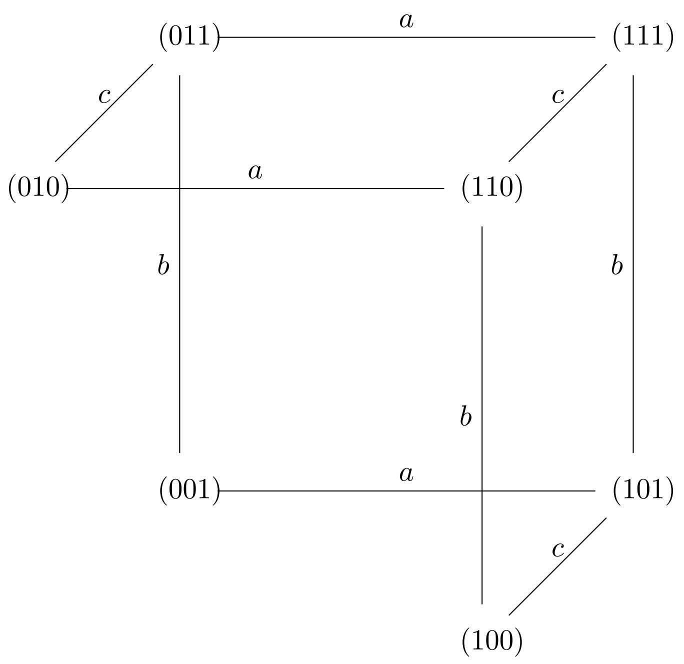


Figure 3.6: Example Cheating Husbands: Kripke model M'_{CH} after the public announcement of queen Henrietta.

There are various formalizations regarding this change of knowledge. [Pla89] and [GG97] came up with the *Public Announcement Logic*, which deals with public announcements, albeit without the notion of common knowledge. The axiomatisation of public announcements with common knowledge, as in the cheating husband problem (every member realizes that the announcement is public and that everybody else also realizes this fact), has been established in [BMS98]. A further step has been taken by Baltag [Bal99], who introduced *Epistemic Actions Logic.* An example of such an action can be found in the Buy or Sell example, where Carol tells Alice that United Agents is doing well at some time during the day. In Epistemic Actions Logic, this is formalized as *Learning formulas* ([vD99], [vD02]), like $L_{a,b}(!L_a?p \cup L_a?\neg p)$, which means that both Alice and Bob (denoted by $L_{a,b}$) learn that Alice will learn either p or not p (denoted by $L_a?p \cup L_a?\neg p$), and Alice actually learns p (denoted by $!L_a?p$). Since these formulas are quite tedious and are growing very fast [BM04] introduced *Action Models*, which we will discuss in this section.

3.3.1 Action Models

We will first define action models, the syntax and the semantics of action model logic (as in [vDvdHK08], Section 6.2), and then apply it to our examples.

We start with some remarks on the notation. In the following we will refer to the Kripke structures introduced in Section 3.2.1 as *epistemic structures* or *epistemic models.* We will see in the next definitions that the syntax of *epistemic models* and *action models* is quite similar, since both are defined as Kripke structures. Thus we use the font $M = \langle S, \sim, V\rangle$ to denote epistemic models, and $\mathtt{M} = \langle \mathtt{S}, \sim, \mathtt{pre}\rangle$ when talking about action models.

An action model consists of all possible *actions* (events) any process can observe in a given interval of time, e.g. a day or a synchronous round. Like an epistemic model, it is modeled as a Kripke structure, which contains all those possible actions and the relation $\sim$ between those actions, such that $\mathtt{s} \sim_a \mathtt{t}$ means that process a cannot distinguish if action $\mathtt{s}$ or action $\mathtt{t}$ actually happened. An epistemic model is *updated* by *applying* an action model, in such a way that a state s' of the updated model is built from a state s of the original epistemic model and an action $\mathtt{s}$, denoted by $s' = (s, \mathtt{s})$. Note that $s' = (s, \mathtt{s})$ only exists if $\mathtt{s}$ is *applicable* on s; we also say that s fulfills the *preconditions* of $\mathtt{s}$, formally $M, s \models \mathtt{pre}(\mathtt{s})$. The fact that, when updating an epistemic model with an action model, not every pair $(s, \mathtt{s})$ needs to be contained in the updated epistemic model makes sense, since in a world in which it is sunny in Vienna, the (truthful) action "a tells b that it is not sunny in Vienna" is not applicable.

Since we want to be able to reason about the change of certain epistemic states when applying certain actions, we need the notion of a *pointed action model* $(\mathtt{M}, \mathtt{s})$. Updating a certain epistemic state (M, s) with a pointed action model $(\mathtt{M}, \mathtt{s})$, such that $\mathtt{M}$ is the action model of all the possible actions and $\mathtt{s}$ is the action which is actually applied, the result is another epistemic state (M', s'), with M' the updated action model and $s' = (s, \mathtt{s})$. Note that S' may be the empty set. We will denote the application of $(\mathtt{M}, \mathtt{s})$ to (M, s) resulting in (M', s') by $(M, s)[\![\mathtt{M}, \mathtt{s}]\!](M', s')$. In our graphical representations the actual epistemic state respectively the actual applied action will be underlined. Please note that we will sloppily use the word "action" for pointed action models $(\mathtt{M}, \mathtt{s})$ as well as for the actual action $\mathtt{s}$, which are part of an action model. In cases where it is not clear which "action" we mean, we will state this explicitly.

Definition 3.3.1 (Action Model ([vDvdHK08], Definition 6.2))**.** For given processes A and atomic propositions P and any logical language $\mathcal{L}$, the action model $\mathtt{M}$ is a structure $\langle \mathtt{S}, \sim, \mathtt{pre}\rangle$ such that $\mathtt{S}$ is a set of *actions*, $\sim_a$ is an equivalence relation on $\mathtt{S}$ for each $a \in A$, and $\mathtt{pre} : \mathtt{S} \to \mathcal{L}$ a preconditions function that assigns a *precondition* $\mathtt{pre}(\mathtt{s}) \in \mathcal{L}$ to each $\mathtt{s} \in \mathtt{S}$. A *pointed action model* is a structure $(\mathtt{M}, \mathtt{s})$, with $\mathtt{s} \in \mathtt{S}$.

The languages of action models, $\mathcal{L}_{\mathcal{KC}\otimes}$, formulas on action models, $\mathcal{L}^{stat}_{\mathcal{KC}\otimes}$, and actions on action models, $\mathcal{L}^{act}_{\mathcal{KC}\otimes}$, are defined in Definition 3.3.2 in an inductive manner.

Definition 3.3.2 (Syntax of action model logic ([vDvdHK08], Definition 6.3))**.** Given processes A and atoms P, the language $\mathcal{L}_{\mathcal{KC}\otimes}(A, P)$ is the union of *formulas* $\varphi \in \mathcal{L}^{stat}_{\mathcal{KC}\otimes}(A, P)$ and *pointed action models* $\alpha \in \mathcal{L}^{act}_{\mathcal{KC}\otimes}(A, P)$ defined by:

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a\varphi \mid C_B\varphi \mid [\alpha]\varphi \\ \alpha &::= (\mathtt{M}, \mathtt{s}) \mid (\alpha \cup \alpha) \end{aligned}$$

with $p \in P$, $a \in A$, $B \subseteq A$, and $(\mathtt{M}, \mathtt{s})$ a pointed action model with finite domain $\mathtt{S}$ such that for all $\mathtt{t} \in \mathtt{S}$ the precondition $\mathtt{pre}(\mathtt{t})$ is a $\mathcal{L}^{stat}_{\mathcal{KC}\otimes}(A, P)$ formula that has already been constructed in a previous stage of the inductively defined hierarchy.

In Definition 3.3.2 we see two new constructs: $[\alpha]\varphi$ stands for "after the application of action α, it holds that φ". Note that we will also use "update with action α" instead of "application of α" in the following. $(\alpha \cup \alpha')$ denotes a *non-deterministic choice* between α and α'.

A valid action model is for example $\mathtt{pub}(\varphi)$, the public announcement of formula φ. $\mathtt{pub}(\varphi) = \langle \{\mathtt{pub}\}, \sim, \mathtt{pre} \rangle$ such for all processes a $\mathtt{pub} \sim_a \mathtt{pub}$ and $\mathtt{pre}(\mathtt{pub}) = \varphi$. As we will see later, this action model states that:

- No process a can distinguish the actions $\mathtt{pub}$ and $\mathtt{pub}$ — which is clear from this context as $\sim_a$ is an indistinguishability relation which contains a self-loop for all the actions.
- The action $\mathtt{pub}$ can only be applied to a state s of some epistemic model M if $M, s \models \varphi$, as will be seen in the Definition 3.3.4 of the semantics of action model logic.

The *composition* of two action models is also defined syntactically in Definition 3.3.3. Applying the actions $(\mathtt{M}, s)$ and $(\mathtt{M}', \mathtt{s}')$ to an epistemic state (M, s), one can either apply them one after the other: $(M, s)[\![\mathtt{M}, \mathtt{s}]\!](M', s')[\![\mathtt{M}', \mathtt{s}']\!](M'', s'')$, or determine their composition $\mathtt{M}'' = (\mathtt{M}; \mathtt{M}')$, with actual action $\mathtt{s}'' = (\mathtt{s}, \mathtt{s}')$, first and apply this composition on (M, s): $(M, s)[\![\mathtt{M}'', \mathtt{s}'']\!](M'', s'')$.

Definition 3.3.3 (Composition of action models, [vDvdHK08], Definition 6.7)**.** Let $\mathtt{M} = \langle \mathtt{S}, \sim, \mathtt{pre} \rangle$ and $\mathtt{M}' = \langle \mathtt{S}', \sim', \mathtt{pre}' \rangle$ be two action models in $\mathcal{L}_{\mathcal{KC}\otimes}$. Then their *composition* $(\mathtt{M}; \mathtt{M}')$ is the action model $\mathtt{M}'' = \langle \mathtt{S}'', \sim'', \mathtt{pre}'' \rangle$, such that:

$$\begin{array}{rcl} \mathtt{S}'' & = & \mathtt{S} \times \mathtt{S}' \\ (\mathtt{s}, \mathtt{s}') \sim''_a (\mathtt{t}, \mathtt{t}') & \text{iff} & \mathtt{s} \sim_a \mathtt{t} \text{ and } \mathtt{s}' \sim'_a \mathtt{t}' \\ \mathtt{pre}''((\mathtt{s}, \mathtt{s}')) & = & \langle \mathtt{M}, \mathtt{s} \rangle \mathtt{pre}'(\mathtt{s}') \end{array}$$

with $\langle \mathtt{M}, \mathtt{s} \rangle \mathtt{pre}'(\mathtt{s}')$ denoting an abbreviation for $\neg[\mathtt{M}, \mathtt{s}]\neg\mathtt{pre}'(\mathtt{s}')$.

We will now give the definition of the semantics of action model logic. This definition starts with the semantics of formulas on $\mathcal{L}^{stat}_{\mathcal{KC}\otimes}$, similar to the semantics of epistemic logic. The definition will conclude with the semantics of *applying* an action to an epistemic model — the heart of this section.

Definition 3.3.4 (Semantics of action model logic ([vDvdHK08], Definition 6.8))**.** Let $M = \langle S, \sim, V \rangle$ be an epistemic model with (M, s), $s \in S$, an epistemic state of this model, $\mathtt{M} = \langle \mathtt{S}, \sim, \mathtt{pre} \rangle$ an action model, and $\varphi \in \mathcal{L}^{stat}_{\mathcal{KC}\otimes}$ and $\alpha \in \mathcal{L}^{act}_{\mathcal{KC}\otimes}$. Furthermore let A be a set of processes and P the set of atoms, while $a \in A$, $B \subseteq A$ and $p \in P$.

$$\begin{array}{lll}
M, s \models p & \text{iff} & s \in V(p) \\
M, s \models (\varphi \wedge \psi) & \text{iff} & M, s \models \varphi \text{ and } M, s \models \psi \\
M, s \models \neg\varphi & \text{iff} & M, s \not\models \varphi \\
M, s \models K_a\varphi & \text{iff} & \text{for all } t \in S \text{ such that } s \sim_a t\colon\ M, t \models \varphi \\
M, s \models E_B\varphi & \text{iff} & \text{for all } t \in S \text{ such that } s \sim_{E_B} t\colon\ M, t \models \varphi \\
M, s \models C_B\varphi & \text{iff} & \text{for all } t \in S \text{ such that } s \sim^*_{E_B} t\colon\ M, t \models \varphi \\
M, s \models [\alpha]\varphi & \text{iff} & \text{for all } M', s' \text{ such that } (M, s)[\![\alpha]\!](M', s')\colon\ M', s' \models \varphi \\
(M, s)[\![\mathtt{M}, \mathtt{s}]\!](M', s') & \text{iff} & M, s \models \mathtt{pre(s)} \text{ and } (M', s') = (M \otimes \mathtt{M}, (s, \mathtt{s})) \\
[\![\alpha \cup \alpha']\!] & = & [\![\alpha]\!] \cup [\![\alpha']\!]
\end{array}$$

We define $M' = (M \otimes \mathtt{M})$ as $M' = \langle S', \sim', V' \rangle$ with:

$$\begin{array}{lll}
S' & = & \{(s, \mathtt{s}) \mid s \in S, \mathtt{s} \in \mathtt{S}, \text{and } M, s \models \mathtt{pre(s)}\} \\
(s, \mathtt{s}) \sim'_a (t, \mathtt{t}) & \text{iff} & s \sim_a t \text{ and } \mathtt{s} \sim_a \mathtt{t} \\
(s, \mathtt{s}) \in V'(p) & \text{iff} & s \in V(p)
\end{array}$$

Please note that, in $s \sim_a t$, $\sim_a$ corresponds to the epistemic model M, while in $\mathtt{s} \sim_a \mathtt{t}$, $\sim_a$ corresponds to the action model $\mathtt{M}$.

3.3.2 Buy or Sell?

We will extend this example by some additional scenarios, which differ in the actions Carol may do:

1. *talk*: In the original Example 3.1.1, Carol will talk to Alice at some time that day.
2. *maytalk*: In this variant of Example 3.1.1, the messenger tells them that Carol wants to talk to Alice at some time that day, but it may be possible that Carol does not find time to do so.
3. *bothmaytalk*: In this variant of Example 3.1.1, the messenger tells them that Carol wants to talk to any or both of them at some time that day, but it may be possible that Carol does not find time to do so. In any case she will first try to talk to Alice.

For all of our variants, the initial epistemic model M_{BoS} depicted in Figure 3.3 applies. To simplify the figures, we redefine M_{BoS}, discarding the state labels s_0 and s_1 and denoting the states with p and $\neg p$ corresponding to the validity of p in a state. We will also omit self-loops in the figures, and will denote them by *loops* in written definitions.

$M_{BoS} = \langle S, \sim, V \rangle$, with:

$$S = \{p, \neg p\}$$

$$\sim_a = \sim_b = \{(p, \neg p), (\neg p, p)\} \cup \textit{loops}$$

$$V(p) = \{p\}$$

In M_{BoS} neither Alice nor Bob are able to distinguish whether United Agents is doing well or not.

talk We will now establish an action model $\mathtt{M}_{talk} = \langle \mathtt{S}, \sim, \mathtt{pre} \rangle$ for the first scenario. There are two possible actions: "Carol tells Alice that United Agents is doing well", denoted by w, and "Carol tells Alice that United Agents is not doing well", denoted by n.

$$\mathtt{S} = \{\mathtt{w}, \mathtt{n}\}$$

Since Bob does not know what Carol tells Alice he cannot distinguish w from n, while Alice perfectly can.

$$\sim_a = \{(\mathtt{w}, \mathtt{w}), (\mathtt{n}, \mathtt{n})\} = \mathit{loops}$$

$$\sim_b = \{(\mathtt{w}, \mathtt{w}), (\mathtt{n}, \mathtt{n}), (\mathtt{w}, \mathtt{n}), (\mathtt{n}, \mathtt{w})\} = \{(\mathtt{w}, \mathtt{n}), (\mathtt{n}, \mathtt{w})\} \cup \mathit{loops}$$

Since Carol only tells the truth, Carol can only tell Alice that United Agents is doing well, if it actually does well, and vice versa. Thus:

$$\mathtt{pre}(\mathtt{w}) = p$$

$$\mathtt{pre}(\mathtt{n}) = \neg p$$

Figure 3.7 shows $(M_{BoS} \otimes \mathtt{M}_{talk})$ which has the same epistemic structure as M'_{BoS} depicted in Figure 3.4. In particular, Figure 3.7 incorporates the information that, if United Agents is actually doing well and Carol actually tells Alice that it is, then in M_{talk} the actual state is $(p, \mathtt{w})$: $(M_{talk}, (p, \mathtt{w}))$ is the result of applying $(\mathtt{M}_{talk}, \mathtt{w})$ on (M_{BoS}, p).
In all of the figures depicting the application of an action model, the original epistemic model is at the top left, the applied action model on the bottom left, and the resulting epistemic model to the right. Note that, in Figure 3.12 and Figure 3.13, the resulting epistemic model is at the bottom of the figure, while the applied action model is on the top right.

$$\begin{tikzcd}
\underline{p} \arrow[rr, dash, "{a,b}"] & & \neg p & & & \\
& \otimes & & \Longrightarrow & \underline{(p,\mathtt{w})} \arrow[r, dash, "b"] & (\neg p, \mathtt{n}) \\
\underline{\mathtt{w}} \arrow[rr, dash, "b"] & & \mathtt{n} & & &
\end{tikzcd}$$

Figure 3.7: Example Buy or Sell: Application of the Action model for scenario *talk*. w denotes the action that Carol tells Alice that United Agents is doing well and n denotes the action that Carol tells Alice that United Agents is not. The actual initial (resp. updated) epistemic state, as well as the actually performed action, is underlined. Please note that self-loops are omitted.

We now evaluate the knowledge of Alice and Bob about the fact p. We want to know whether:

$$M_{BoS}, p \overset{?}{\models} [\mathtt{M}_{talk}, \mathtt{w}]K_a p$$

By the semantics of action model logic we have to consider all epistemic models (M', s'), such that $(M_{BoS}, p)[\![\mathtt{M}_{talk}, \mathtt{w}]\!](M', s')$. Luckily $(M_{talk}, (p, \mathtt{w}))$ is the only such epistemic model. Clearly $M_{talk}, (p, \mathtt{w}) \models K_a p$, thus:

$$M_{BoS}, p \models [\mathtt{M}_{talk}, \mathtt{w}]K_a p$$

Analogously we see that:

$$M_{BoS}, p \not\models [\mathtt{M}_{talk}, \mathtt{w}]K_b p$$

since $M_{talk}, (p, \mathtt{w}) \not\models K_b p$ as $(p, \mathtt{w})$ and $(\neg p, \mathtt{n})$ are indistinguishable for b, and p does not hold in $(\neg p, \mathtt{n})$.

maytalk In this alternative scenario there are three possible actions: the two actions from the previous scenario and an additional one, "Carol does not tell Alice anything", denoted by $\neg\mathtt{t}$ (for does not tell). As in the previous scenario Figure 3.8 depicts the resulting action model $(M_{BoS} \otimes \mathtt{M}_{mt})$ and its application to the initial situation, for the actual action $\neg\mathtt{t}$ and United Agents doing well.

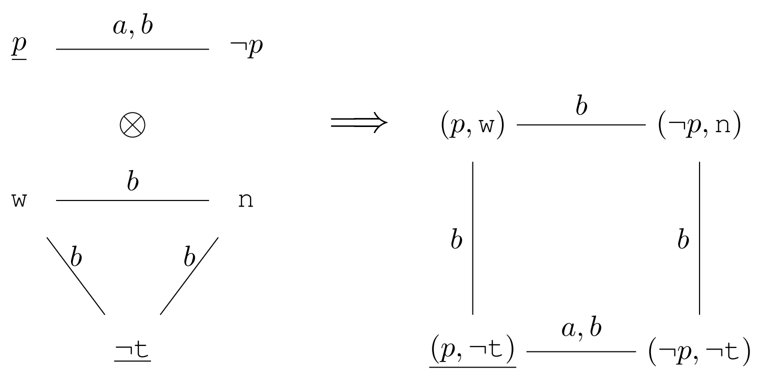


Figure 3.8: Example Buy or Sell: Action model $\mathtt{M}_{mt}$ for alternative scenario *maytalk*. $\mathtt{w}$ denotes the action that Carol tells Alice that United Agents is doing well, $\mathtt{n}$ denotes the action that Carol tells Alice that United Agents is not, and $\neg\mathtt{t}$ that Carol does not talk to Alice. Please note that self-loops are omitted. The actual action is $\neg\mathtt{t}$ and United Agents is doing well.

Since the action that "Carol does not tell Alice anything" does not depend on whether United Agents is doing well or not, $\mathtt{pre}(\neg\mathtt{t}) = \top$. Thus $\neg\mathtt{t}$ can be applied to any state of M_{BoS}. Intuitively, neither Alice nor Bob do know that United Agents is doing well if Carol does not tell Alice, as Figure 3.8 reveals:

$$M_{BoS}, p \not\models [\mathtt{M}_{mt}, \neg\mathtt{t}]K_a p$$

$$M_{BoS}, p \not\models [\mathtt{M}_{mt}, \neg\mathtt{t}]K_b p$$

Since Alice and Bob both know that Bob does not know p, it holds that:

$$M_{BoS}, p \models [\mathtt{M}_{mt}, \neg\mathtt{t}]E_{\{a,b\}}\neg K_b p$$

It even holds that:

$$M_{BoS}, p \models [\mathtt{M}_{mt}, \neg\mathtt{t}]C_{\{a,b\}}\neg K_b p$$

Since this knowledge exists already in (M_{BoS}, p), i.e., $M_{BoS}, p \models C_{\{a,b\}}\neg K_b p$, the application of the action model does not affect any knowledge about the knowledge of Bob of the fact p. On the other hand, since Bob does not know whether Carol talked to Alice or not, it holds that

$$M_{BoS}, p \models [\mathtt{M}_{mt}, \neg\mathtt{t}]K_a\neg K_a p$$

albeit

$$M_{BoS}, p \not\models [\mathtt{M}_{mt}, \neg\mathtt{t}]K_b\neg K_a p.$$

However, since

$$M_{BoS}, p \models K_b\neg K_a p,$$

it is apparent that the application of $(\mathtt{M}_{mt}, \neg\mathtt{t})$ changes Bob's knowledge about the knowledge of Alice's knowledge of p.

bothmaytalk In this last scenario, the number of possible actions increases again. For both Alice and Bob we have all three possibilities we had in *maytalk*. Thus we can create the action model $\mathtt{M}_{bmt}$ for this scenario by composing two instances of the action model for *maytalk*, as depicted in Figure 3.9. The instance which handles Carols communication to Alice (resp. Bob) is denoted by $\mathtt{M}^a_{mt}$ (resp. $\mathtt{M}^b_{mt}$). An action of $\mathtt{M}^a_{mt}$ (resp. $\mathtt{M}^b_{mt}$) is written as $\mathtt{x}_a$ (resp. $\mathtt{x}_b$). Note that for simplicity Carols tries to talk to Alice before she considers talking to Bob.

Each possible composed action $(\mathtt{x}, \mathtt{y})$ contains the action Carol does regarding Alice $(\mathtt{x})$ and the one regarding Bob $(\mathtt{y})$. For example, $(\neg\mathtt{t}_a, \neg\mathtt{t}_b)$ denotes the action that Carol is talking to neither Alice nor Bob, while in action $(\neg\mathtt{t}_a, \mathtt{w}_b)$, Carol is only talking to Bob, telling him that United Agents is doing well. The precondition function is:

$$\begin{array}{lcl|lcl}
\mathtt{pre}(\neg\mathtt{t}_a, \neg\mathtt{t}_b) & = & \langle\mathtt{M}^a_{mt}, \neg\mathtt{t}_a\rangle\mathtt{pre}_b(\neg\mathtt{t}_b) & \mathtt{pre}(\neg\mathtt{t}_a, \mathtt{n}_b) & = & \langle\mathtt{M}^a_{mt}, \neg\mathtt{t}_a\rangle\mathtt{pre}_b(\mathtt{n}_b) \\
\mathtt{pre}(\neg\mathtt{t}_a, \mathtt{w}_b) & = & \langle\mathtt{M}^a_{mt}, \neg\mathtt{t}_a\rangle\mathtt{pre}_b(\mathtt{w}_b) & \mathtt{pre}(\mathtt{n}_a, \neg\mathtt{t}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{n}_a\rangle\mathtt{pre}_b(\neg\mathtt{t}_b) \\
\mathtt{pre}(\mathtt{n}_a, \mathtt{n}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{n}_a\rangle\mathtt{pre}_b(\mathtt{n}_b) & \mathtt{pre}(\mathtt{n}_a, \mathtt{w}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{n}_a\rangle\mathtt{pre}_b(\mathtt{w}_b) \\
\mathtt{pre}(\mathtt{w}_a, \neg\mathtt{t}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{w}_a\rangle\mathtt{pre}_b(\neg\mathtt{t}_b) & \mathtt{pre}(\mathtt{w}_a, \mathtt{n}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{w}_a\rangle\mathtt{pre}_b(\mathtt{n}_b) \\
\mathtt{pre}(\mathtt{w}_a, \mathtt{w}_b) & = & \langle\mathtt{M}^a_{mt}, \mathtt{w}_a\rangle\mathtt{pre}_b(\mathtt{w}_b) & & &
\end{array}$$

In our example, the precondition function is equivalent to:

$$\begin{array}{lcr|lcr|lcr}
\mathtt{pre}(\neg\mathtt{t}_a, \neg\mathtt{t}_b) & = & \top & \mathtt{pre}(\neg\mathtt{t}_a, \mathtt{n}_b) & = & \neg p & \mathtt{pre}(\neg\mathtt{t}_a, \mathtt{w}_b) & = & p \\
\mathtt{pre}(\mathtt{n}_a, \neg\mathtt{t}_b) & = & \neg p & \mathtt{pre}(\mathtt{n}_a, \mathtt{n}_b) & = & \neg p & \mathtt{pre}(\mathtt{n}_a, \mathtt{w}_b) & = & \bot \\
\mathtt{pre}(\mathtt{w}_a, \neg\mathtt{t}_b) & = & p & \mathtt{pre}(\mathtt{w}_a, \mathtt{n}_b) & = & \bot & \mathtt{pre}(\mathtt{w}_a, \mathtt{w}_b) & = & p
\end{array}$$

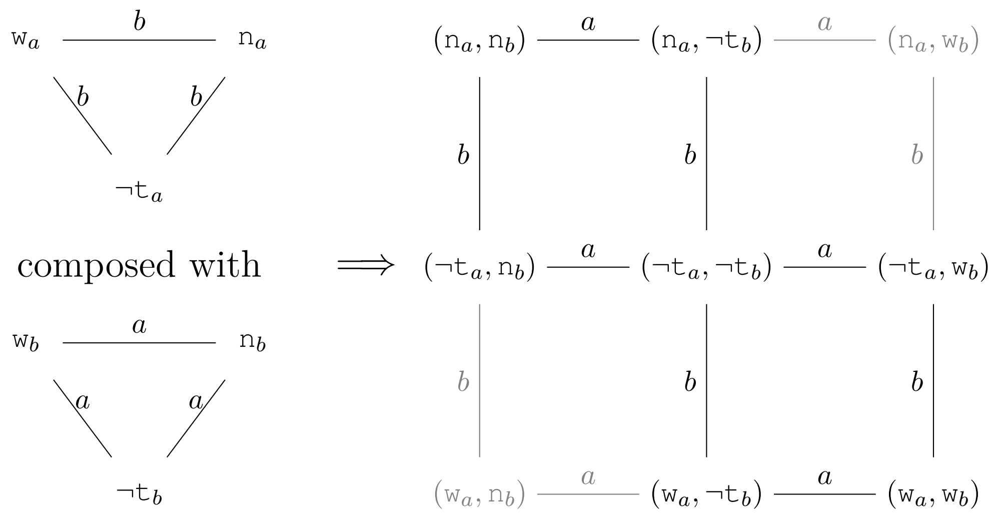


Figure 3.9: Example Buy or Sell: Composing the action model for *bothmaytalk*. Each action $(\mathtt{x}, \mathtt{y})$ contains the action regarding Alice, $\mathtt{x}$, and the action regarding Bob, $\mathtt{y}$. The gray actions $(\mathtt{n}_a, \mathtt{w}_b)$ and $(\mathtt{w}_a, \mathtt{n}_b)$ cannot be applied to any state of any epistemic model, since $\mathtt{pre}(\mathtt{n}_a, \mathtt{w}_b) = \langle \mathtt{M}^a_{mt}, \mathtt{n}_a\rangle \mathtt{pre}_b(\mathtt{w}_b)$ and $\mathtt{pre}(\mathtt{w}_a, \mathtt{n}_b) = \langle \mathtt{M}^a_{mt}, \mathtt{w}_a\rangle \mathtt{pre}_b(\mathtt{n}_b)$. Note that $\mathtt{pre}(\mathtt{n}_a, \mathtt{w}_b)$ and $\mathtt{pre}(\mathtt{w}_a, \mathtt{n}_b)$ are equivalent to $\bot$ in this example. Also note that in the composed action model, the transitive links are implicit.

Thus the actions $(\mathtt{n}_a, \mathtt{w}_b)$ and $(\mathtt{w}_a, \mathtt{n}_b)$ cannot be applied to any state of any epistemic model. Nevertheless they have to be present in the action model $\mathtt{M}_{bmt}$.

Figure 3.10 shows the resulting action model $(M_{BoS} \otimes \mathtt{M}_{bmt})$. The application of $(\mathtt{M}_{bmt}, (\mathtt{w}_a, \mathtt{w}_b))$, where Carol tells both Alice and Bob that United Agents is doing well, on (M_{BoS}, p) is again highlighted by underlining.

Since in the resulting epistemic model state $(M_{bmt}, (p, (\mathtt{w}_a, \mathtt{w}_b)))$, both a and b know p, we have:

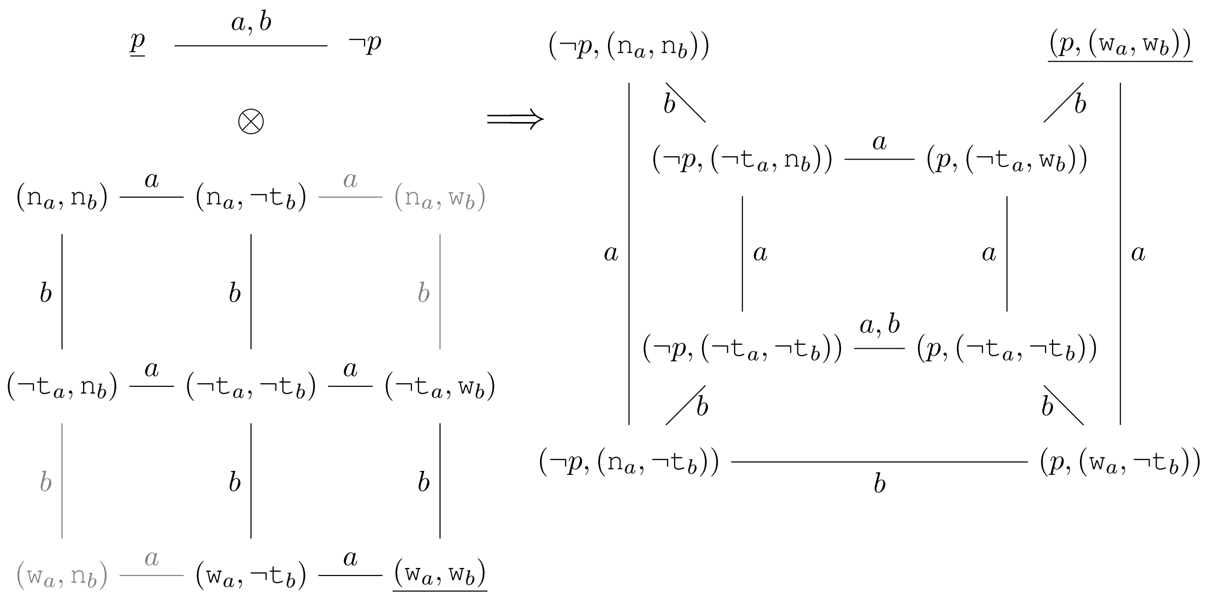
$$M_{BoS}, p \models [\mathtt{M}_{bmt}, (\mathtt{w}_a, \mathtt{w}_b)]E_{\{a,b\}}p$$

On the other hand, since there is a path from $(M_{bmt}, (p, (\mathtt{w}_a, \mathtt{w}_b)))$ to $(M_{bmt}, (\neg p, (\neg \mathtt{t}_a, \neg \mathtt{t}_b)))$ and other states in which $\neg p$,

$$M_{BoS}, p \not\models [\mathtt{M}_{bmt}, (\mathtt{w}_a, \mathtt{w}_b)]C_{\{a,b\}}p$$

3.3.3 Cheating Husbands

In our previous Buy or Sell examples, all communication was private (point-to-point). We will now use the Example 3.1.2 of the cheating husbands to discuss public announcements: Queen Henrietta I announced on the town square (in the presence of all the women in Mamajorca) that "there are (one or more) unfaithful husbands in our community." We will now create a corresponding action model $\mathtt{M}_{pub} = \langle \mathtt{S}, \sim, \mathtt{pre}\rangle$. Since all women in



$\underline{p}$
a, b
$\neg p$
$\otimes$
$\Longrightarrow$
$(\mathtt{n}_a, \mathtt{n}_b)$
$(\mathtt{n}_a, \neg\mathtt{t}_b)$
$(\mathtt{n}_a, \mathtt{w}_b)$
$(\neg\mathtt{t}_a, \mathtt{n}_b)$
$(\neg\mathtt{t}_a, \neg\mathtt{t}_b)$
$(\neg\mathtt{t}_a, \mathtt{w}_b)$
$(\mathtt{w}_a, \mathtt{n}_b)$
$(\mathtt{w}_a, \neg\mathtt{t}_b)$
$\underline{(\mathtt{w}_a, \mathtt{w}_b)}$
$(\neg p, (\mathtt{n}_a, \mathtt{n}_b))$
$\underline{(p, (\mathtt{w}_a, \mathtt{w}_b))}$
$(\neg p, (\neg\mathtt{t}_a, \mathtt{n}_b))$
$(p, (\neg\mathtt{t}_a, \mathtt{w}_b))$
$(\neg p, (\neg\mathtt{t}_a, \neg\mathtt{t}_b))$
$(p, (\neg\mathtt{t}_a, \neg\mathtt{t}_b))$
$(\neg p, (\mathtt{n}_a, \neg\mathtt{t}_b))$
$(p, (\mathtt{w}_a, \neg\mathtt{t}_b))$

Figure 3.10: Example Buy or Sell: Action model $\mathtt{M}_{bmt}$ for scenario *bothmaytalk*. Each action $(\mathtt{x}, \mathtt{y})$ contains the action $\mathtt{x}$ regarding Alice and $\mathtt{y}$ regarding Bob. The application of $(\mathtt{M}_{bmt}, (\mathtt{w}_a, \mathtt{w}_b))$ on (M_{BoS}, p) results in $(M_{BoS}, p)[\![(\mathtt{M}_{bmt}, (\mathtt{w}_a, \mathtt{w}_b))]\!](M_{bmt}, (p, (\mathtt{w}_a, \mathtt{w}_b)))$.

Mamajorca either hear the same announcement (≥ 1) or no announcement at all ($\neg\mathtt{t}$), and none of them considers it possible that any other of them hears anything else, we only need

$$\mathtt{S} = \{\geq 1, \neg\mathtt{t}\}.$$

As the actions are distinguishable for any woman, the indistinguishability relation for any process a is the trivial one consisting of self-loops only:

$$\sim_a = \{(\geq 1, \geq 1), (\neg\mathtt{t}, \neg\mathtt{t})\} = \mathit{loops}$$

And finally, as the queen is truthful and so is the queens announcement, we get the precondition function:

$$\mathtt{pre}(\geq 1) = \varphi$$
$$\mathtt{pre}(\neg\mathtt{t}) = \neg\varphi$$

with φ denoting the proposition that at least one husband is unfaithful. Looking at Figure 3.5, it is obvious that φ holds in every state except for 000.

Thus we can apply $\mathtt{M}_{pub}$ to the epistemic model M_{CH} of Figure 3.5 and obtain the epistemic model $M_{pub} = (M_{CH} \otimes \mathtt{M}_{pub})$ depicted in Figure 3.11. Suppose we started in $(M_{CH}, 011)$. After application of $(\mathtt{M}_{pub}, \geq 1)$, we are in epistemic state $(M_{pub}, (011, \geq 1))$.

Clearly, all epistemic formulas established in Section 3.2.4 for M'_{CH} in Figure 3.6 continue to hold in M_{pub}.

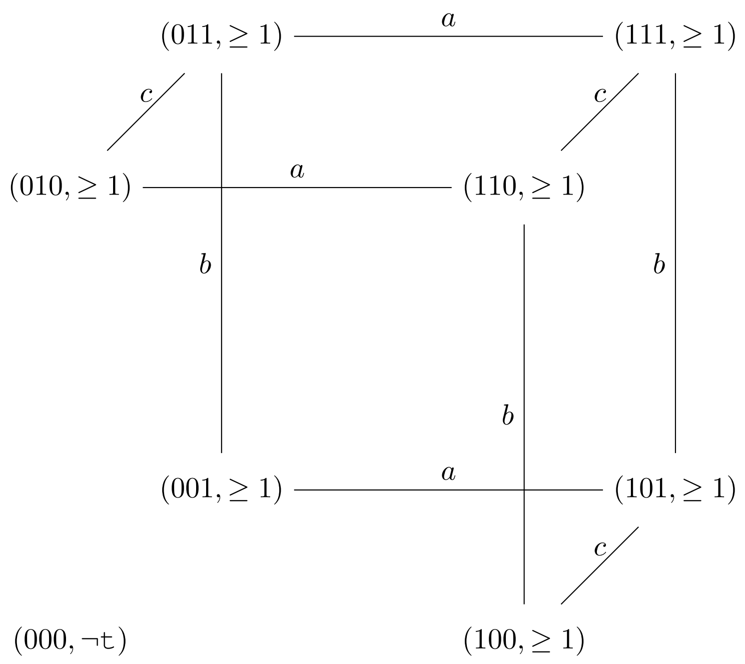


Figure 3.11: Example Cheating Husbands: Kripke model M_{pub} after the public announcement of queen Henrietta applied to the initial model M_{CH} from Figure 3.5

We will now look at the action model for the first night $\mathtt{M}_N = \langle \mathtt{S}_N, \sim_N, \mathtt{pre}_N \rangle$. Clearly there are two possible actions:

$$\mathtt{S}_N = \{\mathtt{shot}, \mathtt{silence}\}$$

As it is commonly known that all the women in Mamajorca hear every shot in town, we get the trivial indistinguishability relation for each process a:

$$\sim_{N,a} = \mathit{loops}$$

Of course the precondition on `shot` is that at least one woman i knows that her husband is unfaithful (denoted by proposition q_i), while the opposite is the precondition for `silence`.

$$\mathtt{pre}_N(\mathtt{shot}) = \bigvee_{i \in \{a,b,c\}} K_i q_i$$

$$\mathtt{pre}_N(\mathtt{silence}) = \neg \mathtt{pre}_N(\mathtt{shot})$$

The epistemic model $M'_{pub} = (M_{pub} \otimes \mathtt{M}_N)$ resulting from applying $\mathtt{M}_N$ to M_{pub} is shown in Figure 3.12. When doing so, we have to take care of two details. First, note that the precondition for the actions in $\mathtt{M}_N$ are not mere propositions any more, but epistemic formulas regarding the knowledge of all the processes. Applying the action model, we have to evaluate these formulas for every single state in M_{pub}, since for each of these states we have to look which of the actions may be applied. Second, $\texttt{pre(shot)}$ does not hold at $(M_{pub}, (011, \geq 1))$, since there is an indistinguishability relation to state $(001, \geq 1)$ for b and $(010, \geq 1)$ for c in which their husband is not unfaithful, so no wife knows that her husband is unfaithful. Thus there is no epistemic state $(M'_{pub}, ((011, \geq 1), \texttt{shot}))$ in M'_{pub}.

For the epistemic model M'_{pub} we find:

$$M'_{pub}, ((011, \geq 1), \texttt{silence}) \models K_b q_b \wedge K_c q_c$$

Thus,

$$M_{CH}, 011 \models [\mathtt{M}_{pub}, \geq 1][\mathtt{M}_N, \texttt{silence}] K_b q_b \wedge K_c q_c$$

And after the second night each wife knows whether she is being cheated or not. Applying $\mathtt{M}_N$ to M'_{pub}, we find $\texttt{pre}_N\texttt{(shot)} = \bigvee_{i \in \{a,b,c\}} K_i q_i$ is satisfied since b knows q_b and c knows q_c in $(M'_{pub}, ((011, \geq 1), \texttt{silence}))$. $\texttt{pre}_N\texttt{(shot)}$ is not satisfied in $(M'_{pub}, ((111, \geq 1), \texttt{silence}))$ since none of the women knows that her husband is cheating on her. In $(M''_{pub}, (((011, \geq 1), \texttt{silence}), \texttt{shot}))$ (after the second application of $\mathtt{M}_N$) it still holds that b knows q_b and c knows q_c but here also a knows $\neg q_a$. Hence,

$$M_{CH}, 011 \models [\mathtt{M}_{pub}, \geq 1][\mathtt{M}_N, \texttt{silence}][\mathtt{M}_N, \texttt{shot}] \bigwedge_{i \in \{a,b,c\}} (q_i \leftrightarrow K_i q_i).$$

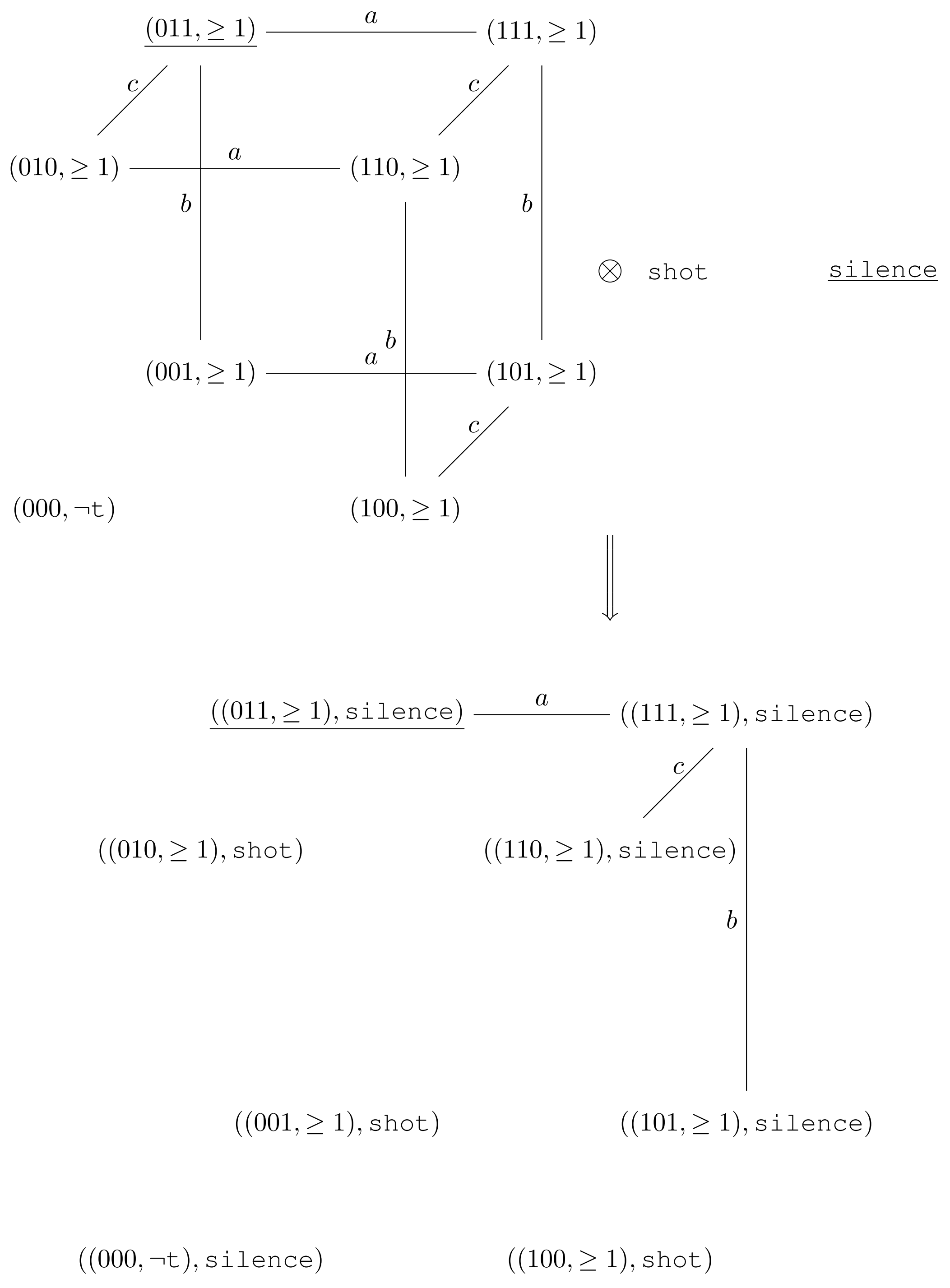


Figure 3.12: Example Cheating Husbands with public announcements: Kripke model M'_{pub}, after applying $(\mathtt{M}_N, \mathtt{silence})$ to $(M_{pub}, (011, \geq 1))$ in the first night.

3.3.4 Cheating Husbands: Scenario II

Note that the fact that “every woman in Mamajorca hears the same announcement, and none of them considers it possible that any other of them hears anything else” is the most crucial assumption allowing us to model queen Henrietta’s announcement as public announcement ($\mathtt{M}_{pub}$). To underline this statement we introduce a second scenario for the cheating husband problem:
Assume that all the women of Mamajorca know (e.g. by some magical understanding) that the queen only announces anything iff there is exactly one unfaithful husband, and that every of the other women knows this behavior of the queen as well, i.e., this fact is common knowledge. Iff there is exactly one unfaithful husband, the queen tells only his wife that her husband is unfaithful. All the other women will never hear anything from the queen and in no other case does the queen announce anything.

Also in this scenario the women are able to find all the cheating husbands. We will sketch the arguments why this is the case, again using proposition φ for “at least one husband is unfaithful” and q_i for “the husband of i is unfaithful”. Consider the case where $k = 1$ husband is cheating. Then all the women know about this husband, except for his wife, which will get the information by the queen and shoots him in the first night. All the other women then know that this wife did not know any other cheating husband and trust their own husbands.
Consider any case where $k = 2$ husbands are cheating. Then $E_A\varphi$, since every woman knows at least one cheating husband. As $k \neq 1$, the queen does not announce anything. Since every woman considers it possible that her husband is faithful, none of them shoots her husband in the first night. But then the two women knowing of exactly one unfaithful husband are shooting their husbands in the second night, since they know them to be unfaithful.
As in the previous scenario it can be proved by induction that for $k \geq 1$ cheating husbands all of them are shot in the k^{th} night.

Let us take a look at the action model $\mathtt{M}_{priv} = \langle \mathtt{S}_{priv}, \sim_{priv}, \mathtt{pre}_{priv} \rangle$ for this private announcement, as before for a set A of three women a, b and c. Suddenly there are four possible actions instead of just two trivial ones: telling nothing (denoted by $\neg\mathtt{t}$) and, for each $i \in \{a, b, c\}$, telling i that her husband is cheating (denoted by $\mathtt{t}_i$).

$$\mathtt{S}_{priv} = \{\neg\mathtt{t}, \mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}$$

As the queen only speaks to i in action $\mathtt{t}_i$, i can only distinguish $\mathtt{t}_i$ from the other three actions, for $j, l \neq i$:

$$\sim_{priv,i} = \{(\neg\mathtt{t}, \mathtt{t}_j), (\mathtt{t}_j, \neg\mathtt{t}), (\neg\mathtt{t}, \mathtt{t}_l), (\mathtt{t}_l, \neg\mathtt{t}), (\mathtt{t}_j, \mathtt{t}_l), (\mathtt{t}_l, \mathtt{t}_j)\} \cup loops$$

The preconditions are:

$$\mathtt{pre}_{priv}(\neg\mathtt{t}) = (q_a \wedge q_b) \vee (q_a \wedge q_c) \vee (q_b \wedge q_c) \vee (\neg q_a \wedge \neg q_b \wedge \neg q_c)$$

$$\mathtt{pre}_{priv}(\mathtt{t}_i) = q_i \wedge \neg q_j \wedge \neg q_l$$

The application of this action model is depicted in Figure 3.13. Note that the grey dotted edges in the action model are drawn only for sake of completeness since they are also transitively implied by the path via $\neg\texttt{t}$.

Applying this action model, it turns out that the result is the same as the one after the public announcement. This is an interesting fact, since in in the public announcement scenario, the women needed to know a priori (the, in real life trivial, fact) that the announcement is public, while in the private announcement scenario, the women needed to a priori know about a more complex situation. On the other hand, in the public scenario, the queen "sends out" a single bit to each of the women in each case, summing up to n bits in total for n women. In the private scenario, though, the queen only needs to send a single bit to a single woman in some special cases and sends nothing in most other cases. Thus, the communication complexity in the public scenario is higher than in the private one.

Intuition tells, that this is connected with the complexity of "a priori knowledge", the knowledge the women have initially, before any actions happen. Is this the case? Does the more complex a priori knowledge in the private scenario allows us to decrease communication complexity? If so, what does that mean for the complexity of the cheating husbands problem? We try to answer these questions in the next chapter.

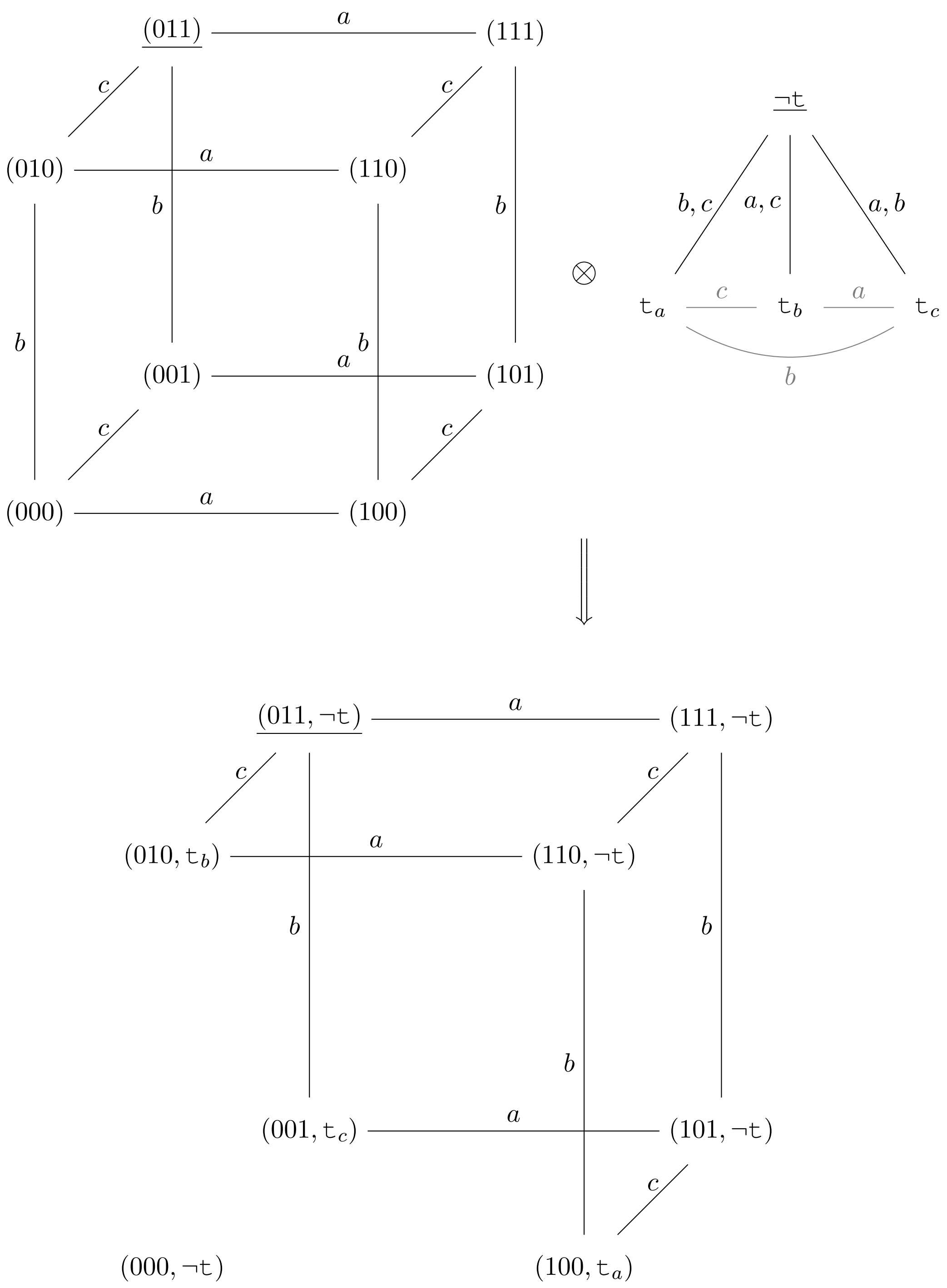


Figure 3.13: Example Cheating Husbands Scenario II with private announcements: Kripke model M_{priv}, after applying the private announcement ($\mathtt{M}_{priv}, \neg\mathtt{t}$) to $(M_{CH}, 011)$ as the queens announcement. Note that the grey edge in $\mathtt{M}_{priv}$ are drawn only for sake of completeness since they are also transitively implied by the paths via $\neg\mathtt{t}$.

CHAPTER 4

Action Models & Communication Complexity

4.1 Introduction

As conjectured at the end of the previous chapter, there may exist a connection between *a priori knowledge*, the knowledge processes have initially, and communication complexity, as introduced in Chapter 2. In this chapter we will go a step further, asking ourselves whether there might be a connection between action models, introduced in Chapter 3, and communication complexity.

We will informally discuss some ideas in the following section before presenting our results in Section 4.2 and Section 4.3.

4.1.1 Ideas

In Section 3.3.4, we recognized that there may be a connection between a priori knowledge and communication complexity. Actually, the answer to this question is "yes and no". Yes, the communication complexity seems to decrease with increasing a priori knowledge. But no, we do not think that the a priori knowledge itself is the reason for the change in communication complexity. We think that the reason for the change in communication complexity is linked to the number of possible events that might happen during an execution, a parameter which is perfectly captured in the applicable action model.

From another point of view, an action model simply defines the possible observations a single process can make. Note that these observations are often the same for all processes in a distributed system. Furthermore, one can typically assume that all of the processes *know* the action model and *know* that the other processes also rely on it. *Thus an action model has to be initially commonly known among all the processes.* So, yes, the *a* *priori common knowledge of the action models* seems to be the reason for the change in communication complexity.

4.1.2 Model

In all of our examples in the previous section (Buy or Sell? and both scenarios of Cheating Husbands), we implicitly used the synchronous model (with perfect communication) as defined in Section 1.3.

Note that the guarantee, that all the messages sent at the start of round r will be delivered by the end of round r, allows *communication by time.* I.e., if a process a did not receive any message of another process b by the end of round r, then *a knows* that b did not send such a message. On the other hand, if a sends a message to b at the start of round r, *a knows* at the end of round r that b received this message. [BZM14] made use of this fact by incorporating *NULL-messages*: The fact that a does not send any message to b in round r is denoted by "a sending a NULL-message to b". The fact that communication by time is possible is crucial for the results presented in the next section.

Regarding the connection of action models and the synchronous model, we assume that a corresponding action model is applied each round r:

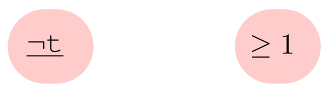
- At the start of round r, all the messages corresponding to the actual action are sent.
- By the end of round r, all such messages are delivered and, due to NULL messages, each process knows that the action model for round r has been applied.

4.2 An Action Model Lower Bound on Communication Complexity

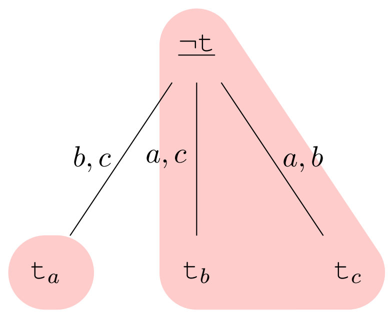
First we will take a look at the communication complexity involved in applying a single action model to some epistemic state. We will deduce our results starting from the two scenarios of the cheating husbands problem, introduced in Chapter 3. We first point out some properties of those examples, and then introduce the notion of *partitions* in action models and a related communication complexity measure based on those partitions.

Reconsider the public and private scenarios of the Cheating Husband problem. Figure 4.1 depicts the action models for the queen's announcement.

As explained previously, in $\mathtt{M}_{pub}$, the queen sends a single bit to each of the women, this bit states "There is at least one unfaithful husband". In $\mathtt{M}_{priv}$ the queen sends a single bit to a single woman, if and only if this woman is the only cheated one. This bit states "Your husband is the only unfaithful husband". Since the system runs in a synchronous setting, a woman knows — via communication by time — that, if the queen sent her a bit, she would have received it by a certain time τ, e.g., at the end of the day or at the



Action model $\mathtt{M}_{pub}$ modeling the public scenario of the cheating husbands problem.



Action model $\mathtt{M}_{priv}$ modeling the private scenario of the cheating husbands problem.

Figure 4.1: Action models for the two scenarios of Cheating Husbands. The actions are denoted by: $\neg\mathtt{t}\ldots$ the queen does not make a statement, $\geq 1\ldots$ the queen publicly announces that there is at least one unfaithful husband, $\mathtt{t}_i\ldots$ the queens tells i privately that her husband is unfaithful. The partitioning regarding a is depicted in red.

end of the current round. Thus a woman can distinguish the actions by time τ, as she knows that the action model has been applied.

Considering just a single woman, w.l.o.g. choose a, we see that both of the action models are *partitioned* regarding the indistinguishability of a (depicted in Figure 4.1).

Definition 4.2.1 (Partitions of Action Models)**.** An Action Model $\mathtt{M} = \langle \mathtt{S}, \sim, \mathtt{pre} \rangle$ is *partitioned regarding process a* if the underlying indistinguishability graph, consisting only of edges corresponding to $\sim_a$, is partitioned. The number of those partitions is denoted by $N^a_{\mathtt{M}}$.

Our claim is that there is a strong connection between the communication complexity (here the number of bits received by process a) and the number of such partitions $N^a_{\mathtt{M}}$ of the action model $\mathtt{M}$. We can see that, for each woman i in the Cheating Husbands problem, both of the action models $\mathtt{M}_{pub}$ and $\mathtt{M}_{priv}$ partition into two partitions. Indeed, if the queen does not announce anything in $\mathtt{M}_{priv}$, woman a does NOT know that the action has been $\neg\mathtt{t}$, she only knows that the action has been in the partition built by $\{\neg\mathtt{t}, \mathtt{t}_b, \mathtt{t}_c\}$.

Since each of the women has to be able to identify the actual partition the current action is in, the number of these partitions determines a lower bound on the number of bits received by any woman in some scenario, i.e., a worst-case lower bound: In Figure 4.1, if ≥ 1 (in $\mathtt{M}_{pub}$) or $\mathtt{t}_a$ (in $\mathtt{M}_{priv}$) occurs, woman a of course immediately knows the action itself, without receiving any additional information. In the case of $\neg\mathtt{t}$ (in $\mathtt{M}_{pub}$) or an action other than $\mathtt{t}_a$ (in $\mathtt{M}_{priv}$), she learns the partition by not receiving anything, i.e., via communication by time. Consequently, as both action models split into two partitions for each woman, the queen has to send one bit to each of them in BOTH scenarios.

This does NOT contradict our above observation that, in $\mathtt{M}_{priv}$, the queen sends only a single bit to a single woman. In more detail, consider the actions regarding woman a in the model $\mathtt{M}_{priv}$.

- Action $\mathtt{t}_a$: The queen *actively* sends a *bit* $= 1$ to woman a.
- Actions $\neg\mathtt{t}$, $\mathtt{t}_b$, $\mathtt{t}_c$: The queen does not *actively* send anything to woman a. Since woman a has a notion of time and she does not receive any bit up to a certain time τ, she can conclude that the queen did not send her any bit actively. Thus, by not sending a bit, the queen sent her some information, namely a *passive* bit interpreted as $\emptyset$ via a NULL message.

Definition 4.2.2. We define an *active* bit as a bit (i.e., 0 or 1) sent via explicit communication from some process a to some process b. A *passive* bit is defined as the bit "sent" in a NULL message from some process a to some process b, i.e., communication by time.

Be aware that, by this definition, multiple active bits can be sent from a to b, while a NULL message in round r counts as a single passive bit only.

We are ready to define the cost of the application of a single action model:

Definition 4.2.3. The worst-case cost $D^a(\mathtt{M})$ of the application of an action model $\mathtt{M}$ regarding process a is the worst-case number of *active* bits received by a when the action model is applied, i.e., the maximum number of active bits received in some scenario.

Note carefully that it is the particular algorithm that actually determines the encoding used for communicating the occurrence of the actions to the processes. The number of active bits received by a may hence depend on which particular action occurs, which explains why we restrict our attention to the maximum number of active bits for defining $D^a(\mathtt{M})$. Of course, this implies that we can only guarantee that $D^a(\mathtt{M})$ bits are sent in *some* scenario, not in *any* scenario. Even worse, we cannot assume that the action that causes the worst-case cost $D^a(\mathtt{M})$ for process a is the same as the action that causes the worst-case cost $D^b(\mathtt{M})$ for process b. Therefore, defining the total worst-case cost $D(\mathtt{M})$ of the application of an action model $\mathtt{M}$ as the sum of $D^a(\mathtt{M})$ over all processes a, would be overly conservative, and does hence not give a lower bound for the system-wide communication complexity.

However, we can give a lower bound for $D^a(\mathtt{M})$:

Lemma 4.2.1. *In a synchronous system with processes A, the worst-case cost $D^a(\mathtt{M})$ of the application of an action model $\mathtt{M}$ for process $a \in A$ satisfies:*

$$log_2\ (N^a_{\mathtt{M}} - 1) \leq D^a(\mathtt{M}),$$

where $N^a_{\mathtt{M}}$ is the number of partitions regarding a in $\mathtt{M}$.

Proof by contradiction. Suppose there exists an action model $\mathtt{M}$ such that $D^a(\mathtt{M}) < log_2\ (N^a_{\mathtt{M}} - 1)$ for some process $a \in A$. Then $2^{D^a(\mathtt{M})} + 1 < N^a_{\mathtt{M}}$. It is easy to see that receiving $D^a(\mathtt{M})$ active bits with value 0 or 1, a can distinguish at most $2^{D^a(\mathtt{M})} + 1$ (including the single passive bit) partitions of $\mathtt{M}$. Since $2^{D^a(\mathtt{M})} + 1 < N^a_{\mathtt{M}}$, by a pigeonhole argument, there are at least two partitions P_0 and P_1 which cannot be distinguished by a.

Now assume that applying $(\mathtt{M}, \mathtt{s})$ at epistemic state (M, s) results in (M', s'), i.e.,

- $s_0 \sim_a s_1$ in epistemic model M,
- $\mathtt{s}_0 \in P_0$, $\mathtt{s}_1 \in P_1$ in action model $\mathtt{M}$ applicable to s_0 respectively s_1 (P_0 and P_1 indistinguishable by a), and
- $(s_0, \mathtt{s}_0) \not\sim_a (s_1, \mathtt{s}_1)$ in epistemic model M'.

If a cannot distinguish between the actions in P_0 and P_1, then, since $s_0 \sim_a s_1$, $(s_0, \mathtt{s}_0) \sim_a (s_1, \mathtt{s}_1)$ in epistemic model M' contradicting the result of applying $(\mathtt{M}, \mathtt{s})$ to (M, s).

Thus a has to receive at least $log_2\ (N^a_{\mathtt{M}} - 1)$ active bits during the application of $(\mathtt{M}, \mathtt{s})$. □

As we will apply our lower bound method to synchronous systems with lossy links also, we cannot always assume that communication by time can be used. The following Lemma 4.2.2 provides a lower bound for $D^a(\mathtt{M})$ in this case.

Lemma 4.2.2. *In a system with processes A, the worst-case cost of $D^a(\mathtt{M})$ of the application of an action model $\mathtt{M}$ for process $a \in A$ satisfies:*

$$log_2\ (N^a_{\mathtt{M}}) \leq D^a(\mathtt{M}),$$

where $N^a_{\mathtt{M}}$ is the number of partitions regarding a in $\mathtt{M}$.

Proof by contradiction. This proof follows the same argument as the proof for Lemma 4.2.1.

Suppose there exists an action model $\mathtt{M}$ such that $D^a(\mathtt{M}) < log_2(N^a_{\mathtt{M}})$ for some process $a \in A$. Then $2^{D^a(\mathtt{M})} < N^a_{\mathtt{M}}$. It is easy to see that receiving $D^a(\mathtt{M})$ active bits with value 0 or 1, a can distinguish at most $2^{D^a(\mathtt{M})}$ (note that in the general setting there is no communication by time and thus no passive bit can be sent or received) partitions of $\mathtt{M}$.

The remaining proof is done by a chain of arguments analogous to the proof of Lemma 4.2.1, which shows that there are at least two partitions which are indistinguishable for a. Thus, a has to receive at least $log_2(N^a_{\mathtt{M}})$ (active) bits during the application of $(\mathtt{M}, \mathtt{s})$. □

In this Section 4.2, we only considered the application of a single action model. Looking at the communication complexity of an algorithm $\mathcal{A}$ solving a specific problem $\mathcal{P}$ using multiple rounds of communication, the first thing that comes to mind with the above method is to sum up the communication complexity of the single rounds. Unfortunately, this would not provide a tight lower bound on communication complexity of algorithm $\mathcal{A}$ solving $\mathcal{P}$, as the worst-case scenario for round r action model $\mathtt{AM}_r$ is not necessarily the same as the worst-case scenario for the action model $\mathtt{AM}_{r'}$ in round r'. Fortunately, however, Definition 3.3.3 may provide a way to circumvent this problem: By computing the composition of the action models of rounds $1, 2, \ldots, k$, where k is the round in which $\mathcal{A}$ has terminated, we get a *single* action model for which we can compute the lower bound using the above method.

We conclude this section by stressing the fact that the worst-case cost $D^a(\mathtt{M})$ given by Definition 4.2.3 is tied to the communication complexity for applying a given action model $\mathtt{M}$, i.e., of an algorithm $\mathcal{A}$ that implements $\mathtt{M}$. It is not related to the specific problem solved by $\mathcal{A}$, in particular, the lower bounds for $D^a(\mathtt{M})$ established in Lemma 4.2.1 and Lemma 4.2.2 are not the communication complexity lower bounds for the problem at hand. We will shed some light on this issue in the following section.

4.3 A Yao Lower Bound on Communication Complexity

In this section, we will develop an alternative method to infer a lower bound on the communication complexity of an algorithm $\mathcal{A}$ solving a specific problem $\mathcal{P}$. This method is based on the approach of [Yao79]. We will restrict ourselves to problems $\mathcal{P}$ in which two processes p_0 (holding x) and p_1 (holding y) both have to compute the result of a function $f(x, y)$ in the reliable synchronous message passing model. Each algorithm computing such a function can be defined using a single action model for each round. As described in Chapter 3, one can either apply the action models of the single rounds one-by-one on the initial epistemic model or can compute the composed action model and apply it once on the initial epistemic state.

4.3.1 Model

Since our method is based on the approach of [Yao79], we use a model quite similar to his: We restrict ourselves to a scenario and model similar to the one described in Section 2.1, with the differences that (i) each process can send an arbitrary number of bits in each round, (ii) that we consider symmetric function computation without communication by time, and (iii) once an algorithm for computing $f(x, y)$ terminates, the result shall be common knowledge among the processes p_0 and p_1.

4.3.2 Action Models and Protocol Trees

In our setting, all actions correspond to messages sent between the two processes. Thus all actions are executed and communicated by one of the two processes.

Since our message passing model is synchronous and reliable here, each action can be distinguished from any other action by both processes at the end of a round. As process a can only send some information x to process b if it knows that x is valid, an action corresponding to this sending process has to have a precondition containing $K_a x$. Consequently, even though process a can distinguish the action in which a sends x to b in round r from the action a sends $\neg x$ to b in round r, the application of one of those actions does not change a's view on the facts x and $\neg x$ in the resulting epistemic state compared to the original one, as a already knew x resp. $\neg x$. On the other hand, since b can distinguish the actions x and $\neg x$, b learns x resp. $\neg x$, which eliminates edges in $\sim_b$ in the resulting epistemic model, leading to a partitioning between the states where $K_b x$ and $K_b \neg x$.

To be precise, this is only true if x is a *preserved formula* (as introduced in [vDvdHK08]), which requires x to be propositional or positive knowledge (but not $x = \neg K_a \varphi$, for example). Thus we will also restrict ourselves to algorithms in which processes communicate only by means of preserved formulas.

Note that the fact that all actions are distinguishable for each process is only valid because we restrict the scenario to two processes: In a system with three processes, it would be possible that process p_0 doing actions $\mathtt{s}$ resp. $\mathtt{t}$ is sending 0 resp. 1 to process p_1 but nothing to process p_2, thus p_2 cannot distinguish actions $\mathtt{s}$ and $\mathtt{t}$.

Nevertheless, even in a system of n processes, the terminal epistemic model, in which the n processes all know the result of f, must be partitioned into several partitions that are separated for *all* processes: Each such partition consists of (potentially multiple) epistemic states, which are similar for all processes and in which the result of f is the same. Otherwise, the result of f would not be common knowledge.

Since all the processes have the same initial knowledge (except for their own initial value), the initial epistemic model $M = (S, \sim, V)$ is not partitioned, but a hyper-cube. An example can be found in the initial model of the Cheating Husbands in Figure 3.5. As already discussed earlier, the terminal epistemic model is the result of applying the composed action model to the initial epistemic model. Thus, the partitioning of the terminal epistemic model results from a partitioning of the composed action model.

We will use this partitioning of the composed action model to build a protocol tree of the algorithm.

Definition 4.3.1. An algorithm $\mathcal{A}$ is defined by a set of action models $\{\mathtt{AM}_1, \ldots, \mathtt{AM}_k\}$, such that a single action model $\mathtt{AM}_i$ is applicable in round i of the synchronous execution. The action model $\mathtt{AM}_i$ partitions into $t_{\mathtt{AM}_i}$ disjoint partitions (for every process p_0, p_1).

Definition 4.3.2. The composed action model of the first k rounds ($\mathtt{CAM}_k$) is defined as the composition of the action models $\mathtt{AM}_1, \ldots, \mathtt{AM}_k$. The inductive definition is:

- $\mathtt{CAM}_1 = \mathtt{AM}_1$
- $\mathtt{CAM}_k = (\mathtt{CAM}_{k-1}; \mathtt{AM}_k)$

The composed action model $\mathtt{CAM}_k = (\mathtt{S}_{\mathtt{CAM}_k}, \sim_{\mathtt{CAM}_k}, \mathtt{pre}_{\mathtt{CAM}_k})$ partitions into $t_{\mathtt{CAM}_k}$ disjoint partitions. Denote the i-th partition of $\mathtt{CAM}_k$, consisting of actions $\mathtt{S}_{k,i} \subseteq \mathtt{S}_{\mathtt{CAM}_k}$, by $\mathtt{P}_{k,i}$.

Note that applying the actions in $\mathtt{P}_{k,i}$ to the initial epistemic model $M = (S, \sim, V)$ leads to a set of partitions of the epistemic model $M' = M \otimes \mathtt{CAM}_k$.

Clearly, if $\mathcal{A}$ computes f in m rounds, the resulting composed action model is $\mathtt{CAM}_m$, the composition of the action models of the first m rounds.

Definition 4.3.3. The protocol tree $\mathcal{T}_{\mathcal{A}} = (V, E)$ of an algorithm $\mathcal{A}$, starting at the root vertex v that represents the initial epistemic model $M = (S, \sim, V)$, is defined as:

- $V = \{v\} \cup \{\mathtt{P}_{k,i} \mid \mathtt{P}_{k,i}$ for some i is a partition of $\mathtt{CAM}_k$, $k \in \{1, m\}\}$

- $E = \{(v, \mathtt{P}_{1,j}) \mid \mathtt{P}_{1,j}$ a partition of $\mathtt{CAM}_1\} \cup \{(\mathtt{P}_{k,i}, \mathtt{P}_{k+1,j}) \mid \exists \mathtt{s} \in \mathtt{S}_{k,i}, \mathtt{t} \in \mathtt{S}_{\mathtt{AM}_{k+1}} : (\mathtt{s}, \mathtt{t}) \in \mathtt{S}_{k+1,j}$ for some i and j, and $k \in \{1, m-1\}\}$.

Definition 4.3.3 states that each partition of each composed action model $\mathtt{CAM}_k$ is a node in the protocol tree. All the nodes corresponding to the partitions of $\mathtt{CAM}_1$ are connected to the root node v. There is a connection between two nodes $\mathtt{P}_{k,i}$ and $\mathtt{P}_{k+1,j}$ if and only if they are on levels k and $k+1$ and there is an action $\mathtt{s}$ in $\mathtt{S}_{k,i}$ (the set of actions corresponding to $\mathtt{P}_{k,i}$) which is a prefix of an action $(\mathtt{s}, \mathtt{t})$ of $\mathtt{S}_{k+1,j}$ (the set of actions corresponding to $\mathtt{P}_{k+1,j}$), with $\mathtt{t} \in \mathtt{S}_{\mathtt{AM}_{k+1}}$ an action of $\mathtt{AM}_{k+1}$. Lemma 4.3.1 proves that $\mathcal{T}_{\mathcal{A}}$ is indeed a tree.

Lemma 4.3.1. *$\mathcal{T}_{\mathcal{A}}$, defined in Definition 4.3.3, is a tree.*

Proof. First we will prove that each node in $\mathcal{T}_{\mathcal{A}}$ (except for v) has at least one predecessor: To do so, we prove that there is no node at level k which is not connected to any node at level $k-1$.
For the nodes on level $k = 1$ this is trivial, since they are by definition connected to the node v, the single root of the tree.
For the other nodes assume that there is a node on some level $k > 1$, which is not connected to any node at level $k-1$. This would mean that there is a node $\mathtt{P}_{k,i}$ on level k such that none of its actions $(\mathtt{s}, \mathtt{t})$ fulfills $\mathtt{s} \in \mathtt{S}_{k-1,j}, \mathtt{t} \in \mathtt{S}_{AM_k}$ for any node $\mathtt{P}_{k-1,j}$ on level $k-1$. Since $\mathtt{P}_{k,i}$ is a partition of $\mathtt{CAM}_k$, and thus all of its actions are actions in the set of $\mathtt{S}_{\mathtt{CAM}_k}$, this contradicts the fact that $\mathtt{CAM}_k = (\mathtt{CAM}_{k-1}; \mathtt{AM}_k)$ and Definition 3.3.3.

Next we will prove that each node in $\mathcal{T}_{\mathcal{A}}$ has at most one predecessor:
Suppose there are two nodes $\mathtt{P}_{k,1}$ and $\mathtt{P}_{k,2}$ on level k, with actions $\mathtt{s}_1 \in \mathtt{S}_{k,1}$ and $\mathtt{s}_2 \in \mathtt{S}_{k,2}$, which are both connected to the same node $\mathtt{P}_{k+1,i}$ on level $k+1$. Since $\mathtt{s}_1$ and $\mathtt{s}_2$ are in two different partitions, we see that $\mathtt{s}_1 \not\sim_{p_0} \mathtt{s}_2 \wedge \mathtt{s}_1 \not\sim_{p_1} \mathtt{s}_2$. But as both $\mathtt{P}_{k,1}$ and $\mathtt{P}_{k,2}$ are connected to $\mathtt{P}_{k+1,i}$, it also holds that $(\mathtt{s}_1, \mathtt{t}_1) \sim_{p_0} (\mathtt{s}_2, \mathtt{t}_2)$ or $(\mathtt{s}_1, \mathtt{t}_1) \sim_{p_1} (\mathtt{s}_2, \mathtt{t}_2)$ for some $\mathtt{t}_1, \mathtt{t}_2 \in \mathtt{S}_{\mathtt{AM}_{k+1}}$. This contradicts Definition 3.3.3. □

So far, we only considered the protocol tree $\mathcal{T}_{\mathcal{A}}$ that is solely defined in terms of the action models $\mathtt{CAM}_k$. Now we turn our attention to the application of $\mathcal{T}_{\mathcal{A}}$ to the initial epistemic model $M = (S, \sim, V)$ that is a hypercube. As already said, it must induce a partitioning of the resulting epistemic model, i.e., the leaves in $\mathcal{T}_\mathcal{A}$, which allows to solve $f(x,y)$. The following Lemma 4.3.2 shows that the $\texttt{CAM}_m$ and the resulting $\mathcal{T}_\mathcal{A}$ of a correct solution must induce rectangles at the leafs of $\mathcal{T}_\mathcal{A}$.

Lemma 4.3.2. *Let $M = (S, \sim, V)$ be the hypercube describing the initial epistemic model of a solution algorithm for computing $f(x,y)$, defined by the action models $\texttt{AM}_1, \ldots, \texttt{AM}_m$ (resulting in the composed action model $\texttt{CAM}_m$) and the resulting $\mathcal{T}_\mathcal{A}$. Then, every rectangle corresponds to at least one partition, in the final epistemic model $M' = M \otimes \texttt{CAM}_m$, i.e., at least one leaf, and every leaf corresponds to some (not necessarily maximal) rectangle.*

Proof. First, as $\mathcal{A}$ must compute $f(x,y)$ for every input (x,y), and $\mathcal{A}$ terminates only in leaves of $\mathcal{T}_\mathcal{A}$, every (x,y) leads to some leaf. Consequently, for every rectangle $\mathcal{R}$, which usually contains more than one input, say (x_1, y_1) and (x_2, y_2), we can assign the set of leafs $L_\mathcal{R}$ its constituent inputs lead to.

We now show that actually $|L_\mathcal{R}| = 1$, which implies that every leaf corresponds to some rectangle. Similar as in Section 2.2, suppose both inputs (x_1, y_1) and (x_2, y_2) allow the application of actions leading to the node ℓ of $\mathcal{T}_\mathcal{A}$, then also (x_1, y_2) and (x_2, y_1) lead to ℓ: The path through the tree has to be the same for all of the four input pairs. We start our inductive argument at level 0, the initial epistemic model. In the initial epistemic model, p_0 cannot distinguish the situation with input (x_1, y_1) from (x_1, y_2) resp. (x_2, y_1) from (x_2, y_2). A similar argument holds for p_1. Since (x_1, y_1) and (x_2, y_2) lead to the same node ℓ, the actions of $\texttt{AM}_1$ have to be in the same partition for both of them and since p_0 cannot distinguish (x_1, y_1) from (x_1, y_2), the action applied by p_0 has to be the same in both cases (similarly for (x_2, y_2) and (x_2, y_1)). Since p_1 cannot distinguish (x_1, y_1) from (x_2, y_1), the action applied by p_1 has to be the same in both cases (similarly for (x_2, y_2) and (x_1, y_2)). As p_0's action is the same for (x_1, y_1) and (x_1, y_2) and p_1's action is the same for (x_1, y_2) and (x_2, y_2), and the actions of $\texttt{AM}_1$ have to be in the same partition for (x_1, y_1) and (x_2, y_2), also the action for (x_1, y_2) has to be in the same partition in $\texttt{AM}_1$. By the analogous argument, it follows that also the action for (x_2, y_1) in $\texttt{AM}_1$ is in the very same partition of $\texttt{AM}_1 = \texttt{CAM}_1$.
Now suppose the execution of $\mathcal{A}$ for (x_1, y_1) resp. (x_2, y_2) reached some node $\texttt{P}_{k,i}$ on level k of $\mathcal{T}_\mathcal{A}$. By the induction hypothesis also the executions for (x_1, y_2) and (x_2, y_1) have reached this node. Due to the initial premise of reaching the same leaf ℓ, the executions for (x_1, y_1) and (x_2, y_2) must reach some common node $\texttt{P}_{k+1,j}$ corresponding to a partition in $\texttt{CAM}_{k+1} = (\texttt{CAM}_k; \texttt{AM}_{k+1})$. As already stated before, the epistemic model after round $k+1$ can be derived in two ways: Applying action model by action model or once applying $\texttt{CAM}_{k+1}$ on the initial epistemic model: the resulting epistemic models are equivalent. Thus, by the same argument as before (only using $\texttt{AM}_{k+1}$ instead of $\texttt{AM}_1$), it follows that all the actions on the inputs have to be in the same partition in $\texttt{AM}_{k+1}$ and hence in $\texttt{CAM}_{k+1}$. Consequently, all the inputs lead to the same node on level $k+1$ as asserted. □

We proceed with an example using the function $f(x, y)$ given in Figure 4.2, which has to be computed by the processes p_0 (having x as its input) and p_1 (with input y). There are of course several algorithms for computing this function. For example, look at the action models for the multiple rounds of algorithm $\mathcal{A}$ in Example 4.3.1, where, as in the original Yao-Scenario (Figure 2.1 and Figure 2.2), the processes send a single bit in each round alternatingly. Note that $\mathcal{A}$ is optimal in terms of communication complexity.

	y_0	y_1	y_2	y_3
x_0	1	1	0	1
x_1	0	0	1	1
x_2	0	0	1	1
x_3	0	0	0	0

Figure 4.2: The function f computed in the following examples.

Example 4.3.2 shows the action models for the multiple rounds of a different algorithm $\mathcal{B}$. In the first round of $\mathcal{B}$, the process p_0 sends the information whether its input is x_0, x_1, x_2 or x_3 (using two bits), while process p_1 only sends the information whether its input is in $\{y_0, y_1\}$ or $\{y_2, y_3\}$ (using one bit). If the result is not computed at the end of the first round, which is the case if p_0's input is x_0 and p_1's input is either y_2 or y_3, p_1 again sends a single bit stating whether its input is y_2 or y_3.

In both of the examples, an action of the form e.g. (x_0, x_1) depicts that p_0 sends the information that its input value is either x_0 or x_1 to p_1, while $K_i x_0$ in a precondition formula means that, in the appropriate epistemic state, p_i knows that x_0 is the input to p_0.

We depict by $\mathcal{R}^f_{\mathcal{A}}$ the set of rectangles that corresponds to the leaves of $\mathcal{T}_{\mathcal{A}}$.

Example 4.3.1. The action models for the optimal algorithm $\mathcal{A}$ for f, given in Figure 4.2, are the following:

$$\begin{array}{llcl}
\texttt{AM}_1: & \texttt{S}_{\texttt{AM}_1} & = & \{(x_0),(x_1,x_2,x_3)\} \\
& \sim_{p_i} & = & \textit{loops} \\
& \texttt{pre}(x_0) & = & K_0 x_0 \\
& \texttt{pre}((x_1,x_2,x_3)) & = & K_0(x_1 \vee x_2 \vee x_3)
\end{array}$$

$$\begin{array}{llcl}
\texttt{AM}_2: & \texttt{S}_{\texttt{AM}_2} & = & \{(y_0,y_1,y_3),y_2,(y_0,y_1),(y_2,y_3)\} \\
& \sim_{p_i} & = & \textit{loops} \\
& \texttt{pre}((y_0,y_1,y_3)) & = & K_1(x_0 \wedge (y_0 \vee y_1 \vee y_3)) \\
& \texttt{pre}(y_2) & = & K_1(x_0 \wedge y_2) \\
& \texttt{pre}((y_0,y_1)) & = & K_1((x_1 \vee x_2 \vee x_3) \wedge (y_0 \vee y_1)) \\
& \texttt{pre}((y_2,y_3)) & = & K_1(x_1 \vee x_2 \vee x_3) \wedge (y_2 \vee y_3))
\end{array}$$

$$\begin{array}{llcl}
\texttt{AM}_3: & \texttt{S}_{\texttt{AM}_2} & = & \{x_3,(x_1,x_2)\} \\
& \sim_{p_0} & = & \textit{loops} \\
& \texttt{pre}(x_3) & = & K_0(x_3 \wedge (y_2 \vee y_3)) \\
& \texttt{pre}((x_1,x_2)) & = & K_0((x_1 \vee x_2) \wedge (y_2 \vee y_3))
\end{array}$$

The resulting composed action model is $\texttt{CAM}_3$:

$$\begin{array}{rcl}
\texttt{S}_{\texttt{CAM}_3} & = & \{(x_0,(y_0,y_1,y_3)),(x_0,y_2), \\
& & ((x_1,x_2,x_3),(y_0,y_1)), \\
& & (((x_1,x_2,x_3),(y_2,y_3)),x_3), \\
& & (((x_1,x_2,x_3),(y_2,y_3)),(x_1,x_2))\} \\
\sim_{p_i} & = & \textit{loops} \\
\texttt{pre}((x_0,(y_0,y_1,y_2))) & = & K_0 x_0 \wedge K_1(y_0 \vee y_1 \vee y_3) \\
\texttt{pre}((x_0,y_2)) & = & K_0 x_0 \wedge K_1 y_2 \\
\texttt{pre}((x_1,x_2,x_3),(y_0,y_1)) & = & K_0(x_1 \vee x_2 \vee x_3) \wedge K_1(y_0 \vee y_1) \\
\texttt{pre}(((x_1,x_2,x_3),(y_2,y_3)),x_3) & = & K_0 x_3 \wedge K_1(y_2 \vee y_3) \\
\texttt{pre}(((x_1,x_2,x_3),(y_2,y_3)),(x_1,x_2)) & = & K_0(x_1 \vee x_2) \wedge K_1(y_2 \vee y_3)
\end{array}$$

The corresponding protocol tree $\mathcal{T}_\mathcal{A}$ is given in Figure 4.3.

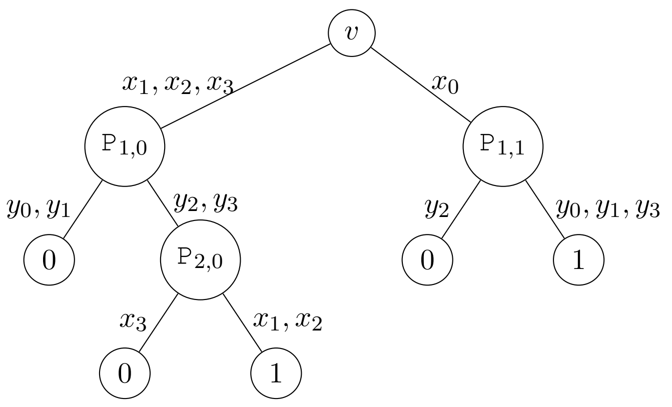


Figure 4.3: The protocol tree $\mathcal{T}_\mathcal{A}$ for function f defined in Figure 4.2.

The rectangles $\mathcal{R}^f_{\mathcal{A}}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{A}}$ are depicted in Figure 4.4.

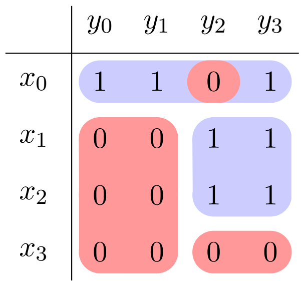


Figure 4.4: The rectangles $\mathcal{R}^f_{\mathcal{A}}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{A}}$ in Figure 4.3.

We see that there are 5 completely separated partitions in $\texttt{CAM}_3$ corresponding to 5 leaves in the protocol tree $\mathcal{T}_{\mathcal{A}}$. Since $\mathcal{A}$ follows the original Yao protocol, we get that $\mathcal{A}$ has to send at least $log_2(5) \leq 3$ bit to compute f. Another way to look at it is that in the protocol tree of height 3, in each of the 3 rounds at least 1 bit is received (as the communication is reliable and at least 1 bit is sent).

Example 4.3.2. Since in the first round of $\mathcal{B}$ both processes send a message, the precondition for each of the actions in $\texttt{AM}_1$ is of the form $K_0 a \wedge K_1 b$, with a and b denoting the information in the message sent by p_0 resp. p_1. The action models for the alternative algorithm $\mathcal{B}$ computing f, given in Figure 4.6, are the following:

$$\begin{aligned}
\texttt{AM}_1: \qquad \mathsf{S}_{\texttt{AM}_1} &= \{(x_0,(y_0,y_1)),(x_0,(y_2,y_3)),\\
&\quad (x_1,(y_0,y_1)),(x_1,(y_2,y_3)),\\
&\quad (x_2,(y_0,y_1)),(x_2,(y_2,y_3)),\\
&\quad (x_3,(y_0,y_1)),(x_3,(y_2,y_3))\}\\
\sim_{p_i} &= \mathit{loops}\\
\texttt{pre}((x_0,(y_0,y_1))) &= K_0x_0 \wedge K_1(y_0 \vee y_1)\\
\texttt{pre}((x_0,(y_2,y_3))) &= K_0x_0 \wedge K_1(y_2 \vee y_3)\\
\texttt{pre}((x_1,(y_0,y_1))) &= K_0x_1 \wedge K_1(y_0 \vee y_1)\\
\texttt{pre}((x_1,(y_2,y_3))) &= K_0x_1 \wedge K_1(y_2 \vee y_3)\\
\texttt{pre}((x_2,(y_0,y_1))) &= K_0x_2 \wedge K_1(y_0 \vee y_1)\\
\texttt{pre}((x_2,(y_2,y_3))) &= K_0x_2 \wedge K_1(y_2 \vee y_3)\\
\texttt{pre}((x_3,(y_0,y_1))) &= K_0x_3 \wedge K_1(y_0 \vee y_1)\\
\texttt{pre}((x_3,(y_2,y_3))) &= K_0x_3 \wedge K_1(y_2 \vee y_3)
\end{aligned}$$

$$\begin{aligned}
\texttt{AM}_2: \qquad \mathsf{S}_{\texttt{AM}_2} &= \{y_2,y_3\}\\
\sim_{p_i} &= \mathit{loops}\\
\texttt{pre}(y_2) &= K_1(x_0 \wedge y_2)\\
\texttt{pre}(y_3) &= K_1(x_0 \wedge y_3)
\end{aligned}$$

The resulting composed action model is $\texttt{CAM}_2$:

$$\begin{aligned}
\mathtt{S}_{\mathtt{CAM}_2} &= \{(x_0,(y_0,y_1)),((x_0,(y_2,y_3)),y_2),\\
&\quad ((x_0,(y_2,y_3)),y_3),(x_1,(y_0,y_1)),\\
&\quad (x_1,(y_2,y_3)),(x_2,(y_0,y_1)),\\
&\quad (x_2,(y_2,y_3)),(x_3,(y_0,y_1)),\\
&\quad (x_3,(y_2,y_3))\}\\
\sim_{p_i} &= \mathit{loops}\\
\mathtt{pre}((x_0,(y_0,y_1))) &= K_0x_0 \wedge K_1(y_0 \vee y_1)\\
\mathtt{pre}((x_0,(y_2,y_3)),y_2) &= K_0x_0 \wedge K_1y_2\\
\mathtt{pre}((x_0,(y_2,y_3)),y_3) &= K_0x_0 \wedge K_1y_3\\
\mathtt{pre}((x_1,(y_0,y_1))) &= K_0x_1 \wedge K_1(y_0 \vee y_1)\\
\mathtt{pre}((x_1,(y_2,y_3))) &= K_0x_1 \wedge K_1(y_2 \vee y_3)\\
\mathtt{pre}((x_2,(y_0,y_1))) &= K_0x_2 \wedge K_1(y_0 \vee y_1)\\
\mathtt{pre}((x_2,(y_2,y_3))) &= K_0x_2 \wedge K_1(y_2 \vee y_3)\\
\mathtt{pre}((x_3,(y_0,y_1))) &= K_0x_3 \wedge K_1(y_0 \vee y_1)\\
\mathtt{pre}((x_3,(y_2,y_3))) &= K_0x_3 \wedge K_1(y_2 \vee y_3)
\end{aligned}$$

The corresponding protocol tree $\mathcal{T}_{\mathcal{B}}$ is given in Figure 4.5.

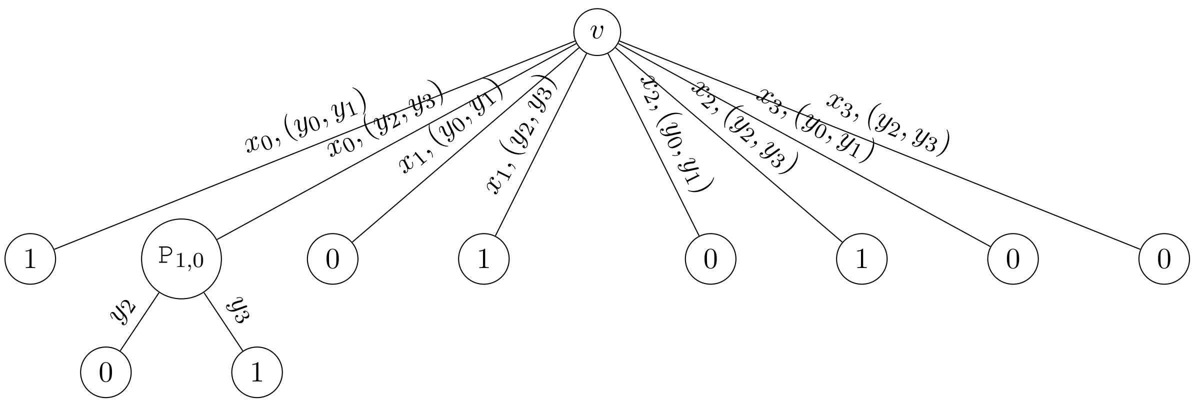


Figure 4.5: The protocol tree $\mathcal{T}_{\mathcal{B}}$ for function f defined in Figure 4.2.

The rectangles $\mathcal{R}^f_{\mathcal{B}}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{B}}$ are depicted in Figure 4.6.

We see that there are 9 completely separated actions in $\mathtt{CAM}_2$, corresponding to 9 leaves in the protocol tree. Here one can argue that in round 1 p_0 distinguishes between 4 different values, thus sends at least 2 bits in round 1, while p_1 distinguishes between (y_0, y_1) and (y_2, y_3), thus sends at least 1 bit in round 1, which adds up to a total number of at least 3 bits in round 1. In round 2 only p_1 sends a single bit, thus in the worst-case the total number of sent bits in $\mathcal{B}$ is at least 4. Since communication is reliable, at least 4 bits are received. Note that also here at least $log_2(9) \leq 4$ bit are received, thus there seems to be a strong connection between the minimum number of received bits and the number of partitions in the composed action model. In the following, we will establish and prove this connection.

	y_0	y_1	y_2	y_3
x_0	1	1	0	1
x_1	0	0	1	1
x_2	0	0	1	1
x_3	0	0	0	0

Figure 4.6: The rectangles $\mathcal{R}^f_{\mathcal{B}}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{B}}$ in Figure 4.5.

Example 4.3.3. This example demonstrates the consequences of duplicated messages.

We use the same scenario as in Example 4.3.1 with an additional duplicated action x_0' in the first round. Thus the first round for algorithm $\mathcal{A}'$ is as follows:

$$
\begin{aligned}
\texttt{AM}_1\text{:} \qquad \texttt{S}_{\texttt{AM}_1} &= \{(x_0), (x_0'), (x_1, x_2, x_3)\} \\
\sim_{p_i} &= \textit{loops} \\
\texttt{pre}(x_0) &= K_0 x_0 \\
\texttt{pre}(x_0') &= K_0 x_0 \\
\texttt{pre}((x_1, x_2, x_3)) &= K_0(x_1 \vee x_2 \vee x_3)
\end{aligned}
$$

The resulting composed action model is $\texttt{CAM}_3$:

$$
\begin{aligned}
\texttt{S}_{\texttt{CAM}_3} &= \{(x_0, (y_0, y_1, y_3)), (x_0, y_2), \\
&\quad (x_0', (y_0, y_1, y_3)), (x_0', y_2), \\
&\quad ((x_1, x_2, x_3), (y_0, y_1)), \\
&\quad (((x_1, x_2, x_3), (y_2, y_3)), x_3), \\
&\quad (((x_1, x_2, x_3), (y_2, y_3)), (x_1, x_2))\} \\
\sim_{p_i} &= \textit{loops} \\
\texttt{pre}((x_0, (y_0, y_1, y_2))) &= K_0 x_0 \wedge K_1(y_0 \vee y_1 \vee y_3) \\
\texttt{pre}((x_0, y_2)) &= K_0 x_0 \wedge K_1 y_2 \\
\texttt{pre}((x_0', (y_0, y_1, y_2))) &= K_0 x_0 \wedge K_1(y_0 \vee y_1 \vee y_3) \\
\texttt{pre}((x_0', y_2)) &= K_0 x_0 \wedge K_1 y_2 \\
\texttt{pre}((x_1, x_2, x_3), (y_0, y_1)) &= K_0(x_1 \vee x_2 \vee x_3) \wedge K_1(y_0 \vee y_1) \\
\texttt{pre}(((x_1, x_2, x_3), (y_2, y_3)), x_3) &= K_0 x_3 \wedge K_1(y_2 \vee y_3) \\
\texttt{pre}(((x_1, x_2, x_3), (y_2, y_3)), (x_1, x_2)) &= K_0(x_1 \vee x_2) \wedge K_1(y_2 \vee y_3)
\end{aligned}
$$

The corresponding protocol tree $\mathcal{T}_{\mathcal{A}'}$ is given in Figure 4.7.

The rectangles $\mathcal{R}^f_{\mathcal{A}'}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{A}}$ are depicted in Figure 4.8.

In Figure 4.7, the paths through node $P_{1,2}$ are duplicates of the paths through $P_{1,1}$. These duplicates are caused by the action x_0', which is a duplicate of x_0. Furthermore, we see that the two leaves colored in red (resp. blue) *both* correspond to a *single* rectangle in Figure 4.8.

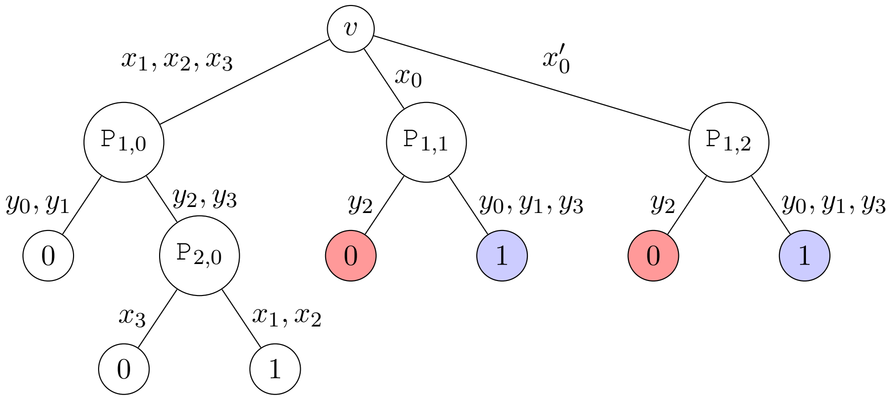


Figure 4.7: The protocol tree $\mathcal{T}_{\mathcal{A}'}$ for function f defined in Figure 4.2. The colored leaves correspond to the colored rectangles in Figure 4.8. We see that two leaves correspond to a single rectangle in $\mathcal{A}'$ which defines duplicated messages.

	y_0	y_1	y_2	y_3
x_0	1	1	0	1
x_1	0	0	1	1
x_2	0	0	1	1
x_3	0	0	0	0

Figure 4.8: Two of the rectangles $\mathcal{R}^f_{\mathcal{A}'}$ corresponding to the protocol tree $\mathcal{T}_{\mathcal{A}'}$ in Figure 4.7. We see that two leaves of $\mathcal{T}_{\mathcal{A}'}$ correspond to each of the colored rectangles.

4.3.3 Main Results

According to the previous section, action models can be related to protocol trees and rectangles of a function f. In this section, we will prove a lower bound on the number of bits received by the processes during the worst-case execution of an algorithm $\mathcal{A}$ for computing a function $f(x, y)$. This section is again restricted to the 2-process case, in which process p_0 resp. p_1 has input x resp. y. In the following we denote the maximum number of bits received by p_i by $\mathcal{D}_i$, and $\mathcal{R}^f_{\mathcal{A}}$ the rectangles corresponding to the protocol tree $\mathcal{T}_{\mathcal{A}}$.

Lemma 4.3.3. *p_0 receives at least $log_2(R_X) \leq \mathcal{D}_0$ bits during the worst-case execution of $\mathcal{A}$ solving $f(x, y)$, with R_X denoting the maximum number of rectangles in any row of M_f, the matrix defining the function f.*

Proof. Assume the input of p_0 is some fixed value $x_i \in X$ such that there are $R_{x_i} = R_X$ rectangles in the row of M_f corresponding to x_i. Suppose in contradiction that $log_2(R_X) > \mathcal{D}_0$. Then $R_X > 2^{\mathcal{D}_0}$. It is easy to see that, by receiving $\mathcal{D}_0$ bits, p_0 can distinguish at most $2^{\mathcal{D}_0}$ rectangles in the row of x_i. Since $R_X > 2^{\mathcal{D}_0}$ there is a row in $\mathcal{R}^f_{\mathcal{A}}$, such that there are at least two rectangles R_0 and R_1 which cannot be distinguished by p_0.

The rectangles R_0 and R_1 correspond to some leaves ℓ_0 and ℓ_1 of $\mathcal{T}_{\mathcal{A}}$, which again correspond to some partitions P_0 and P_1 of the composed action model for $\mathcal{A}$. Thus, if p_0 cannot distinguish R_0 and R_1, p_0 is also not able to distinguish the corresponding partitions P_0 and P_1.

As in Lemma 4.2.1, assume that applying $(\mathtt{M}, \mathtt{s})$ to epistemic state (M, s) results in (M', s'), i.e.,:

- $s_0 \sim_{p_0} s_1$ in epistemic model M, which follows from Lemma 4.3.2 in conjunction with the fact that p_0 is the only process that can be uncertain about the initial epistemic state (x_i, y_i), as x_i is fixed.
- $\mathtt{s}_0 \in P_0$, $\mathtt{s}_1 \in P_1$ in action model $\mathtt{M}$ applicable to s_0 respectively s_1 (P_0 and P_1 indistinguishable by p_0), and
- $(s_0, \mathtt{s}_0) \not\sim_{p_0} (s_1, \mathtt{s}_1)$ in epistemic model M'.

If p_0 cannot distinguish between the states in partitions P_0 and P_1, then, since $s_0 \sim_{p_0} s_1$, $(s_0, \mathtt{s}_0) \sim_{p_0} (s_1, \mathtt{s}_1)$ in epistemic model M' contradicting the result of applying $(\mathtt{M}, \mathtt{s})$ to (M, s). □

Lemma 4.3.4. *p_1 receives at least $log_2(R_Y) \leq \mathcal{D}_1$ bits during the worst-case execution of $\mathcal{A}$ solving $f(x, y)$, with R_Y denoting the maximum number of rectangles in any column of M_f, the matrix defining the function f.*

Proof. Analogous to the proof of Lemma 4.3.3. □

The following Theorem 4.3.1 will establish a lower bound on the cost $\mathcal{D}_{\mathcal{A}}$ of an algorithm $\mathcal{A}$ using the number of partitions $t_{\mathtt{CAM}_m}$ of the composed action model $\mathtt{CAM}_m$, which can be seen as the action model analogon of Corollary 2.2.1.

Theorem 4.3.1. *During any execution of an algorithm $\mathcal{A}$, which computes $f(x, y)$ in m rounds,*

$$t_{CAM_m} \geq t$$

where t_{CAM_m} is the number of partitions of the composed action model of $\mathcal{A}$ after m rounds, and t is the number of monochromatic rectangles of $f(x, y)$. Moreover, at least $log_2\ t_{CAM_m}$ bits must be received in the system during the execution of $\mathcal{A}$, i.e.,

$$log_2\ t_{CAM_m} \leq \mathcal{D}_{\mathcal{A}}.$$

Proof. Suppose $t_{\texttt{CAM}_m} < t$, i.e., there are less leaves in $\mathcal{T}_\mathcal{A}$ than there are monochromatic rectangles. Then, there are two rectangles $\mathcal{R}_1, \mathcal{R}_2$ that lead to the same leaf. However, this contradicts Lemma 4.3.2, as every leaf corresponds to a single rectangle.

By Lemma 4.2.2 a lower bound for the worst-case cost $\mathcal{D}^a_\mathcal{A}$, regarding a process a, is $log_2\ t^a_{\texttt{CAM}_m} \leq \mathcal{D}^a_\mathcal{A}$, with $t^a_{\texttt{CAM}_m}$ the number of partitions of $\texttt{CAM}_m$ regarding some process a. Additionally each process has to be able to distinguish all the partitions of $\texttt{CAM}_m$, else the result of $f(x, y)$ would not be common knowledge. Thus $t^a_{\texttt{CAM}_m} = t_{\texttt{CAM}_m}$ and furthermore $log_2\ t_{\texttt{CAM}_m} \leq \mathcal{D}^a_\mathcal{A}$. Trivially, $\mathcal{D}^a_\mathcal{A} \leq \mathcal{D}_\mathcal{A}$.

Thus we can conclude that $log_2\ t \leq log_2\ t_{\texttt{CAM}_m} \leq \mathcal{D}_\mathcal{A}$. □

Note that Theorem 4.3.1 does not state anything about the worst-case cost $\mathcal{D}_f$ for computing $f(x, y)$, but only establishes a lower bound for the worst-case cost $\mathcal{D}_\mathcal{A}$ for some execution of algorithm $\mathcal{A}$ computing $f(x, y)$.

4.4 Application on consensus in directed dynamic networks

We will now apply the method developed in Section 4.3.3 to get a lower bound on the bit complexity for the consensus problem. The considered scenario consists of two processes connected via a directed dynamic network. The message adversary we consider is $MA_{\leftrightarrow^2}$, which may generate any graph sequence from $\{\leftarrow, \leftrightarrow, \rightarrow\}^* \setminus \{\leftrightarrow^2\}$. The problem here is that consensus does not specify a unique function: Validity only fixes the outcome for all inputs being the same, but not in the remaining cases. Agreement, on the other hand, only requires the outputs at p_0 and p_1 to be the same. As a consequence, the actual output computed here may depend on the algorithm and on the particular graph sequence σ, chosen by the message adversary.

In a directed dynamic network, we can consider consensus as a function $f(x, y, \sigma)$, thus the result depends on the input of p_0 (x) and p_1 (y), but also on the graph sequence σ chosen by MA. An example for the message adversary $MA_{\leftrightarrow^2}$, which may generate any graph sequence except $(\leftrightarrow, \leftrightarrow)$, is given in Figure 4.9. Whereas there are multiple correct ways for defining $f_{\leftrightarrow^2}(x, y, \sigma)$, they need to satisfy $f_{\leftrightarrow^2}(v, v, \sigma) = v$ by validity. Moreover, by a simple indistinguishability argument, the result on $\sigma = \leftarrow^* \in MA_{\leftrightarrow^2}$ must be y, while it is x on $\sigma = \rightarrow^* \in MA_{\leftrightarrow^2}$ (for a detailed explanation see Chapter 5).

There is an algorithm $\mathcal{A}$ solving consensus for each $MA_{\leftrightarrow^k}$, $k \geq 2$, where every process receives at least 2 bit. In this algorithm, p_0 and p_1 both send their initial value x resp. y in round 1 to each other. In each round $r \geq 2$, each process sends the last value it received (or its initial value if it did not receive any value up to round r). Once a process received the same value it sent or does not receive a value in some round $r \geq 2$, it decides the last sent value and terminates. This algorithm $\mathcal{A}$, also solving consensus for the eventual message adversary $MA_{\leftrightarrow^*}$, is explained in more detail in Section 5.4.

	$y=0$		$y=1$	
$x =$	1	0	0	1
$\leftarrow, \leftarrow$	0	0	1	1
$\leftarrow, \leftrightarrow$	0	0	1	1
$\leftarrow, \rightarrow$	0	0	1	1
$\leftrightarrow, \rightarrow$	0	0	1	1
$\leftrightarrow, \leftarrow$	1	0	0	1
$\rightarrow, \leftarrow$	1	0	0	1
$\rightarrow, \leftrightarrow$	1	0	0	1
$\rightarrow, \rightarrow$	1	0	0	1

Figure 4.9: A function $f_{\leftrightarrow^2}(x, y, \sigma)$ defining the outcome of consensus in the 2 process case with adversary $MA_{\leftrightarrow^2}$.

In the following, the action models for this algorithm $\mathcal{A}$ are given:
The single actions are given in the form $(v_0, v_1, \mathcal{G})$, with $v_i \in \{0,1\}$ the value considered by p_i (either p_i's initial value, or the last value it received) and $\mathcal{G} \in \{\leftarrow, \leftrightarrow, \rightarrow\}$ the graph chosen by the message adversary in the current round.

$$\begin{aligned}
\texttt{AM}_1: \qquad \texttt{S}_{AM_1} &= \{(0,0,\rightarrow), \ldots, (1,1,\rightarrow), (0,0,\leftrightarrow), \ldots, (1,1,\leftrightarrow),\\
&\quad (0,0,\leftarrow), \ldots, (1,1\leftarrow)\}\\
\sim_{p_0} &= \{((0,1,\rightarrow),(0,0,\rightarrow)), ((0,0,\leftrightarrow),(0,0,\leftarrow)),\\
&\quad ((1,0,\leftarrow),(1,0,\leftrightarrow)), ((1,0,\rightarrow),(1,1,\rightarrow)),\\
&\quad ((1,1,\leftrightarrow),(1,1,\leftarrow)), ((0,1,\leftarrow),(0,1,\leftrightarrow))\}\\
\sim_{p_1} &= \{((0,0,\rightarrow),(0,0,\leftrightarrow)), ((0,0,\leftarrow),(1,0,\leftarrow)),\\
&\quad ((1,0,\leftrightarrow),(1,0,\rightarrow)), ((1,1,\rightarrow),(1,1,\leftrightarrow)),\\
&\quad ((1,1,\leftarrow),(0,1,\leftarrow)), ((0,1,\leftrightarrow),(0,1,\rightarrow))\}\\
\texttt{pre}((v_0,v_1,\rightarrow)) &= K_0v_0 \wedge K_1v_1\\
\texttt{pre}((v_0,v_1,\leftarrow)) &= K_0v_0 \wedge K_1v_1\\
\texttt{pre}((v_0,v_1,\leftrightarrow)) &= K_0v_0 \wedge K_1v_1
\end{aligned}$$

$\texttt{AM}_2$ is identical to $\texttt{AM}_1$, except that the precondition function must explicitly omit combining $(v_i, v_j, \leftrightarrow) \in \texttt{AM}_1$ and $(v_i', v_j', \leftrightarrow) \in \texttt{AM}_2$. The graphical representation of $\texttt{AM}_i$ for this example is given in Figure 4.10.

The resulting composed action model after two rounds, $\texttt{CAM}_2 = (\texttt{AM}_1; \texttt{AM}_2)$, is depicted in Figure 4.11 and has 32 actions split up in 2 partitions. Recall that the composition must explicitly omit combining $(., ., \leftrightarrow) \in \texttt{AM}_1$ and $(., ., \leftrightarrow) \in \texttt{AM}_2$. For readability, the composed actions are labeled in Table 4.1.

$$\begin{tikzcd}
(1,0,\rightarrow) \arrow[r, dash, "p_0"] \arrow[d, dash, "p_1"'] & (1,1,\rightarrow) \arrow[r, dash, "p_1"] & (1,1,\leftrightarrow) \arrow[r, dash, "p_0"] & (1,1,\leftarrow) \arrow[r, dash, "p_1"] & (0,1,\leftarrow) \arrow[r, dash, "p_0"] & (0,1,\leftrightarrow) \arrow[d, dash, "p_0"] \\
(1,0,\leftrightarrow) \arrow[r, dash, "p_0"] & (1,0,\leftarrow) \arrow[r, dash, "p_1"] & (0,0,\leftarrow) \arrow[r, dash, "p_0"] & (0,0,\leftrightarrow) \arrow[r, dash, "p_1"] & (0,0,\rightarrow) \arrow[r, dash, "p_0"] & (0,1,\rightarrow)
\end{tikzcd}$$

Figure 4.10: The graphical representation of $\mathtt{AM}_i$.

1	$(0,0,\rightarrow),(0,0,\rightarrow)$	17	$(1,0,\rightarrow),(1,1,\rightarrow)$
2	$(0,0,\rightarrow),(0,0,\leftrightarrow)$	18	$(1,0,\rightarrow),(1,1,\leftrightarrow)$
3	$(0,0,\rightarrow),(0,0,\leftarrow)$	19	$(1,0,\rightarrow),(1,1,\leftarrow)$
4	$(0,0,\leftrightarrow),(0,0,\rightarrow)$	20	$(1,0,\leftrightarrow),(0,1,\rightarrow)$
5	$(0,0,\leftrightarrow),(0,0,\leftarrow)$	21	$(1,0,\leftrightarrow),(0,1,\leftarrow)$
6	$(0,0,\leftarrow),(0,0,\rightarrow)$	22	$(1,0,\leftarrow),(0,0,\rightarrow)$
7	$(0,0,\leftarrow),(0,0,\leftrightarrow)$	23	$(1,0,\leftarrow),(0,0,\leftrightarrow)$
8	$(0,0,\leftarrow),(0,0,\leftarrow)$	24	$(1,0,\leftarrow),(0,0,\leftarrow)$
9	$(0,1,\rightarrow),(0,0,\rightarrow)$	25	$(1,1,\rightarrow),(1,1,\rightarrow)$
10	$(0,1,\rightarrow),(0,0,\leftrightarrow)$	26	$(1,1,\rightarrow),(1,1,\leftrightarrow)$
11	$(0,1,\rightarrow),(0,0,\leftarrow)$	27	$(1,1,\rightarrow),(1,1,\leftarrow)$
12	$(0,1,\leftrightarrow),(1,0,\rightarrow)$	28	$(1,1,\leftrightarrow),(1,1,\rightarrow)$
13	$(0,1,\leftrightarrow),(1,0,\leftarrow)$	29	$(1,1,\leftrightarrow),(1,1,\leftarrow)$
14	$(0,1,\leftarrow),(1,1,\rightarrow)$	30	$(1,1,\leftarrow),(1,1,\rightarrow)$
15	$(0,1,\leftarrow),(1,1,\leftrightarrow)$	31	$(1,1,\leftarrow),(1,1,\leftrightarrow)$
16	$(0,1,\leftarrow),(1,1,\leftarrow)$	32	$(1,1,\leftarrow),(1,1,\leftarrow)$

Table 4.1: Composed actions of $\mathtt{CAM}_2 = (\mathtt{AM}_1; \mathtt{AM}_2)$ in Figure 4.11.

The corresponding protocol tree $\mathcal{T}_{cons}$ is depicted in Figure 4.12.

We observe that $t_{\mathtt{CAM}_2} = 2$ here. Recalling Theorem 4.3.1, the lower bound for $\mathcal{D}_{\mathcal{A}}$ in directed dynamic networks on adversary $MA_{\leftrightarrow^2}$ hence satisfies $\mathcal{D}_{\mathcal{A}} \geq 1$, which is trivially a lower bound on the number of received bits in the system.

Relation to [Yao79]

2-process consensus in directed dynamic networks can also be directly related to the approach of [Yao79]. Actually, there are two possibilities to define rectangles in our setting:

Possibility I The obvious approach is to extend the original definition of rectangles to also incorporate σ:

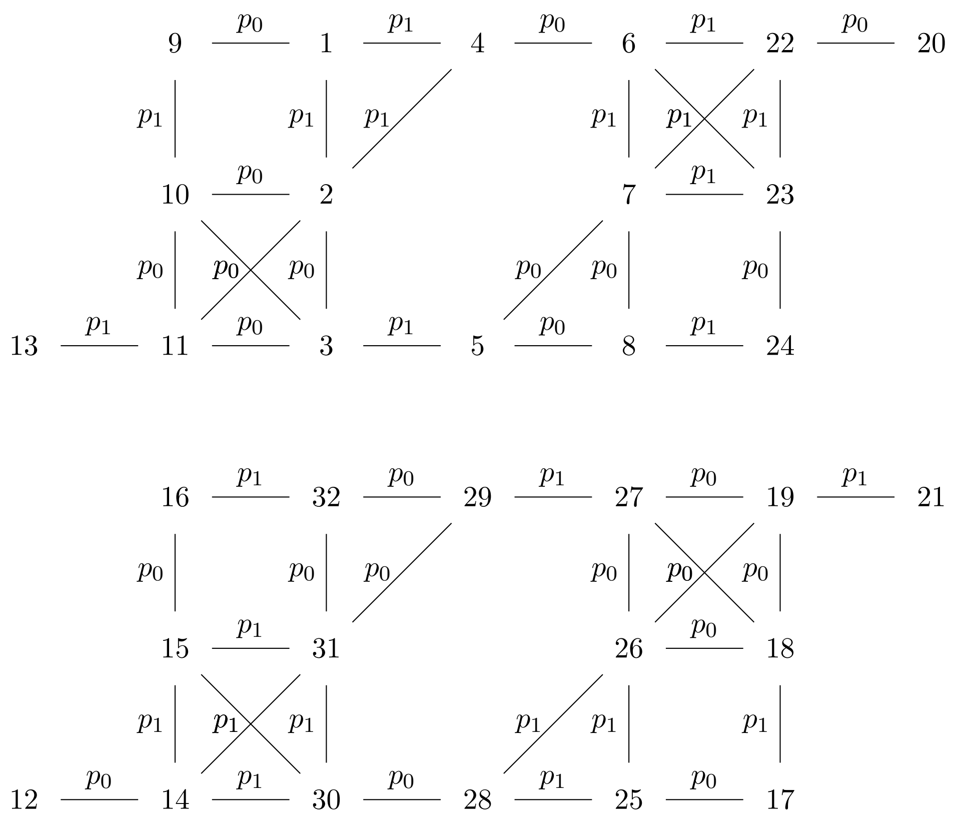


Figure 4.11: The graphical representation of $\texttt{CAM}_2$. The actions in the upper partition all lead to the decision 0, the ones in the lower partition to a decision 1. They hence correspond to monochromatic rectangles.

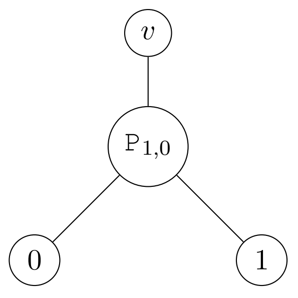


Figure 4.12: The protocol tree $\mathcal{T}_{cons}$ corresponding to the action model $\texttt{CAM}_2 = (\texttt{AM}_1; \texttt{AM}_2)$ defined in Figure 4.10 that solve consensus by computing the function $f_{\leftrightarrow^2}$ given in Figure 4.9. $P_{1,0}$ represents the single partition of $\texttt{AM}_1$ representing Figure 4.10, 0 resp. 1 the two partitions of Figure 4.11.

Definition 4.4.1. A rectangle R of a function $f(x, y, \sigma)$ is defined as a set such that: (x_1, y_1, σ_1), $(x_2, y_2, \sigma_2) \in R$, iff $(x_1, y_1, \sigma_2), (x_1, y_2, \sigma_1), (x_1, y_2, \sigma_2), (x_2, y_1, \sigma_1), (x_2, y_1, \sigma_2), (x_2, y_2, \sigma_1) \in R$.

Using Definition 4.4.1, the minimum number of monochromatic rectangles of $f_{\leftrightarrow^2}$ turns out to be 4: The partitioning of $f_{\leftrightarrow^2}$ is shown in Figure 4.13. The number of leaves of $\mathcal{T}_{cons}$ (which determines the communication complexity (1 bit)) is 2 and hence smaller than the number of rectangles this definition leads to. It may be possible to trace this contradiction to the fact that σ is not a purely local input to a single process, as all processes can infer something about σ in some cases. For example, if $\sigma = \rightarrow$, then p_1 can infer that $\sigma \neq \leftarrow$ even though only p_0 knows $\sigma = \rightarrow$ exactly.

	$y = 0$		$y = 1$	
$x =$	1	0	0	1
$\leftarrow, \leftarrow$	0	0	1	1
$\leftarrow, \leftrightarrow$	0	0	1	1
$\leftarrow, \rightarrow$	0	0	1	1
$\leftrightarrow, \rightarrow$	0	0	1	1
$\leftrightarrow, \leftarrow$	1	0	0	1
$\rightarrow, \leftarrow$	1	0	0	1
$\rightarrow, \leftrightarrow$	1	0	0	1
$\rightarrow, \rightarrow$	1	0	0	1

Figure 4.13: The rectangle partitioning of the function $f_{\leftrightarrow^2}(x, y, \sigma)$ defining the outcome of consensus in the 2 process case with adversary $MA_{\leftrightarrow^2}$ using Definition 4.4.1.

Possibility II An alternative method, which does not suffer from the problem described above, is to use the original rectangle definition by Yao. To get rid of the indeterminism in the consensus specification, i.e., the dependency of the computed function on the algorithm and the graph sequence, we partition the set of computed functions into all the possibilities allowed by the consensus specification. Every partition is then an independent distributed function computation that can be treated with Yao's approach. The worst-case communication complexity of the original algorithm is determined by the maximum worst-case communication complexity of the individual distributed function computations.

We denote the set of graph sequences for which the input of x and y results in a decision on z by P^z_{xy}. For example, P^0_{01} of the function $f_{\leftrightarrow^2}$ given in Figure 4.9 is $\{(\leftrightarrow, \leftarrow), (\rightarrow, \leftarrow), (\rightarrow, \leftrightarrow), (\rightarrow, \rightarrow)\}$. Enumerating all the possible sets P^z_{xy}, we get: P^0_{00}, P^1_{00}, P^0_{01}, P^1_{01}, P^0_{10}, P^1_{10}, P^0_{11} and P^1_{11}. Clearly P^1_{00} and P^0_{11} are empty due to validity. Since the decision in P^0_{00} and P^1_{11} is given by validity and, thus, independent of the algorithm, there is no need to explicitly consider these two sets of graph sequences. These eliminations leave only the non-trivial sets P^0_{01}, P^1_{01}, P^0_{10} and P^1_{10}. We now use those sets to build multiple two-dimensional function matrices depending solely on x and y by taking all the possible intersections of those sets. For $f_{\leftrightarrow^2}(x, y, \sigma)$ given in Figure 4.9, this results in $P^0_{01} \cap P^0_{10} = \emptyset$, $P^0_{01} \cap P^1_{10} = \{(\leftrightarrow, \leftarrow), (\rightarrow, \leftarrow), (\rightarrow, \leftrightarrow), (\rightarrow, \rightarrow)\}$, $P^1_{01} \cap P^0_{10} = \{(\leftarrow, \leftarrow), (\leftarrow, \leftrightarrow), (\leftarrow, \rightarrow), (\leftrightarrow, \rightarrow)\}$ and $P^1_{01} \cap P^1_{10} = \emptyset$ (we omit $P^0_{xy} \cap P^1_{xy}$ here since those are trivially empty sets). Each such intersection specifies the graph sequences for which a unique two-dimensional function matrix in x and y applies.

For example, $P^0_{01} \cap P^1_{10} = \{(\leftrightarrow, \leftarrow), (\rightarrow, \leftarrow), (\rightarrow, \leftrightarrow), (\rightarrow, \rightarrow)\}$ are the graph sequences for which $f(x, y) = x$ applies.

As those intersections obviously are disjoint, the adversaries choice of the graph sequence σ also determines the actual function $f'(x, y)$ that has to be calculated.

Figure 4.14 shows the consensus algorithm $\mathcal{A}$ of Figure 4.9 partitioned into those two-dimensional matrices. The trivial partitions for P^0_{00} and P^1_{11} are marked blue, while the partitions relevant for the creation of the non-trivial functions are marked orange resp. red. Intersecting, e.g., P^0_{10} and P^1_{01}, we get function $f'_{\leftrightarrow^2}$ (orange). The results for $f'_{\leftrightarrow^2}(0, 0)$ and $f'_{\leftrightarrow^2}(1, 1)$ are given by Validity, while the other two results are defined by the used partitions, e.g., P^0_{10} corresponds to $f'_{\leftrightarrow^2}(1, 0) = 0$ in the lower left corner. In order to analyze the communication complexity of $\mathcal{A}$, we can analyze the involved functions in isolation, and take the maximum.

Since we are now back at the usage of two-dimensional function matrices depending solely on local input values, we can use the original definition of rectangles by Yao, and hence all the techniques of Section 4.3. Note that, in each of those matrices, the number of monochromatic rectangles t is 2.

As the two-dimensional functions $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$ are valid for specific subsets of graph sequences only, we only need to consider composed actions which correspond to these graph sequences. E.g., looking at $f'_{\leftrightarrow^2}$, we need to consider only the graph sequences in $\{(\leftarrow, \leftarrow), (\leftarrow, \leftrightarrow), (\leftarrow, \rightarrow), (\leftrightarrow, \rightarrow)\}$. Figure 4.15 again shows the graphical representation of $\mathtt{CAM}_2$ based on Figure 4.11. The composed actions corresponding to $f'_{\leftrightarrow^2}$ are marked in orange, while the composed actions corresponding to $f''_{\leftrightarrow^2}$ are marked in red. We see, that in both cases the Composed Action Model corresponding to the considered function has two partitions, regarding $\mathtt{AM}_1$, the ability to restrict the composed Action Model $\mathtt{CAM}_2$ to the partition relevant for, say, $f'_{\leftrightarrow^2}$, does not lead to a restriction of Figure 4.10, hence P_{01} remains the same. Thus the protocol tree for the two-dimensional functions is the same as in Figure 4.12, so $t_{\mathtt{CAM}_2} = 2$ again.

This is because there is at least one process p_0 or p_1, which does not know after the first round using this algorithm whether it will compute $f'_{\leftrightarrow^2}$ or $f''_{\leftrightarrow^2}$. Since $t_{\mathtt{CAM}_2} = t = 2$, for either $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$, Theorem 4.3.1 shows that the algorithm is optimal.

4.5 Summary and Discussion of our Findings

In Section 4.2, we first established a lower bound on the number of bits received by a single process p_0 when applying some Action Model. We found that the number of bits received by a single process is tightly coupled to the number of partitions of the Action Model. Lemma 4.2.1 resp. Lemma 4.2.2 provide a lower bound depending on the availability resp. non-availability of *communication by time.* As we cannot assume that the worst action (in terms of communication) for process p_0 is the same action as the worst action for process p_1, however, the sum of the lower bounds for all processes does not lead to an overall lower bound on communication complexity for applying the Action Model.

Following [Yao79], we restrict ourselves to scenarios similar to his in Section 4.3: We concentrate on the symmetric computation of a function $f(x, y)$, x resp. y being the inputs of two processes p_0 resp. p_1, and use a synchronous model, in which each process can send arbitrarily many bits in each asynchronous round in a model with guaranteed message delivery. Once an algorithm for computing $f(x, y)$ terminates, the result shall be commonly known among the processes p_0 and p_1. Similar to [Yao79], we defined a protocol tree $\mathcal{T}_{\mathcal{A}}$ on the basis of the Action Models $\{\mathtt{AM}_1, \dots, \mathtt{AM}_m\}$ defined by some algorithm $\mathcal{A}$ in Section 4.3, where m is the round in which $\mathcal{A}$ terminates. Using these Action Models, we compute the Composed Action Model $\mathtt{CAM}_k = (\mathtt{CAM}_{k-1}; \mathtt{AM}_k)$ for any round k, $k \leq m$. The nodes on level k of the protocol tree correspond to the partitions of the Composed Action Model $\mathtt{CAM}_k$ for round k. Using a relation between the monochromatic rectangles of the function matrix M_f and $\mathcal{T}_{\mathcal{A}}$'s leaves, which correspond to the partitions of $\mathtt{CAM}_m$, we established Theorem 4.3.1, which gives a (possibly conservative) lower bound on the overall (system-wide) communication complexity for any given function computation algorithm.

We applied our methods for deriving lower bounds on the communication complexity for the consensus problem in directed dynamic networks in Section 4.4. To do so, we considered a scenario consisting of two processes connected via a directed dynamic network. As the consensus result does not only depend on the initial values x and y, but also on the graph sequence σ chosen by a message adversary, we cannot interpret the problem just as the computation of a 2-dimensional function. Thus, we considered consensus in directed dynamic networks as the computation of a 3-dimensional function $f(x, y, \sigma)$.

Since [Yao79] does not deal with 3-dimensional functions, we also proposed two possibilities to handle this problem. The first approach just generalizes the definition of rectangles to also incorporate σ. As the latter is not purely local to one of the processes, however, this does not lead to a communication complexity lower bound, even for the simple message adversary $MA_{\leftrightarrow^2}$. The other approach exploits the structure of the message adversary and the consensus problem, dividing the 3-dimensional function matrix into multiple 2-dimensional matrices, which are valid for disjoint sets of graph sequences. This second approach leads to the correct lower bound for the simple message adversary

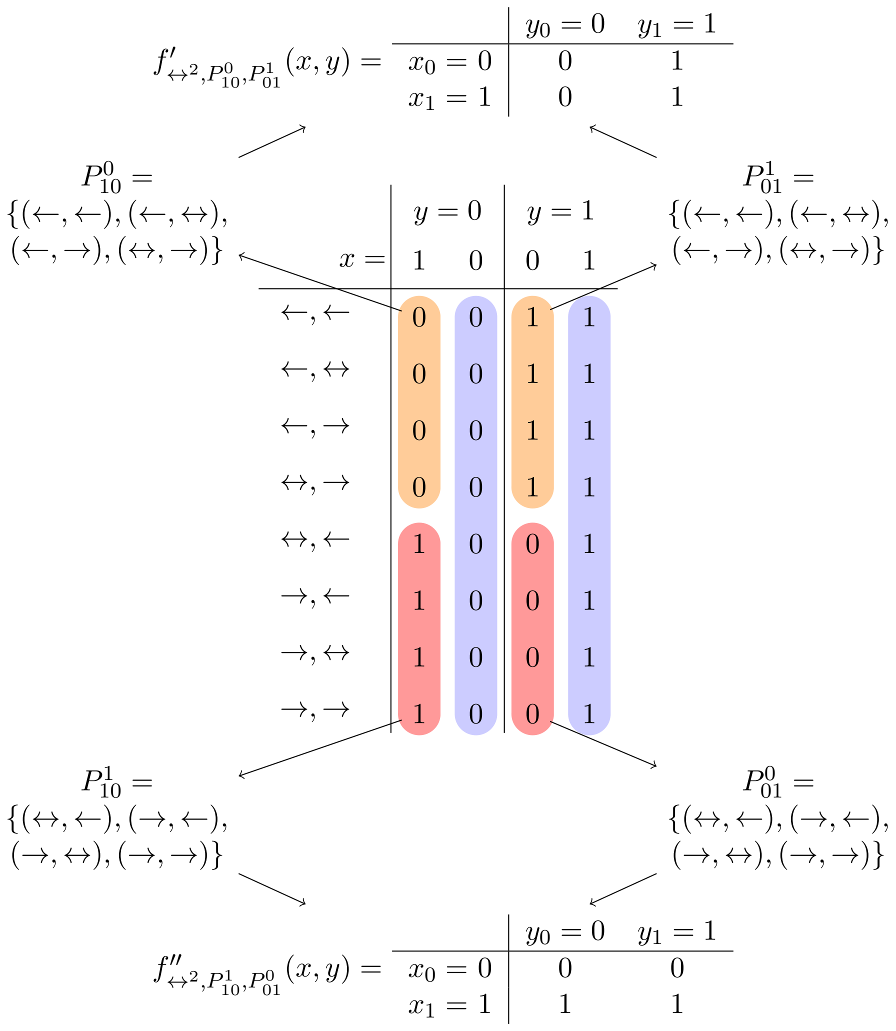
$MA_{\leftrightarrow^2}$. However, it remains to be proven that this approach is universally applicable in directed dynamic networks.

The main shortcomings of our approach are the following:

1. Besides the restrictions inherited from the basic model, we can only apply our lower-bound results on algorithms, for which there exists a maximum number of rounds until termination. Consequently we can only apply the deduced method to finite message adversaries, since there is no algorithm with a maximum number of rounds for eventual message adversaries. In Chapter 5, we discuss the impact of eventual message adversaries on the bit complexity and solvability of the consensus problem.

2. The restriction to 2 processes is extremely limiting. A main focus of future work has to be the extension of our method to systems of $n > 2$ processes.

3. Further research should also consider a definition of rectangles on multi-dimensional functions and functions with non-local input, like the graph sequence of directed dynamic networks, and may prove or disprove our second approach to do so.



$f'_{\leftrightarrow^2,P^0_{10},P^1_{01}}(x,y) =$
$y_0 = 0$ $y_1 = 1$
$x_0 = 0$ 0 1
$x_1 = 1$ 0 1
$P^0_{10} = \{(\leftarrow,\leftarrow),(\leftarrow,\leftrightarrow),(\leftarrow,\rightarrow),(\leftrightarrow,\rightarrow)\}$
$P^1_{01} = \{(\leftarrow,\leftarrow),(\leftarrow,\leftrightarrow),(\leftarrow,\rightarrow),(\leftrightarrow,\rightarrow)\}$
$y = 0$ $y = 1$
$x =$ 1 0 0 1
$\leftarrow,\leftarrow$ 0 0 1 1
$\leftarrow,\leftrightarrow$ 0 0 1 1
$\leftarrow,\rightarrow$ 0 0 1 1
$\leftrightarrow,\rightarrow$ 0 0 1 1
$\leftrightarrow,\leftarrow$ 1 0 0 1
$\rightarrow,\leftarrow$ 1 0 0 1
$\rightarrow,\leftrightarrow$ 1 0 0 1
$\rightarrow,\rightarrow$ 1 0 0 1
$P^1_{10} = \{(\leftrightarrow,\leftarrow),(\rightarrow,\leftarrow),(\rightarrow,\leftrightarrow),(\rightarrow,\rightarrow)\}$
$P^0_{01} = \{(\leftrightarrow,\leftarrow),(\rightarrow,\leftarrow),(\rightarrow,\leftrightarrow),(\rightarrow,\rightarrow)\}$
$f''_{\leftrightarrow^2,P^1_{10},P^0_{01}}(x,y) =$
$y_0 = 0$ $y_1 = 1$
$x_0 = 0$ 0 0
$x_1 = 1$ 1 1

Figure 4.14: Partitioning of three-dimensional function $f_{\leftrightarrow^2}$ into two two-dimensional functions $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$. The trivial partitions for P^0_{00} and P^1_{11} are marked blue, while the partitions relevant for the creation of the functions are marked orange resp. red. Combining, e.g., P^0_{10} and P^1_{01} we get function $f'_{\leftrightarrow^2}$ (orange). The results for $f'_{\leftrightarrow^2}(0,0)$ and $f'_{\leftrightarrow^2}(1,1)$ are given by Validity, while the other two results are defined by the used partitions, e.g., P^0_{10} corresponds to $f'_{\leftrightarrow^2}(1,0) = 0$ in the lower left corner. Since $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$ are two-dimensional, the original rectangle-definition by Yao and hence the results of Section 4.3 can be used again.

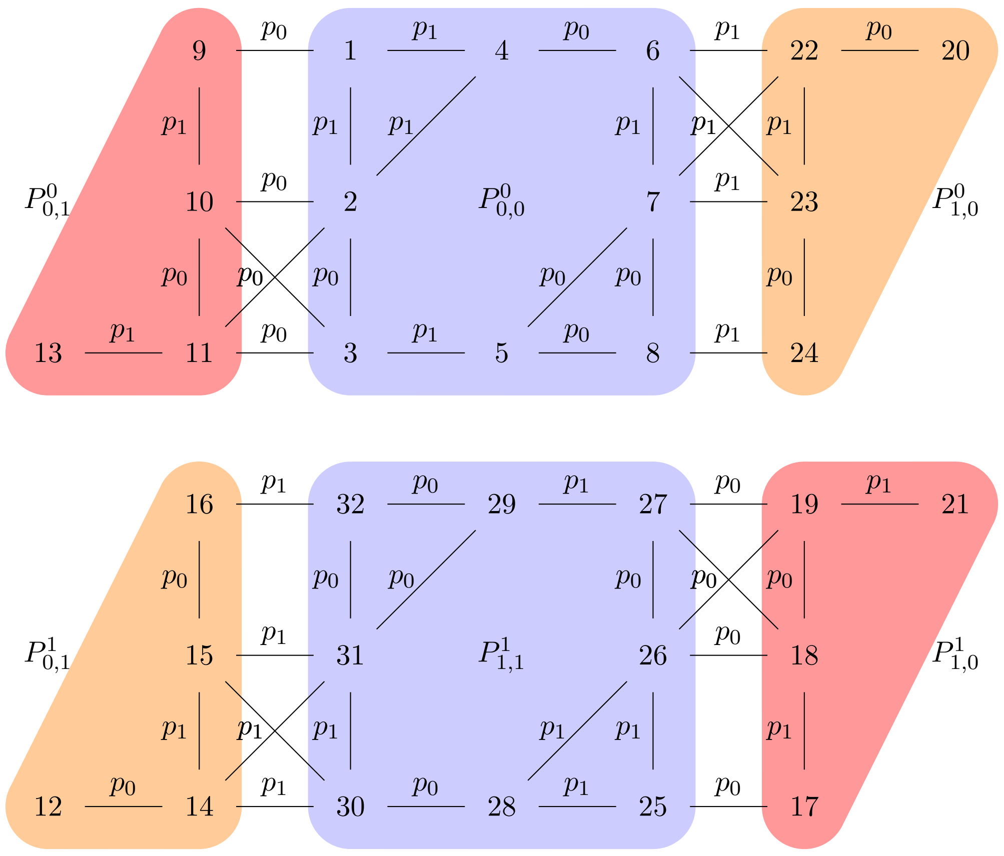


Figure 4.15: The graphical representation of $\texttt{CAM}_2$ for the two-dimensional functions $f'_{\leftrightarrow^2}$ (defined by $P^0_{1,0}$ and $P^1_{0,1}$ and depicted in orange) and $f''_{\leftrightarrow^2}$ (defined by $P^0_{0,1}$ and $P^1_{1,0}$ and depicted in red). The trivial partitions $P^1_{1,1}$ and $P^0_{0,0}$ are marked in blue.

CHAPTER 5

2-player Consensus in Directed Dynamic Networks

In this section, we present our main contribution: We apply the model of the previous chapters to deterministic distributed consensus in synchronous directed dynamic networks connected by unreliable, unidirectional links controlled by a message adversary. We restrict our investigation to systems made up of 2 processes when solving the following tasks:

1. Specify the knowledge required to solve consensus in this setting.
2. Specify the possible actions and thus the minimal Action Models.
3. Use the approach from Chapter 4 to get a lower bound on the communication complexity.
4. Specify a „minimal“ communication topology that allows to solve consensus for this communication complexity.

We start with some definitions in Section 5.1 and a survey of some related previous results on consensus in directed dynamic networks, obtained both via classical methods as well as knowledge-based ones in Section 5.2. The lessons learned from applying our approach are presented in Section 5.3. Section 5.4 provides our results on necessary and sufficient conditions for solving consensus in the case of two processes.

5.1 Problem Definition

In the *consensus* problem, each process p has an initial value x_p and a decision value y_p in its local state. The value y_p is written only once, and is undefined ($y_p = \perp$) initially.

To solve consensus, an algorithm has to fulfill the following properties for each process $p, q \in \Pi$:
(Agreement) If p assigns value v_p to y_p and q assigns value v_q to y_q, then $v_p = v_q$.
(Termination) Eventually, every p assigns a value to y_p.
(Validity) If each process p has input $x_p = v$, then all processes q have to decide $y_q = v$.

In Section 5.3 and Section 5.4, we restrict the system to have two processes $\{p_0, p_1\}$ only. Thus, in each round r, a message adversary may choose the network graph $\mathcal{G}^r$ from the set of graphs $\{\leftarrow, \leftrightarrow, \rightarrow\}$. Also note that we do not allow the empty graph there. Whenever an empty graph is in this set, the adversary may choose the sequence consisting of only empty graphs arbitrarily long, which makes consensus not solvable.

Many message adversaries are defined based on the guaranteed existence of certain graph properties over time. A *root component*, or simply *root*, $R(\mathcal{G})$ of a graph $\mathcal{G}$ denotes a set of vertices which is strongly connected and where there is no edge from any vertex outside $R(\mathcal{G})$ to some vertex inside $R(\mathcal{G})$. A graph containing exactly one root is called a *rooted* graph. This notion is extended towards a *vertex-stable root component*, or *stable root*: If in some graph sequence $(\mathcal{G}^r)_{r=a}^{b} = (\mathcal{G}^a, \mathcal{G}^{a+1}, \dots, \mathcal{G}^b)$ each graph $\mathcal{G}^r$ is rooted and for each such graph the root $R(\mathcal{G}^r)$ consists of the same processes, this root is called a *vertex-stable root component.* Its *dynamic diameter D* is the number of rounds needed such that the state s_i^{a-1} of the root members $i \in R$, at the beginning of a graph sequence $\sigma = (\mathcal{G}^r)_{r=a}^{b}$ in which R is a vertex-stable root component for a sufficient number of consecutive rounds in σ, affects another state s_j^{a-1+D}. In the case of two processes, obviously $D = 1$.

5.2 Previous Results

We start with some related existing results regarding consensus. Section 5.2.1 lists algorithms and impossibility results for consensus in directed dynamic networks, as defined above. Some related results obtained by epistemic reasoning in the general distributed computing setting are surveyed in Section 5.2.2.

5.2.1 Consensus in Directed Dynamic Networks

[BRS12] showed that consensus is impossible to solve for a message adversary that does not eventually generate a root component that is not vertex-stable for at least D rounds, with D the dynamic diameter of the dynamic network. They also developed an algorithm, which solves consensus for a message adversary that guarantees that each graph $\mathcal{G}^r$ contains a single root component that is eventually stable for $4D$ rounds.

[BRS$^+$15] developed a more complex algorithm, which is able to solve consensus under a message adversary that may generate multiple vertex-stable root components but ensures some distinct information flow between successive root components.

In [SWS16], a message adversary has been presented that guarantees rooted graphs and the eventual existence of a single vertex-stable root for $2D + 1$ rounds. An algorithm has been provided, which solves consensus under this message adversary.

[WSS16] (Theorem 2) proves that even non-uniform consensus cannot be solved under $\Diamond\text{STABLE}_D(x)$ (see Definition 5.3.1) if $x \leq D$, where x is the duration of the eventually guaranteed stable root. Informally, this is because there are executions, even with a stability phase of D rounds, where some process cannot precisely determine the root component of the stability window. The determination of this root component is crucial since each root component may be the base for a decision in the suffix of some indistinguishable execution. In addition, they present an algorithm that solves consensus under message adversary $\Diamond\text{STABLE}_D(D + 1)$.

5.2.2 Consensus and Knowledge

The classic consensus problem is defined in the model were all the links are reliable, but up to f processes may fail by crashing. To solve consensus in this model, an algorithm has to fulfill the following properties for each process $p, q \in \Pi$:
(Agreement) $y_p = y_q$ for all non-faulty processes p and q.
(Termination) Eventually, y_p is assigned a value at every non-faulty process p.
(Validity) If $x_p = v$ for all processes p, then $y_q = v$ for every terminated non-faulty process q.

[HM90] is a seminal work on the knowledge-base approach in distributed environments. The authors used the well-known *Coordinated Attack Problem*, which can be seen as an instance of simultaneous consensus, to investigate different types of common knowledge. It turns out that any protocol for the coordinated attack problem has the property that whenever the generals attack, it is common knowledge that they are attacking. They also investigate in how common knowledge can be attained and come to the conclusion that nothing can become common knowledge unless it is also common knowledge in the absence of communication. Since practical distributed systems are subject to temporal imprecision, common knowledge cannot be attained in such systems.

Thus, [HM90] also examines the states of knowledge which are attainable in systems in which communication delivery is guaranteed but message delivery times are uncertain. They come up with the notion of *ε-Common Knowledge*, which is defined similar to classic common knowledge. Rather than everyone knowing a fact at the same time, however, all of the processes come to know the fact within an interval of ε time units. As common knowledge corresponds to simultaneous actions, ε-common knowledge corresponds to actions that are guaranteed to be performed within ε time units of one another. E.g., the attacking generals can attack within one hour of each other if and only if they attain $1h$-common knowledge of the fact that they are attacking.

Another notion of [HM90] is *$\diamond$-Common Knowledge* defined similarly to ε-common knowledge, with the difference that all processes come to know a fact at some unknown point of time in an execution. $\diamond$-common knowledge corresponds to eventual actions.

E.g., in an instance of the consensus problem in which all of the processes eventually decide a value, the value which is decided upon has to become $\diamond$-common knowledge.

[CGM13] investigated the knowledge that a single process requires to decide a value in binary consensus (where the input values x_p are from $\{0, 1\}$). They came up with the notions of *strict domination* and *unbeatable protocols*. $\mathcal{Q}$ *strictly dominates* $\mathcal{P}$, denoted by $\mathcal{Q} \preceq \mathcal{P}$, if for all adversaries α and every process p, if p decides in $\mathcal{P}$ under α at time t_p, then p decides in $\mathcal{Q}$ under α at some time $t'_p \leq t_p$. $\mathcal{P}$ is an unbeatable protocol for some problem P if no protocol $\mathcal{Q}$ solving P strictly dominates $\mathcal{P}$. Their protocols differ from others, since they are not defined based on the usual algorithmic *message passing* level, but on the *knowledge* level. They assume that the underlying communication is *full-information*, i.e., each process sends its entire local state to all the other processes in each message.

In more detail, a process can decide 0 as soon as it *knows* that $x_p = 0$ for some correct process p. Moreover, as long as a process considers it possible that some process currently knows that $x_p = 0$ for some correct process p, it can not decide 1, but may do so as soon as it *knows* that no process q knows $x_p = 0$ for any correct process p. Thus, the necessary knowledge for solving consensus of process p is:

$$(1) \text{ if } K_p(x_q = 0) \text{ for some correct } q \text{ then decide } 0$$

$$(2) \text{ if } K_p \bigwedge_{q \in \Pi} \neg K_q(x_p = 0) \text{ for some correct } p \text{ then decide } 1$$

A process p can decide 0 if it knows that some correct process q has an initial value of 0 according to (1). A process p can decide 1 at time t_p if there is no hidden path with respect to p by (2).

Based on this, the authors identified conditions on the necessary and sufficient communication structures, namely, the absence of *hidden paths*. They say the state of process p at time t_p is hidden from the process q at time t_q, if both (i) p does not know that q failed before time t_p and (ii) there is no message chain from process q at time t_q to process p at time t_p. There is a hidden path with respect to process p at time t_p if there is a sequence of processes $q_0, q_1, \ldots, q_m$ and the state of q_i at time i is hidden from p at time t_p, for all $i \in \{0, 1, \ldots, t_p\}$.

5.3 Consensus and Communication Complexity

The goal of our research is to specify a minimal network topology a message adversary has to guarantee such that consensus is solvable under this adversary. The approach pursued in this section is to (i) determine a lower bound on communication complexity for the consensus problem in directed dynamic networks using the knowledge each single process requires to decide a value, such that Agreement, Validity and Termination hold, and (ii) to infer some conditions from it. Unfortunately, however, we encountered a problem here: There is no finite lower bound on the communication complexity for the consensus problem in general dynamic networks. To show this, we will prove that there is a message adversary such that every algorithm solving consensus under this adversary has unbounded bit complexity in some run.

Definition 5.3.1 ($\Diamond$STABLE$_D(x)$)**.** The message adversary $\Diamond$STABLE$_D(x)$ ensures that the following properties hold for any graph sequence $\sigma \in \Diamond$STABLE$_D(x)$:

- σ eventually has a R-rooted subsequence of length x,
- every graph $\mathcal{G}^r \in \sigma$ is rooted, and
- the dynamic diameter is D.

[WSS16] developed an algorithm solving consensus under $\Diamond$STABLE$_D(D+1)$. In the following, we use the term *stabilization round* r_σ as the first round of the first 2-stable root component of a run σ in $\Diamond$STABLE$_D(2)$.

Lemma 5.3.1. *There is no algorithm solving consensus for $\Diamond$STABLE$_1(2)$ that terminates in any round $r \leq r_\sigma$ on every graph sequence σ.*

Proof. Suppose in contradiction that there is such an algorithm $\mathcal{A}$.

Let σ be an arbitrary graph sequence with stabilization round r_σ generated by $\Diamond$STABLE$_1(2)$, such that $\mathcal{A}$ solves consensus and terminates in some round $r \leq r_\sigma$.
Consider an arbitrary graph sequence ε generated by $\Diamond$STABLE$_1(1) \supseteq \Diamond$STABLE$_1(2)$ without any root stable for at least 2 consecutive rounds, such that $\sigma_{r_\sigma} = \varepsilon_{r_\sigma}$, i.e., the prefixes of length r_σ are the same for σ and ε.

By our assumption, $\mathcal{A}$ would solve consensus on σ and terminate in some round $r \leq r_\sigma$, thus it would terminate within prefix σ_{r_σ}. Since $\varepsilon_{r_\sigma} = \sigma_{r_\sigma}$, no process p would be able to distinguish σ and ε within this prefix, thus $\mathcal{A}$ would solve consensus also on ε and terminate in the same round $r \leq r_\sigma$, a contradiction to the fact that consensus under $\Diamond$STABLE$_1(1)$ is impossible according to [WSS16] (proof of) Theorem 2. □

Lemma 5.3.2. *In any algorithm $\mathcal{A}$ which solves consensus in $\Diamond$STABLE$_1(2)$, both processes attempt to send at least one bit in r_σ in run σ.*

Proof. Suppose in contradiction that there is such an algorithm $\mathcal{A}$ in which at most one of the processes attempts to send some message containing at least one bit.

Let σ be an arbitrary graph sequence generated by $\Diamond$STABLE$_1(2)$. Denote by r_σ the first round of the first 2-stable root in σ. By Lemma 5.3.1, $\mathcal{A}$ terminates in some round $t_\sigma > r_\sigma$ on σ.

Let ε be another arbitrary graph sequence generated by $\Diamond$STABLE$_1(2)$, such that $\sigma_{r_\sigma - 1} = \varepsilon_{r_\sigma - 1}$, $\mathcal{G}_\sigma^{r_\sigma} \neq \mathcal{G}_\varepsilon^{r_\sigma}$, $(\mathcal{G}_\sigma^r)_{r=r_\sigma+1}^{t_\sigma} = (\mathcal{G}_\varepsilon^r)_{r=r_\sigma+1}^{t_\sigma}$ and $r_\varepsilon > t_\sigma$. I.e., the prefix of length t_σ is equal except for round r_σ and $r_\varepsilon > t_\sigma$. By Lemma 5.3.1, $\mathcal{A}$ terminates in some round $t_\varepsilon > r_\varepsilon > t_\sigma$ on ε.

There are two cases:

- No process attempts to send some bit in r_σ:
  Looking at all the possible pairs of graphs $(\mathcal{G}_\sigma^{r_\sigma}, \mathcal{G}_\varepsilon^{r_\sigma}) \in \{\leftarrow, \leftrightarrow, \rightarrow\}^2$, we see that without attempting to send any bit in round r_σ none of the processes can distinguish $\mathcal{G}_\sigma^{r_\sigma}$ and $\mathcal{G}_\varepsilon^{r_\sigma}$, since none of them receives any bit in r_σ in both of the sequences.

- Exactly one process attempts to send a bit in r_σ (w.l.o.g. say p_0):
  There is a possible pair of graphs $(\mathcal{G}_\sigma^{r_\sigma}, \mathcal{G}_\varepsilon^{r_\sigma}) = (\leftrightarrow, \rightarrow)$ such that none of the processes can distinguish $\mathcal{G}_\sigma^{r_\sigma}$ and $\mathcal{G}_\varepsilon^{r_\sigma}$. p_1 cannot distinguish the two graphs since in both σ and ε it receives the message from p_0. Since p_1 does not attempt to send any bit, p_0 does not receive any bit in r_σ, even if there is an incoming edge for p_0 in $\mathcal{G}_\sigma^{r_\sigma}$.

  An example for such a scenario is $\sigma_{t_\sigma} = ((\leftarrow, \rightarrow)^k, \leftrightarrow, \leftrightarrow, \leftarrow, \rightarrow)$, in which $\mathcal{G}_\sigma^{r_\sigma} = \mathcal{G}_\sigma^{r_\sigma+1} = \leftrightarrow$, and $\varepsilon_{t_\varepsilon} = ((\leftarrow, \rightarrow)^k, \leftrightarrow, \rightarrow, \leftarrow, \rightarrow, \leftarrow, \leftarrow)$, in which $\mathcal{G}_\varepsilon^{r_\varepsilon} = \mathcal{G}_\varepsilon^{r_\varepsilon+1} = \leftarrow$ and $r_\varepsilon = t_\sigma + 1$.

  Note that if p_1 is the only process attempting to send something, the crucial pair of graphs is $(\leftarrow, \leftrightarrow)$.

Since the prefixes of length $t_\sigma > r_\sigma$ of σ and ε differ only in round r_σ and none of the processes can distinguish $\mathcal{G}_\sigma^{r_\sigma}$ and $\mathcal{G}_\varepsilon^{r_\sigma}$ in both of the above cases, none of the processes can distinguish the prefixes σ_{t_σ} and ε_{t_σ}. Thus, since $\mathcal{A}$ terminates in round t_σ on σ, it also terminates in round $t_\sigma < r_\varepsilon$ on ε. This is a contradiction to Lemma 5.3.1 for ε, thus in any algorithm $\mathcal{A}$ which solves consensus in this setting, both processes attempt to send at least one bit in round r_σ. ☐

Lemma 5.3.3. *In any run of any algorithm solving consensus in $\Diamond$STABLE$_1(2)$, at least one process has to receive at least one bit in each round $r \leq r_\sigma$.*

Proof. Let σ be an arbitrary graph sequence generated by $\Diamond$STABLE$_1(2)$ and denote the first round of the first 2-stable root component by r_σ. Let ε be another arbitrary graph sequence generated by $\Diamond$STABLE$_1(2)$, such that the first round of the first 2-stable root component in ε is in some round $r \leq r_\sigma$ and $\sigma_{r'} = \varepsilon_{r'}$, for all $r' \leq r$.

By Lemma 5.3.2 both processes attempt to send at least one bit in round r of any run of $\mathcal{A}$ on ε. Since any possible graph in round r has at least one edge, at least one process has to receive at least one bit in round r on both σ and ε.

Thus in each run of each algorithm at least one process has to receive at least one bit on any arbitrary graph sequence in any arbitrary round $r \leq r_\sigma$. ☐

Theorem 5.3.1. *Any algorithm solving consensus in $\Diamond STABLE_1(2)$ has unbounded bit complexity.*

Proof. This follows directly from Lemma 5.3.3. Since in $\Diamond$STABLE$_1(2)$ the number r_σ of the first round of the first stable root may be unbounded, in each run of each algorithm at least one process has to receive at least one bit in an unbounded number of rounds, leading to an unbounded number of bits received. □

Theorem 5.3.1 implies that it is not possible to give a finite lower bound on communication complexity for the consensus problem in directed dynamic networks under $\Diamond$STABLE$_1(2)$ and, hence, in general. Thus our idea to look for such a bound to find a minimal communication topology based on the results of Chapter 4 cannot work out. Nevertheless, during this research, we developed alternative ideas that lead us to necessary and sufficient conditions for consensus in dynamic networks with $n = 2$. In the next section we present our results.

5.4 Solvability of Consensus

In this section, we will identify necessary and sufficient conditions for a message adversary that allows consensus to be solved on two processes in a directed dynamic network.

In Theorem 5.4.1, we will re-prove the well-known result [SW89, SWK09, CG13] that it is impossible to solve consensus under the unrestricted message adversary MA_u, which is allowed to choose any graph sequence $(\mathcal{G}^r)_{r=1}^{\infty}$ with $\mathcal{G}^r \in \{\leftarrow, \leftrightarrow, \rightarrow\}$.

There are algorithms solving consensus under messages adversaries that are restricted by exactly one infinite graph sequence, e.g., $MA_{\leftrightarrow^*} = MA_u \setminus \{\leftrightarrow^*\}$, which is allowed to choose any graph sequence except the one consisting of $\leftrightarrow$ in each round. Solving consensus on $MA_{\leftrightarrow^*}$ is surprisingly easy: Initially each process p sends its initial value x_p. In each round $r \geq 2$, each process sends the value it received in the previous round. Once it receives the same value it sent in the previous round or does not receive a value at all, a process decides this value and terminates. This algorithm is guaranteed to work correctly since $MA_{\leftrightarrow^*}$ ensures that the graph in some round is eventually $\leftarrow$ or $\rightarrow$, i.e., some process (say p) does not receive any value in some round. Thus, the value process q attempted to send is lost, and from this time on there is only the value that p sent present in the system. Hence it is safe to decide on it.

Therefore, our hypothesis is that consensus is solvable under some message adversary MA_S, if and only if $MA_S = MA_u \setminus S$, with S a (small) set of sequences satisfying certain conditions. We start our considerations by looking at the unrestricted message adversary MA_u and will derive that, in each round r, there is an indistinguishability chain: For each prefix σ_r, there is a prefix σ'_r which is indistinguishable from σ for p and a prefix σ''_r which it is indistinguishable from σ for q, formally expressed as $\sigma' \sim_r \sigma \sim_q \sigma''$. This indistinguishability chain includes all possible prefixes of graph sequences that arise in our impossibility proof for MA_u (Theorem 5.4.1). To prove this, we come up with the notion of a *Sorted Sequence Tree.*

In Section 5.4.1 we define this tree and prove some useful properties. Subsequently, in Section 5.4.2, we prove the impossibility of solving consensus under MA_u. Finally, in Section 5.4.3, we introduce the notions of *fair* and *unfair* graph sequences and give necessary and sufficient conditions a message adversary has to meet to allow solving consensus in a directed dynamic network of two processes.

5.4.1 Sorted Sequence Tree

A crucial prerequisite for our analysis is the *Sorted Sequence Tree* (SST), which is designed to (i) enumerate all the r-round prefixes of possible graph sequences σ in the two process system and (ii) to sort these sequences such that two neighboring prefixes are indistinguishable from each other for at least one process.

Definition 5.4.1 (Sorted Sequence Tree)**.**
The Sorted Sequence Tree (SST) for graph sequences made up from $\{\rightarrow, \leftrightarrow, \leftarrow\}$ is constructed as follows:

- Start with the first level:
$$\begin{tikzcd}
& \bullet \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\leftarrow & \leftrightarrow & \rightarrow
\end{tikzcd}$$
- At each level attach to each leaf L, where L is identified by the path leading to it
  - If $N_{\leftrightarrow}^{L}$ is even:
$$\begin{tikzcd}
& L \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\leftarrow & \leftrightarrow & \rightarrow
\end{tikzcd}$$
  - If $N_{\leftrightarrow}^{L}$ is odd:
$$\begin{tikzcd}
& L \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & \\
\rightarrow & \leftrightarrow & \leftarrow
\end{tikzcd}$$

  with $N_{\leftrightarrow}^{L}$ denoting the number of bidirectional graphs ($\leftrightarrow$) along the path from the root to leaf L.

It is easy to see that the SST enumerates all graph sequences, since each node has all of the three possible graphs as children, i.e., each prefix of a graph sequence is extended with all possible graphs for the next round. Figure 5.1 depicts the SST up to level 3. We see that each single prefix of length r (σ_r) corresponds to exactly one path from the root to exactly one leaf L at level r, e.g., the dashed path in the figure corresponds to the prefix $(\leftrightarrow, \leftarrow, \rightarrow)$. We also say that the leaf at level r itself corresponds to the prefix σ_r. Note that $N_{\leftrightarrow}^{\sigma_r}$, the number of bidirectional rounds in prefix σ_r, is equal to $N_{\leftrightarrow}^{L}$, and $N_{\leftrightarrow}^{(\leftrightarrow,\leftarrow,\rightarrow)} = 1$

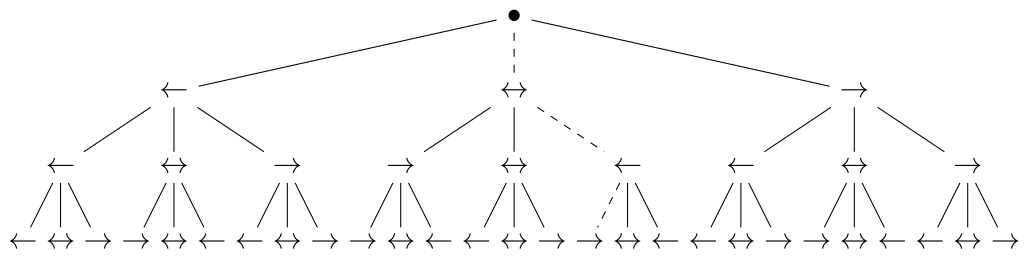


Figure 5.1: Sorted Sequence Tree up to level 3. The dashed path corresponds to the graph sequence $(\leftrightarrow, \leftarrow, \rightarrow)$.

As already stated, we need the notion of neighborhood between nodes of the SST and further on graph sequences and prefixes. The goal is to prove that at each level r there is an indistinguishability chain consisting of all prefixes of length r. In such an indistinguishability chain, two neighboring sequences in the chain are indistinguishable from each other for at least one of the processes. To do so, we define the neighbors of a node at each level r (Definition 5.4.2) and prove that this neighborhood relation creates exactly one chain of neighbors at this level, such that this chain contains all the nodes at level r (Lemma 5.4.3). Using this neighborhood relation for nodes, we then define the neighborhood relation on prefixes and prove the existence of such a chain there (Corollary 5.4.1).

Definition 5.4.2. We say that node ℓ is the *left neighbor* of node ℓ' ($\ell \neq \ell'$) on level r of the SST, iff:

- If ℓ and ℓ' share the same parent p and
  - ℓ is the left child of p and ℓ' is the central child of p or
  - ℓ is the central child of p and ℓ' is the right child of p.
- If ℓ and ℓ' do not share the same parent but have parents $p \neq p'$ respectively, then ℓ is ℓ''s left neighbor, iff:
  - p is p''s left neighbor on level $r - 1$ and
  - ℓ is p's right child and ℓ' is p''s left child.

ℓ is the *right neighbor* of ℓ' iff ℓ' is the left neighbor of ℓ.

We define the *leftmost* node at level r to be the one with only $\leftarrow$ on the path from the root of the SST to the node. The *rightmost* node at level r is the one with only $\rightarrow$ on the corresponding path.

Lemma 5.4.1. *At each level r, the leftmost (rightmost) node has exactly one right (left) neighbor, while the other nodes have exactly one right and one left neighbor.*

Proof by induction on level r. The claim obviously holds at level $r = 1$. All the nodes have the same parent, meaning $\leftarrow$ is left neighbor to $\leftrightarrow$, which is left neighbor to $\rightarrow$ by Definition 5.4.2.

Assume the claim holds for all the nodes at level $r - 1$.
Suppose ℓ is any node at level r except the leftmost and assume that it does not have a left neighbor. I.e., there is no node ℓ' such that ℓ' is a left neighbor to ℓ. Consider the following cases:

- ℓ is not the left child of its parent p. By the definition of the SST, each node p has three children and since ℓ is not the left child of p, there has to be a child left of ℓ. Thus ℓ has a left neighbor.
- ℓ is the left child of its parent p. Since ℓ is not the leftmost node on level r, p cannot be the leftmost node on level $r - 1$. By the induction hypothesis, there has to be a node p', which is the left neighbor of p at level $r - 1$. Since p' has three children, there is a node ℓ' (p''s right child), which is left neighbor to ℓ by Definition 5.4.2.

Thus any node ℓ at level r except the leftmost has to have at least one left neighbor. To show that every such node has exactly one left neighbor, suppose in contradiction that ℓ has at least two left neighbors ℓ', ℓ''.

- ℓ, ℓ' and ℓ'' share the same parent p. By Definition 5.4.2 we get that $\ell' = \ell''$.
- ℓ and ℓ' do not share the same parent and ℓ and ℓ'' do not share the same parent. We denote those parents by p, p' and p'' respectively, all of them at level $r - 1$. By Definition 5.4.2 we get that both p' and p'' (precisely $p' = p''$) are left neighbors of p. The induction hypothesis guarantees $p' = p''$ in any case. To be left neighbors of ℓ, both ℓ' and ℓ'' have to be p''s right child by Definition 5.4.2, concluding that $\ell' = \ell''$.
- ℓ and ℓ' share the same parent p, while ℓ and ℓ'' do not share the same parent. By Definition 5.4.2, ℓ' can only be ℓ's left neighbor if ℓ is p's central or right child, while ℓ has to be p's left child (so that ℓ'' can be ℓ's left neighbor), a contradiction.

Up to now we proved that each node ℓ at level r except the leftmost has exactly one left neighbor.

We will continue to show a similar property for all nodes ℓ at level r, except the rightmost, namely that each of them has exactly one right neighbor.
As before suppose that ℓ is any node at level r, except the rightmost, and assume that it does not have any right neighbor. I.e., there is no node ℓ' such that ℓ is left neighbor to ℓ'. Consider the following cases:

- ℓ is not the right child of its parent p. By the same argument as before we see that there has to be a right neighbor for ℓ.

- ℓ is the right child of its parent p. Since ℓ is not the rightmost node on level r, p cannot be the rightmost node at level $r-1$. By induction hypothesis, there has to be a node p', which is the right neighbor of p at level $r-1$. Since p' has three children, by Definition 5.4.2 there is a node ℓ' (p''s left child), which is right neighbor to ℓ.

Thus any node ℓ at level r, except the rightmost, has at least one right neighbor. To show that ℓ has exactly one right neighbor, suppose in contradiction that ℓ has at least two right neighbors ℓ' and ℓ'', i.e., ℓ is left neighbor to both ℓ' and ℓ''.

- ℓ, ℓ' and ℓ'' share the same parent p. By Definition 5.4.2 we get that $\ell' = \ell''$.

- ℓ and ℓ' do not share the same parent and ℓ and ℓ'' do not share the same parent. Denote those parents by p, p' and p'' respectively, all of them at level $r-1$. By Definition 5.4.2 p is left neighbor to both p' and p'' (precisely $p' = p''$). The induction hypothesis guarantees $p' = p''$ in any case. ℓ has to be p's right child and both ℓ' and ℓ'' have to be p''s left child by Definition 5.4.2, thus, by Definition 5.4.1, $\ell' = \ell''$.

- ℓ and ℓ' share the same parent p, while ℓ and ℓ'' do not share the same parent. By Definition 5.4.2 ℓ can only be ℓ''s left neighbor if ℓ is p's left or central child. On the other hand ℓ can only be ℓ'''s right neighbor if ℓ is p's right child, a contradiction.

Thus we can conclude that each node ℓ, except of the rightmost, at level r has exactly one right neighbor. □

Lemma 5.4.2. *The neighborhood relation does not create a cycle at any level r of the SST.*

Proof by induction on level r. The claim holds trivially at level $r = 1$, since no node is neighbor to itself and $\leftarrow$ and $\rightarrow$ both have only one neighbor.

Assume the claim holds for level $r-1$.
From Definition 5.4.1, we know that each node at level r has a parent at level $r-1$ and each node at level $r-1$ has three children at level r. By Definition 5.4.1 and Definition 5.4.2 all the children of a node build a chain of neighbors and if nodes ℓ and ℓ' at level r are neighbors, their parents p and p' at level $r-1$ are either neighbors or ℓ and ℓ' have the same parent ($p = p'$).

Suppose there is a cycle of neighbors at level r, i.e., there are nodes $\ell_0, \ldots, \ell_m$, such that ℓ_i is left neighbor to ℓ_{i-1} (for $i \geq 1$) and ℓ_0 is left neighbor to ℓ_m. Then, by the above observation, their parents $p_0, \ldots, p_{m'}$ also build a cycle at level $r-1$, which contradicts our induction hypothesis. □

Lemma 5.4.3. *At each level r, the neighborhood relation builds a chain from the leftmost node at this level to the rightmost node, such that there is a right neighbor to the leftmost node, which is a left neighbor to another node, ..., which is a left neighbor to the rightmost node. This chain contains all the nodes at level r.*

Proof. Suppose this would not be the case.
By Lemma 5.4.1 the leftmost and rightmost nodes of some level r have exactly one right respectively left neighbor, while all the other nodes of this level have exactly one right and one left neighbor. Thus there has to be such a chain starting out from the leftmost (rightmost) node and go to the right (left) neighbor. By Lemma 5.4.2, an already visited node cannot be reached twice, so the path ends in the rightmost (leftmost) node. Since no node is visited twice and each node is part of a chain, all the nodes are included in the chain. □

Lemma 5.4.4. *If node p_0 is left of p_m at level r, i.e., there are nodes $p_0, \ldots, p_m$ at level r, such that p_{i-1} is left neighbor of p_i, then all of the children of p_0 (ℓ_0^0, ℓ_0^1, ℓ_0^2) are left of all of the children of p_m (ℓ_m^0, ℓ_m^1, ℓ_m^2). The analogous statement holds if p_0 is right of p_m.*

Proof. Clearly, ℓ_0^0 and ℓ_0^1 are both left of ℓ_0^2 and ℓ_m^1 and ℓ_m^2 are both right of ℓ_m^0, by Definition 5.4.2, so it is sufficient to prove that ℓ_0^2 is left of ℓ_m^0 at level $r+1$.

Suppose ℓ_0^2 is not left of ℓ_m^0. Since p_0 is left of p_m, $p_0 \neq p_m$, thus $\ell_0^2 \neq \ell_m^0$, so ℓ_0^2 has to be right of ℓ_m^0. Thus there have to be nodes $v_0, \ldots, v_n$, such that $\ell_0^2 = v_0$, $\ell_m^0 = v_n$ and v_{i-1} right of v_i. So for each pair of neighbors (v_{i-1}, v_i) they either share the same parent or the parent p_{i-1} is right neighbor of the parent p_i, by Definition 5.4.2. Thus at level r, node p_0, parent of $v_0 = \ell_0^2$, is right neighbor to p_1, which is ... right neighbor to p_{m-1}, which is right neighbor to p_m. So p_0 is right of p_m, contradicting our premise, as the same node cannot be the right and left neighbor of some other node by Definition 5.4.2.

The proof, that if p_0 is right of p_m, the children of p_0 are right of the children of p_m, is analogous. □

As already stated, each prefix σ_r of length r of some graph sequence σ corresponds to exactly one path from the root of the SST to a node ℓ at level r of the SST respectively to node ℓ itself. Thus it is easy to extend the neighborhood of nodes to the neighborhood of prefixes.

Definition 5.4.3. Consider two graph sequences σ and σ' and their prefixes σ_r and σ'_r of length r. We say that σ_r is left (right) neighbor of σ'_r, iff their corresponding paths in the SST end in leaves ℓ and ℓ' respectively and ℓ is left (right) neighbor to ℓ'.

Corollary 5.4.1 directly follows from Lemma 5.4.3, since, by Definition 5.4.3, the neighborhood of graph sequences is defined using the neighborhood of nodes.

Corollary 5.4.1. The neighborhood relation for graph sequences creates a chain of prefixes of length r from prefix $\leftarrow^r$ to $\rightarrow^r$.

In the following Lemma we take a look at the structure of two neighboring graph sequences.

Definition 5.4.4. We say that two graphs $\mathcal{G}_1^r$ and $\mathcal{G}_2^r$ differ asymmetrically (in round r) if w.l.o.g. $\mathcal{G}_1^r = \leftrightarrow$ and $\mathcal{G}_2^r \in \{\leftarrow, \rightarrow\}$. Two graph sequences σ, σ' with prefixes σ_{r_0}, σ'_{r_0} differ asymmetrically from each other in exactly one round $r \leq r_0$, if $\mathcal{G}_1^r \in \sigma_{r_0}$ and $\mathcal{G}_2^r \in \sigma'_{r_0}$ differ asymmetrically in round r and all other graphs in σ_{r_0}, σ'_{r_0} are the same.

Lemma 5.4.5. *Consider two graph sequences σ and σ' with prefixes σ_{r_0} and σ'_{r_0} of length r_0. σ_{r_0} and σ'_{r_0} differ asymmetrically from each other in exactly one round $r \leq r_0$, if and only if they are neighbors.*

Proof $\Rightarrow$ by induction on prefix length r_0. Obviously this holds for all prefixes of length $r_0 = 1$, since all of them differ in exactly one round and both $\leftarrow$ and $\rightarrow$ are neighbors to $\leftrightarrow$.

Assume the claim holds for all prefixes of length $r_0 - 1$.
Consider two prefixes of length r_0, σ_{r_0} and σ'_{r_0}, that differ asymmetrically in exactly one round $r \leq r_0$, with $\mathcal{G}_\sigma^r = \leftrightarrow$ and $\mathcal{G}_{\sigma'}^r \in \{\leftarrow, \rightarrow\}$. There are two cases:

- $r < r_0$: Since σ_{r_0} and σ'_{r_0} differ asymmetrically in exactly one round, we can conclude that $\mathcal{G}_\sigma^{r_0} = \mathcal{G}_{\sigma'}^{r_0}$. Thus we can apply the induction hypothesis on the two prefixes σ_{r_0-1} and σ'_{r_0-1} of length $r_0 - 1$, which implies that σ_{r_0-1} and σ'_{r_0-1} are neighbors. Clearly σ_{r_0-1} and σ'_{r_0-1} correspond to the parent nodes of σ_{r_0} and σ'_{r_0}. As $\mathcal{G}_\sigma^{r_0} = \mathcal{G}_{\sigma'}^{r_0}$, by Definition 5.4.2 and Definition 5.4.3, σ_{r_0} and σ'_{r_0} are neighbors in the SST, since $N_{\leftrightarrow}^{\sigma_{r_0-1}} \neq N_{\leftrightarrow}^{\sigma'_{r_0-1}}$.
- $r = r_0$: Since σ_{r_0} and σ'_{r_0} differ asymmetrically in exactly one round, we can conclude that the two prefixes σ_{r_0-1} and σ'_{r_0-1} are equal. Thus they correspond to the same node in the SST, which is the parent of the nodes corresponding to σ_{r_0} and σ'_{r_0}. Since $\mathcal{G}_\sigma^{r_0} = \leftrightarrow$ and $\mathcal{G}_{\sigma'}^{r_0} \in \{\leftarrow, \rightarrow\}$, by Definition 5.4.1, Definition 5.4.2 and Definition 5.4.3, we can conclude that σ_{r_0} and σ'_{r_0} are neighbors. □

Proof $\Leftarrow$ by induction on sequence length r_0. Obviously all prefixes of length $r_0 = 1$ differ from each other in exactly one round. Since $\leftarrow$ is left neighbor to $\leftrightarrow$, which is left neighbor to $\rightarrow$, the claim holds.

Assume the claim holds for all prefixes of length $r_0 - 1$.
Consider two neighboring prefixes of length r_0, σ_{r_0} and σ'_{r_0}, and denote their prefixes of length $r_0 - 1$ by σ_{r_0-1} and σ'_{r_0-1} respectively.
We have to consider two cases:

- $\sigma_{r_0-1} = \sigma'_{r_0-1}$:
  Obviously σ_{r_0} and σ'_{r_0} differ only in round r_0. By Definition 5.4.1, Definition 5.4.2 and Definition 5.4.3, we see that, since σ_{r_0} and σ'_{r_0} are neighbors, (w.l.o.g.) $\mathcal{G}_\sigma^{r_0} = \leftrightarrow$. Hence, σ_{r_0} and σ'_{r_0} differ asymmetrically in round r_0 only.

- $\sigma_{r_0-1} \neq \sigma'_{r_0-1}$:
  By Definition 5.4.2 and Definition 5.4.3, σ_{r_0-1} and σ'_{r_0-1} are neighbors, since σ_{r_0} and σ'_{r_0} are neighbors. Due to the induction hypothesis, there exists a round $r \leq r_0 - 1$ such that $\mathcal{G}_\sigma^r \neq \mathcal{G}_{\sigma'}^r$ and (w.l.o.g.) $\mathcal{G}_\sigma^r = \leftrightarrow$, while $\mathcal{G}_{\sigma'}^r \in \{\leftarrow, \rightarrow\}$, and for all rounds $r' \neq r$: $\mathcal{G}_\sigma^{r'} = \mathcal{G}_{\sigma'}^{r'}$. Thus, $N_\leftrightarrow^{\sigma_{r_0-1}} = N_\leftrightarrow^{\sigma'_{r_0-1}} + 1$. By Definition 5.4.2 and Definition 5.4.3, the neighborhood of σ_{r_0} and σ'_{r_0} requires that σ_{r_0} corresponds to the left (right) child of the node corresponding to σ_{r_0-1} and σ'_{r_0} corresponds the right (left) child of the node corresponding to σ'_{r_0-1}. Due to Definition 5.4.1 and the fact that $N_\leftrightarrow^{\sigma_{r_0-1}}$ and $N_\leftrightarrow^{\sigma'_{r_0-1}}$ differ by exactly 1, we get that $\mathcal{G}_\sigma^{r_0} = \mathcal{G}_{\sigma'}^{r_0}$. Thus we can conclude that σ_{r_0} and σ'_{r_0} only differ in round r, with $\mathcal{G}_\sigma^r = \leftrightarrow$ and $\mathcal{G}_{\sigma'}^r \in \{\leftarrow, \rightarrow\}$. □

We are now ready to move on to the next section, where we prove the impossibility of consensus under the unrestricted message adversary MA_u.

5.4.2 Impossibility of Consensus under MA_u

In this section, we re-prove the well-known result [SWK09] that consensus is impossible to solve in a 2 process system in a directed dynamic network under the unrestricted message adversary MA_u by using SSTs. To do so, we exploit that, in any round r_0, each process p is uncertain about the round $r \leq r_0$, in which it received the last message, and that the notions of indistinguishability and neighborhood are equivalent (Lemma 5.4.6 and Lemma 5.4.7). The impossibility follows from the fact that there is an indistinguishability chain between the prefixes $\leftarrow^{r_0}$ and $\rightarrow^{r_0}$, due to which at least one process cannot decide in round r_0 under MA_u.

We will first explore what processes can possibly know about the particular graph sequence experienced in a run. Obviously, their local knowledge is maximized by a full-history graph approximation algorithm:

Definition 5.4.5. In a run with graph sequence $\sigma = (\mathcal{G}^r)_{r=1}^\infty$, a *full history algorithm* maintains a local estimate $\sigma_{k,p} = (\mathcal{G}_p^r)_{r=1}^k$ at process p that is computed at the end of round k. It is sent to the other process q in round $k+1$. As q also maintains a local estimate $\sigma_{k,q} = (\mathcal{G}_q^r)_{r=1}^k$, it updates this estimate to $\sigma_{k+1,q} = (\mathcal{G'}_q^r)_{r=1}^{k+1}$. For each round $r \in \{1, k\}$, $\mathcal{G}_q^r \in \sigma_{k,q}$ is a set which contains either a single graph or contains two graphs, which are indistinguishable for q. I.e., $\mathcal{G}_q^r \in \{\{\leftarrow\}, \{\leftrightarrow\}, \{\rightarrow\}, \{\rightarrow, \leftrightarrow\}\}$. The analogous statement holds for p.

Receiving the estimate $\sigma_{k,p}$ in some round $k+1$, q updates and extends its estimate for each round $r \leq k$ as follows:

- $\mathcal{G}_q^r \in \{\{\leftarrow\}, \{\leftrightarrow\}, \{\rightarrow\}\}$: ${\mathcal{G}'}_q^r = \mathcal{G}_q^r$
- $\mathcal{G}_q^r = \{\rightarrow, \leftrightarrow\}$: ${\mathcal{G}'}_q^r = \mathcal{G}_q^r \cap \mathcal{G}_p^r$

For round $k+1$, the estimate is ${\mathcal{G}'}_q^r = \leftarrow$ if q does not receive a message and ${\mathcal{G}'}_q^r = \{\rightarrow, \leftrightarrow\}$ else. Again, the analogous statement holds for p's estimate.

Lemma 5.4.6. *Running a* full graph history *algorithm $\mathcal{A}$, for any graph sequence σ, process p at any round r_0 is uncertain about the graph $\mathcal{G}_\sigma^r$ of round $r \leq r_0$ in which it received the last message $m_{q,p}^r$ from process q, provided $m_{q,p}^r$ exists. In this case, p considers it possible that $\mathcal{G}_\sigma^r$ is either the graph in which both processes received a message or the graph in which only p received a message. However, p knows the graph $\mathcal{G}_\sigma^{r'}$ for any other round $r' \neq r$ $(r' \leq r_0)$.*

Proof by induction on sequence length r_0. Choose any sequence σ and consider its prefix σ_{r_0} of length r_0.

In an inductive manner, we start at the prefix of length $r_0 = 1$. Since all the processes run a *full graph history* algorithm, both p and q try to send their whole history to the other process. W.l.o.g. consider the situation of p. If p does not receive any message in round $r_0 = 1$, it knows that the underlying communication graph is $\mathcal{G}_\sigma^1 = \rightarrow$. On the other hand, if q receives a message from p in round $r_0 = 1$, it is uncertain about $\mathcal{G}_\sigma^1$, since it does not know whether or not q received the message sent from p to q, so it cannot distinguish the graphs $\leftrightarrow$ and $\leftarrow$ in the case it received a message in round $r_0 = 1$.

Suppose the claim holds for any prefix of length $r_0 - 1$.
Consider the prefix σ_{r_0} of length r_0 and prefix σ_{r_0-1} of length $r_0 - 1$. By the induction hypothesis, there may be a round $r \leq r_0 - 1$ in which p received the last message $m_{q,p}^r$ in σ_{r_0-1}, where p is uncertain about the graph $\mathcal{G}_\sigma^r$, but it knows the graph $\mathcal{G}_\sigma^{r'}$ for any other round $r' \neq r$ $(r' \leq r_0 - 1)$. Distinguish the two cases:

- p does not receive any message in round r_0. As in the base case, p knows that the graph is $\mathcal{G}_\sigma^{r_0} = \rightarrow$. But as it does not receive any additional information about any previous rounds it is still uncertain about the graph $\mathcal{G}_\sigma^r$. For any other round $r' \neq r$ $(r' < r_0)$, p knows the graph $\mathcal{G}_\sigma^{r'}$ by the induction hypothesis.
- p receives a message in round r_0. As in the base case, p is uncertain about the graph $\mathcal{G}_\sigma^{r_0}$, since it cannot distinguish between $\leftarrow$, in which the message to q gets lost, and $\leftrightarrow$ in which q receives the message. On the other hand, it receives q's whole graph history and thus gets additional information about the graph $\mathcal{G}_\sigma^r$ in the message $m_{q,p}^{r_0}$:
  - q is uncertain about $\mathcal{G}_\sigma^r$:
    By the induction hypothesis, q considers it possible that $\mathcal{G}_\sigma^r \in \{\rightarrow, \leftrightarrow\}$ at the end of round $r_0 - 1$. As already stated, p considers it possible that $\mathcal{G}_\sigma^r \in \{\leftarrow, \leftrightarrow\}$. Thus, at the reception of $m_{q,p}^{r_0}$, p can conclude that $\mathcal{G}_\sigma^r = \leftrightarrow$.

- q is certain about $\mathcal{G}_\sigma^r$:
  By the induction hypothesis, q is certain that $\mathcal{G}_\sigma^r = \leftarrow$ at round $r_0 - 1$. At the reception of $m_{q,p}^{r_0}$ p is also certain that $\mathcal{G}_\sigma^r = \leftarrow$.

Thus p_0 is certain about $\mathcal{G}_\sigma^r$ after receiving $m_{q,p}^{r_0}$ but is uncertain about $\mathcal{G}_\sigma^{r_0}$, the graph of the round it received its last message from q. For any other round $r' \neq r$ ($r' < r_0$) p_0 knows the graph $\mathcal{G}_\sigma^{r'}$ by the induction hypothesis. □

Intuitively, if a process p is uncertain about a specific round in some prefix σ_{r_0}, it cannot distinguish between σ_{r_0} and σ'_{r_0}, the prefix which is equal to σ_{r_0} except for this single round.

Lemma 5.4.7. *Two prefixes σ_{r_0} and σ'_{r_0} are indistinguishable for some process p which is running a full graph history algorithm iff they are neighbors in the SST at round r_0.*

Proof.

⇒ Assume the prefixes σ_{r_0} and σ'_{r_0} are indistinguishable for p running a full graph history algorithm for determining the actual graph sequence chosen by the adversary. Suppose in contradiction σ_{r_0} and σ'_{r_0} are not neighbors.
Since σ_{r_0} and σ'_{r_0} are no neighbors, they have to differ in at least two rounds from each other (by Lemma 5.4.5). This contradicts the indistinguishability of σ_{r_0} and σ'_{r_0} since, by Lemma 5.4.6, p is uncertain about exactly one round $r \leq r_0$ in σ_{r_0}, thus would be able to distinguish σ_{r_0} and σ'_{r_0} via the difference in at least one other round. Thus σ_{r_0} and σ'_{r_0} have to be neighbors.

⇐ Assume prefixes σ_{r_0} and σ'_{r_0} are neighbors. Suppose in contradiction that σ_{r_0} and σ'_{r_0} are distinguishable for all processes p.
By Lemma 5.4.5, σ_{r_0} and σ'_{r_0} differ in exactly one round r_1, with $\mathcal{G}_\sigma^{r_1} =\leftrightarrow$ and $\mathcal{G}_{\sigma'}^{r_1} \in \{\leftarrow, \rightarrow\}$. Due to the possible graphs in this round, there is a process, say p, such that p received a message in round r_1 in both σ_{r_0} and σ'_{r_0}, thus (w.l.o.g.) $\mathcal{G}_{\sigma'}^{r_1} =\leftarrow$. Since r_1 is the last round in which p received a message, $\mathcal{G}_{\sigma_{r_0}}^r = \mathcal{G}_{\sigma'_{r_0}}^r =\rightarrow$ for all rounds $r \in [r_1 + 1, r_0]$. Thus p does not receive a message in rounds $r_1 + 1$ up to r_0, and so it is still uncertain about the graphs $\mathcal{G}_{\sigma_{r_0}}^{r_1}$ and $\mathcal{G}_{\sigma'_{r_0}}^{r_1}$ in round r_1 (by Lemma 5.4.6).
Thus p cannot distinguish σ_{r_0} and σ'_{r_0} in round r_0. □

Lemma 5.4.8. *For a* full graph history *algorithm $\mathcal{A}$ for determining the actual graph sequence σ, for any prefix length r, there is an indistinguishability chain from $\leftarrow^r$ to $\rightarrow^r$ that contains all the prefixes of length r. I.e., for all σ_r^i ($i \in \{1, \ldots, 3^r - 1\}$) it holds that, for $\sigma_r^0 =\leftarrow^r$ and $\sigma_r^{3^k-1} =\rightarrow^r$, σ_r^{i-1} is indistinguishable from σ_r^i for some process p.*

Proof. By Lemma 5.4.7, the indistinguishability relation and neighborhood relation are equivalent. Thus, from Corollary 5.4.1, it follows that the indistinguishability relation on any level r in the SST creates such an indistinguishability chain for every r. □

Lemma 5.4.9. *Any algorithm $\mathcal{A}$ which solves consensus, decides x_p (resp. x_q) on graph sequence $\rightarrow^*$ (resp. $\leftarrow^*$).*

Proof. Suppose there is an algorithm $\mathcal{A}$ solving consensus which decides x_q on $\rightarrow^*$. On input $x = (x_p, x_q) = (0, 1)$, both p and q would decide 1. Since p does not receive any message on $\rightarrow^*$, it cannot distinguish $x = (0, 1)$ from $x' = (0, 0)$. Thus, on $x' = (0, 0)$, p would decide 1 also. This decision contradicts validity, thus $\mathcal{A}$ does not solve consensus. An analogous proof works for the decision on x_q on $\leftarrow^*$. □

Definition 5.4.6. We say a prefix σ_r is *v-valent* for algorithm $\mathcal{A}$ if on any graph sequence σ, such that σ_r is the prefix of length r of σ, $\mathcal{A}$ decides v. We also say it is univalent, if v does not matter.
A prefix σ_r is *bivalent* if it is not univalent.

It follows that a prefix σ_r is bivalent if there are two graph sequences σ and σ', such that σ_r is prefix of both of them and $\mathcal{A}$ decides v on σ and v' on σ'. Note that once a prefix σ_r is v-valent, any $\sigma_{r'}$, with $r' \geq r$ and σ_r a prefix of $\sigma_{r'}$, is also v-valent.

With these preparations, we can now start with proving the impossibility of consensus:

Theorem 5.4.1 (Impossibility of Consensus)**.** *No algorithm can solve consensus under the unrestricted message adversary MA_u.*

Proof. It suffices to prove that no consensus algorithm based on a full history graph algorithm exists, as reduction can be used to also rule out other consensus algorithms as well. So assume there is a full history algorithm $\mathcal{A}$ which solves consensus under MA_u. Then $\mathcal{A}$ terminates on any graph sequence σ, deciding either x_p or x_q. Due to Lemma 5.4.9 there is at least one graph sequence in MA_u on which $\mathcal{A}$ decides x_p and at least one other graph sequence on which $\mathcal{A}$ decides x_q. Since each graph sequence has a decision value, there are two cases:

- $\exists r_0 \colon \forall \sigma \colon \sigma_{r_0}$ is univalent: There is a round r_0 such that all prefixes of length r_0 (and longer) are univalent.
  Consider this round r_0, in which all of those prefixes start to be univalent and look at the prefixes σ_{r_0} and ε_{r_0}, such that $\sigma_{r_0} \sim_p \varepsilon_{r_0}$ for some process p and σ_{r_0} is v-valent, while ε_{r_0} is v'-valent ($v \neq v'$). Due to Lemma 5.4.8, Lemma 5.4.9 and the fact that there is no bivalent prefix of length r_0, such a situation has to exist.
  W.l.o.g. assume that σ_{r_0} is x_p-valent, ε_{r_0} is x_q-valent and $\sigma_{r_0} \sim_p \varepsilon_{r_0}$ for process p. We now extend these two prefixes to $\sigma^{\rightarrow} = (\sigma_{r_0}, \rightarrow^*)$ and $\varepsilon^{\rightarrow} = (\varepsilon_{r_0}, \rightarrow^*)$. Since MA_u contains all the possible graph sequences, $\sigma^{\rightarrow}, \varepsilon^{\rightarrow} \in MA_u$, and since σ_{r_0} is x_0-valent and ε_{r_0} is x_1-valent, $\mathcal{A}$ decides x_0 on $\sigma^{\rightarrow}$ and x_1 on $\varepsilon^{\rightarrow}$.

  Note that since $\mathcal{G}^r_{\sigma^{\rightarrow}} = \mathcal{G}^r_{\varepsilon^{\rightarrow}} = \rightarrow$, for any $r > r_0$, and $\sigma_{r_0} \sim_p \varepsilon_{r_0}$, p cannot distinguish $\sigma^{\rightarrow}$ and $\varepsilon^{\rightarrow}$ in any round $r \geq r_0$, as it does not receive any message after round r_0. Thus, $\forall r \geq r_0 \colon \sigma^{\rightarrow}_r \sim_p \varepsilon^{\rightarrow}_r$.

Now assume w.l.o.g. that MA_u chooses $\varepsilon^{\rightarrow}$ and consider the run of $\mathcal{A}$ on $\varepsilon^{\rightarrow}$, then, by Termination and Agreement, p and q must decide x_q by some round $r \geq r_0$. Since $\sigma_r^{\rightarrow} \sim_p \varepsilon_r^{\rightarrow}$, however, this contradicts x_p-valence of σ_{r_0}.

- $\forall r \colon \exists \sigma \colon \sigma_r$ is bivalent: For each round there is a non-empty set V^r of bivalent prefixes σ_r.
  We use König's infinity lemma to show that in this case there exists some graph sequence $\sigma \in MA_u$ which is forever bivalent. For any prefix $\sigma_r \in V^r$, we choose $f(\sigma_r) = \sigma_{r-1} \in V^{r-1}$. Since $|V^r|$ is finite, so is the degree of any node in the tree induced by f. Hence, König's lemma ensures an infinite path σ in the tree, since the tree contains an infinite number of nodes for infinite r.

  So there exists a graph sequence σ such that for any round r its prefix σ_r is bivalent. Hence, $\mathcal{A}$ cannot terminate on σ. □

5.4.3 Necessary and Sufficient Conditions for Solving Consensus

Using the results of Section 5.4.1 and Section 5.4.2, we will develop necessary and sufficient conditions for solving consensus: If and only if a message adversary meets these conditions, there exists an algorithm that solves consensus in a two process system in a directed dynamic network.

As we stated at the beginning of Section 5.4, there are adversaries, restricted by only one graph sequence, under which consensus can be solved, e.g., $MA_{\leftrightarrow^*} = MA_u \setminus \{\leftrightarrow^*\}$, which is allowed to choose every graph sequence except the forever bidirectional one. One might be prone to believe that it is always enough to restrict the message adversary by just one arbitrary graph sequence. This is not the case, however, since under $MA_{\leftrightarrow,\leftarrow,\rightarrow^*}$, which is not allowed to choose the sequence starting with $\leftrightarrow, \leftarrow$, followed by infinitely many rounds of $\rightarrow$, consensus is impossible to solve, as we will see by Theorem 5.4.4: The reason is that this graph sequence is forever indistinguishable for p from $(\leftrightarrow, \leftrightarrow, \rightarrow^*)$.

Our idea is that any restricted adversary $MA_S = MA_u \setminus S$, with S a set of excluded graph sequences, that allows to solve consensus must satisfy the following property: S must contain a sub-set S', which is such that the graph sequences in S' "split" the SST in two partitions, one of them containing the x_q-valent graph sequence $\leftarrow^*$, the other the x_p-valent graph sequence $\rightarrow^*$. Due to the indistinguishability chain in Lemma 5.4.8, each graph sequence in the first partition has to be x_q-valent, while all the graph sequences in the second partition have to be x_p-valent. If S is chosen such that each process can distinguish the actual graph sequence σ from all the graph sequences in S' at some round r, then each process eventually knows in which partition the current sequence σ is and can safely decide x_p or x_q.

To formalize our idea, we start by defining the notion of *fair* and *unfair* graph sequences.

Definition 5.4.7 (Fair and Unfair Graph Sequences)**.** A *Fair Graph Sequence* is an infinite sequence of graphs $\sigma = \mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^r, \ldots$, with $\mathcal{G}^r \in \{\rightarrow, \leftrightarrow, \leftarrow\}$, such that it guarantees that for each process p and each graph sequence $\varepsilon \neq \sigma$ there exists a round r

such that p can distinguish ε_r from σ_r, i.e., $\varepsilon_r \not\sim_p \sigma_r$, in a full graph history algorithm. An *Unfair Graph Sequence* is an infinite sequence of graphs $\sigma = \mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^r, \ldots$, with $\mathcal{G}^r \in \{\rightarrow, \leftrightarrow, \leftarrow\}$, which is not fair.

Note the importance of this definition: It states that if a message adversary is restricted not to choose some fair graph sequence σ, at some round r process p recognizes that the actual graph sequence ε is not the "forbidden" graph sequence σ.

Lemma 5.4.10. *The graph sequences $\leftarrow^*$ and $\rightarrow^*$ are fair graph sequences.*

Proof. W.l.o.g. consider the graph sequence $\leftarrow^*$; the proof for $\rightarrow^*$ is analogous. Let any graph sequence $\varepsilon \neq \leftarrow^*$ be given. Clearly there has to be a round r_0 such that $\mathcal{G}_\varepsilon^{r_0} \neq \mathcal{G}_{\leftarrow^*}^{r_0}$. By Lemma 5.4.6, any process p is uncertain about the graph in the last round in which it received a message.
If ε differs from $\leftarrow^*$ in more than one round, then p is able to distinguish them at some round, since it will eventually be aware of all but one differences.
Thus, we are only interested in a graph sequence ε, where ε differs from $\leftarrow^*$ in exactly one round r_0. Since $\mathcal{G}_{\leftarrow^*}^{r_0} = \leftarrow$, $\mathcal{G}_\varepsilon^{r_0} \in \{\leftrightarrow, \rightarrow\}$. In both cases, q on ε immediately recognizes the difference, since it receives a message, while it would not receive any if the actual graph sequence would be $\leftarrow^*$. Since $\mathcal{G}_\varepsilon^{r_0+1} = \mathcal{G}_{\leftarrow^*}^{r_0+1} = \leftarrow$, in round $r_0 + 1$, p receives a message and thus receives the information that q recognized a difference between ε and $\leftarrow^*$.

Since we have chosen ε arbitrarily and shown that each process eventually recognizes the difference, the graph sequences $\leftarrow^*$ and $\rightarrow^*$ are fair. □

Definition 5.4.7 defines fair and unfair graph sequences only based on their indistinguishability for processes. We proceed with developing results about the structure of those graph sequences: Lemma 5.4.11 shows that each unfair graph sequence ends with an infinite sequence of either $\leftarrow$ or $\rightarrow$. As a consequence, Corollary 5.4.2 states that each graph sequence that does not end in such an infinite unidirectional sequence is fair.

Lemma 5.4.11. *For each unfair graph sequence ε, there exists a round r_0, such that any graph $\mathcal{G}_\varepsilon^{r+1} = \mathcal{G}_\varepsilon^r \in \{\leftarrow, \rightarrow\}$ for $r > r_0$.*

Proof. Choose an unfair graph sequence ε and suppose that it does not end in an infinite sequence of either $\leftarrow$ or $\rightarrow$. Then, for each round r with w.l.o.g. $\mathcal{G}_\varepsilon^r = \leftarrow$, there is some round $r' > r$ with $\mathcal{G}_\varepsilon^{r'} \in \{\rightarrow, \leftrightarrow\}$. Consider some arbitrary graph sequence $\sigma \neq \varepsilon$. Clearly there is some round r_0 such that $\mathcal{G}_\sigma^{r_0} \neq \mathcal{G}_\varepsilon^{r_0}$, and by Lemma 5.4.6, we can again restrict our attention to the case where σ and ε only differ in round r_0. Since ε does not end with an infinite sequence of either $\leftarrow$ or $\rightarrow$, there is either some round $r_\leftrightarrow > r_0$ with $\mathcal{G}_\varepsilon^{r_\leftrightarrow} = \leftrightarrow$ or two rounds $r_\leftarrow > r_0$ with $\mathcal{G}_\varepsilon^{r_\leftarrow} = \leftarrow$ and $r_\rightarrow > r_0$ with $\mathcal{G}_\varepsilon^{r_\rightarrow} = \rightarrow$. Thus both process p and q receive a message after r_0. By Lemma 5.4.6, they both know the actual graph $\mathcal{G}_\varepsilon^{r_0}$ and, since $\mathcal{G}_\varepsilon^{r_0} \neq \mathcal{G}_\sigma^{r_0}$, can distinguish ε_r and σ_r after some round $r > r_0$.

This contradicts the assumption that ε is an unfair graph sequence and proves that ε has to end in an infinite sequence of either $\leftarrow$ or $\rightarrow$. $\square$

Note that Lemma 5.4.11 does not imply the other direction, since $\leftarrow^*$ and $\rightarrow^*$ are both fair graph sequences, by Lemma 5.4.10.

Corollary 5.4.2. Each graph sequence that does not end in an infinite suffix $\leftarrow^*$ or $\rightarrow^*$ is a fair graph sequence.

Proof. This follows directly from Definition 5.4.7 and Lemma 5.4.11. $\square$

By Definition 5.4.7, for each unfair graph sequence σ, there is another unfair graph sequence ε, such that σ and ε are indistinguishable forever for some process p. Essentially, this is because the difference in σ and ε is not recognized by p before the graph sequence starts to be the infinite sequence of the same unidirectional graph, which does not allow recognizing the difference forever. So what would be the situation if a message adversary was restricted exactly by a pair of such forever indistinguishable unfair graph sequences? Lemma 5.4.12 states that such a pair has the same "splitting" property regarding indistinguishability as a single fair execution:

Lemma 5.4.12. *Consider a pair of unfair graph sequences σ, σ' such that there is some p for which σ and σ' are indistinguishable forever ($\exists p\colon \forall r\colon \sigma_r \sim_p \sigma'_r$). Any other sequence ε ($\varepsilon \neq \sigma$, $\varepsilon \neq \sigma'$) is distinguishable from both σ and σ' for each process from some round r_0 on, i.e., : $\exists r_0\colon \forall r \geq r_0\colon \forall p\colon \sigma_r \not\sim_p \varepsilon_r \wedge \sigma'_r \not\sim_p \varepsilon_r$.*

Proof. Suppose the opposite: There exists such a pair of unfair graph sequences σ, σ' for which there are a graph sequence ε ($\varepsilon \neq \sigma$, $\varepsilon \neq \sigma'$) and some process p which cannot distinguish ε_r from σ_r or σ'_r in any round r. I.e., $\exists p\colon \forall r\colon \varepsilon_r \sim_p \sigma_r \vee \varepsilon_r \sim_p \sigma'_r$.

As in the previous proofs, we will consider p only, since the part for q is analogous and only differs in the direction of specific edges.

First we will take a look at the consequences of $\forall r\colon \varepsilon_r \sim_p \sigma_r$: By Lemma 5.4.7, for all prefix lengths r, ε_r is neighbor to σ_r. Furthermore, by Lemma 5.4.5, ε and σ only differ in a single round r_0 and we can conclude that p is uncertain about round r_0 forever. By Lemma 5.4.6, p received its last message in round r_0 in both ε and σ. Thus, for all rounds $r_1 > r_0$: $\mathcal{G}_\varepsilon^{r_1} = \mathcal{G}_\sigma^{r_1} = \rightarrow$. Lemma 5.4.5 also implies that either $\mathcal{G}_\varepsilon^{r_0} = \leftrightarrow$ and $\mathcal{G}_\sigma^{r_0} \in \leftarrow$ or $\mathcal{G}_\varepsilon^{r_0} \in \leftarrow$ and $\mathcal{G}_\sigma^{r_0} = \leftrightarrow$; recall that p receives a message in r_0.

By analogous arguments, it follows from $\forall r'\colon \varepsilon_{r'} \sim_p \sigma'_{r'}$ that there is a round r'_0 such that $\mathcal{G}_\varepsilon^{r'_0} \neq \mathcal{G}_{\sigma'}^{r'_0}$, $\forall r'_1 \neq r'_0\colon \mathcal{G}_\varepsilon^{r'_1} = \mathcal{G}_{\sigma'}^{r'_1}$ and $\forall r'_1 > r'_0\colon \mathcal{G}_\varepsilon^{r'_1} = \mathcal{G}_{\sigma'}^{r'_1} = \rightarrow$, as well as from $\forall r\colon \sigma_r \sim_p \sigma'_r$ that there is a round r''_0 such that $\mathcal{G}_\sigma^{r''_0} \neq \mathcal{G}_{\sigma'}^{r''_0}$, $\forall r''_1 \neq r''_0\colon \mathcal{G}_\sigma^{r''_1} = \mathcal{G}_{\sigma'}^{r''_1}$ and $\forall r''_1 > r''_0\colon \mathcal{G}_\sigma^{r''_1} = \mathcal{G}_{\sigma'}^{r''_1} = \rightarrow$. It also holds that either $\mathcal{G}_\varepsilon^{r'_0} = \leftrightarrow$ and $\mathcal{G}_{\sigma'}^{r'_0} \in \leftarrow$ or $\mathcal{G}_\varepsilon^{r'_0} \in \leftarrow$ and $\mathcal{G}_{\sigma'}^{r'_0} = \leftrightarrow$ (similar for σ and σ' in round r''_0).

We claim now that $r_0 = r_0''$ resp. $r_0' = r_0''$. Suppose not, then there is a smallest one, say $r_0 \leq r_0''$. Since $\forall r > r_0 \colon \mathcal{G}_\varepsilon^r = \mathcal{G}_\sigma^r = \rightarrow$, $r_0'' > r_0$ could not be the last round in which p received a message in graph sequence ε, thus $r_0'' = r_0$. The analogous argument holds for r_0'.

Putting everything together, we have that $\mathcal{G}_\varepsilon^{r_0} \neq \mathcal{G}_\sigma^{r_0}$, $\mathcal{G}_\varepsilon^{r_0} \neq \mathcal{G}_{\sigma'}^{r_0}$ and $\mathcal{G}_\sigma^{r_0} \neq \mathcal{G}_{\sigma'}^{r_0}$ as well as $\forall r \neq r_0 \colon \mathcal{G}_\varepsilon^r = \mathcal{G}_\sigma^r$ and $\mathcal{G}_\varepsilon^r = \mathcal{G}_{\sigma'}^r$. Looking at the round r_0, we only need to distinguish the following cases: (i) $\mathcal{G}_\varepsilon^{r_0} = \leftrightarrow$, leading to $\mathcal{G}_\sigma^{r_0} = \leftarrow$ (since p receives a message in r_0) leading to $\mathcal{G}_{\sigma'}^{r_0} = \leftrightarrow$, or (ii) $\mathcal{G}_\varepsilon^{r_0} = \leftarrow$ (since p receives a message in r_0), leading to $\mathcal{G}_\sigma^{r_0} = \leftrightarrow$, which means that $\mathcal{G}_{\sigma'}^{r_0} = \leftarrow$. Thus $\varepsilon = \sigma$ or $\varepsilon = \sigma'$, a contradiction.

So any graph sequence ε is either distinguishable from σ and σ' for all processes or is equal to σ or σ'. $\square$

By Definition 5.4.7 and Lemma 5.4.12, any message adversary MA_S, with either a single fair graph sequence $\{\sigma\} = S$ or a pair of forever indistinguishable unfair graph sequences $\{\varepsilon, \varepsilon'\} = S$, allows that at some round r each process knows that the actual graph sequence is not in S.

We now define a message adversary MA_α, for which consensus is solvable, by means of Algorithm 5.1.

Definition 5.4.8. Message adversary MA_α may choose any graph sequence $\sigma \in MA_u \backslash \alpha$, with α a set of graph sequences such that:

- $\sigma \in \alpha$ a fair graph sequence, or
- ε, $\varepsilon' \in \alpha$ are unfair graph sequences such that $\exists p \colon \forall r \colon \varepsilon_r \sim_p \varepsilon_r'$.

In Corollary 5.4.3 and Theorem 5.4.2, we will prove that Algorithm 5.1 actually solves consensus for adversary MA_α. It relies on the following ideas:

As we discovered earlier, a message adversary MA_α, with either a single fair graph sequence $\{\sigma\} = \alpha$ or a pair of forever indistinguishable unfair graph sequences $\{\varepsilon, \varepsilon'\} = \alpha$, allows that at some round r, each process knows that the actual graph sequence is not in α. We can use this fact to partition the SST (Definition 5.4.1), as depicted in Figure 5.2 and Figure 5.3. The dash-dotted red paths depict the graph sequences in α. Since this path(s) are either a single fair one or a pair of forever indistinguishable unfair graph sequences, all the other graph sequences can at some point in time be distinguished from any graph sequence in α. For example, look at the dashed blue sequence, which can be distinguished from all sequences in α in both figures.

It is crucial to note that, although MA_α is not allowed to choose some graph sequence $\sigma \in \alpha$, each finite prefix σ_r is a prefix of infinitely many possibly chosen graph sequences. Thus, even in an unbounded late round r, no process can think of σ_r as forbidden, since the graph sequence σ', with $\sigma_r' = \sigma_r$ but $\sigma' \neq \sigma$, is allowed to be chosen by the adversary.

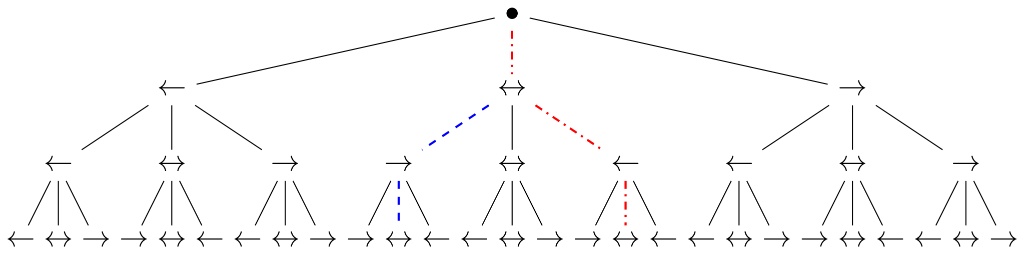


Figure 5.2: The red (dash-dotted) sequence is a single fair sequence and splits the SST in two partitions. I.e., for each graph sequence (e.g. the blue (dashed) one) all the processes are eventually able to distinguish it from this fair sequence.

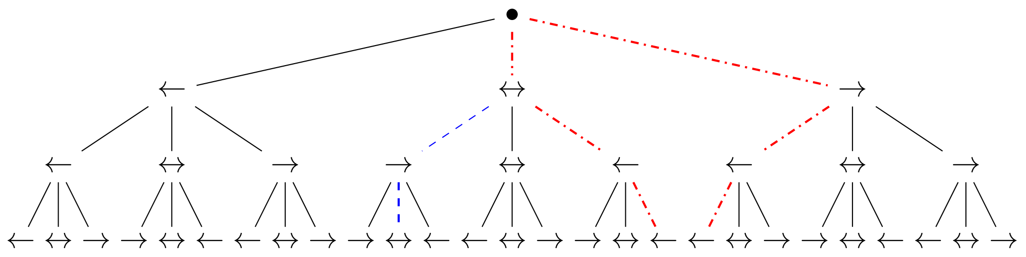


Figure 5.3: The red (dash-dotted) unfair sequences build a pair of forever indistinguishable unfair sequences and partition the SST to two parts. I.e., for each graph sequence (e.g. the blue (dashed) one) all the processes are eventually able to distinguish it from this pair of unfair sequences.

Our algorithm uses this partitioning property. Each process p gets α as input via the sets $fair$ and $unfair$ and chooses either a single fair or a pair of unfair executions deterministically, denoted by the set cut. It uses these chosen graph sequences to partition the SST in a left (yellow) partition, containing the x_q-valent graph sequence $\leftarrow^*$, and a right (green) partition, containing the x_p-valent graph sequence $\rightarrow^*$, as depicted in Figure 5.4. In any graph sequence in the left partition the algorithm has to decide x_q and in the right partition it decides x_p, due to indistinguishable chains from $\leftarrow^*$ and $\rightarrow^*$ respectively.

All the algorithm has to do is to recognize in which partition the actually chosen graph sequence ε is. We know that the prefix ε_r of any chosen graph sequence ε is, at some round r, distinguishable from the prefix c_r of each graph sequence $c \in cut$. Once this is the case in round r_0, the algorithm checks whether the indistinguishability chain from $\leftarrow^{r_0}$ towards its local estimate $\hat{\varepsilon}_{r_0}$ of ε_{r_0} of the SST contains the prefix c_{r_0} of a graph sequence $c \in cut$. If this is the case, then obviously ε is in the right partition and the algorithm decides x_p. In the other case, ε is in the left partition, which forces the algorithm to decide x_q. Note that the local estimate $\hat{\varepsilon}_{r_0}$ may contain two possible graph sequences $\hat{\varepsilon}^0_{r_0}$ and $\hat{\varepsilon}^0_{r_0}$, which differ from each other in one round $r \leq r_0$ (due to Lemma 5.4.6), while one of them is equal to the actual prefix ε_{r_0}.

Algorithm 5.1: Solving consensus, code for p_i

```
   input     : x_i ... initial value
               fair ... set of fair graph sequences not chosen by adversary
               unfair ... set of unfair graph sequences not chosen by adversary
   output    : dec ... decision value
   variables : poss_i ... possible graph sequences considered by $p_i$, represented by an array
               with element r holding the set of possible graphs for round r
               cut ... a set of graph sequences not chosen by adversary, used to split
               the set of all graph sequences
               tree ... SST containing all possible $r$-prefixes, for every $r \geq 1$
1  if fair $\neq \emptyset$ then
2      cut = deterministic_choice(fair);
3  else
4      cut = deterministic_choice_pair(unfair);
5  end
6  poss_i [*] = $\emptyset$; tree = $\perp$; dec = $\perp$; $x_{1-i}$ = $\perp$;
7  for round r $\geq 1$ do
       /* Build next level of SST according to Definition 5.4.1      */
8      tree = next_level(tree); send(x_i, poss_i);
9      receive(x_j, poss_j);
       /* Update possible graph sequences                            */
10     for k = [1 ... r −1] do poss_i [k ] = poss_i [k ] $\cap$ poss_j [k ] ;
11     if $i == 0$ then
           /* I am $p$                                                */
12         if poss_j $\neq \emptyset$ then
13             poss_i [r ] = $\{\leftrightarrow, \leftarrow\}$
14         else
15             poss_i [r ] = $\{\rightarrow\}$
16         end
17     else
           /* I am $q$                                                */
18         if poss_j $\neq \emptyset$ then
19             poss_i [r ] = $\{\rightarrow, \leftrightarrow\}$
20         else
21             poss_i [r ] = $\{\leftarrow\}$
22         end
23     end
       /* Check whether possible graph sequence can be distinguished
          from cut                                                    */
24     if $\forall$c $\in$cut : $\exists$k $\in \{1 \ldots$ r $\}$: c [k ] $\notin$ poss_i [k ] then
           /* If there is a path to $\leftarrow^r$ without crossing the cut, decide
              on $x_1$, else decide on $x_0$                          */
25         for $\sigma_r = \leftarrow^r$; $\sigma_r \notin$ cut; $\sigma_r$ = right_neighbor($\sigma_r$, tree) do
26             if $\sigma_r \in$ poss_i then dec = $x_1$; break;
27         end
28         if dec = $\perp$ then dec = $x_0$;
29     end
30 end
```

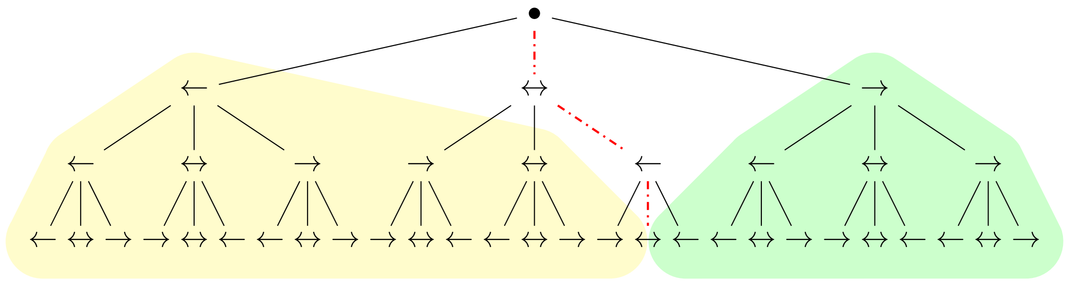


Figure 5.4: The red (dash-dotted) fair graph sequence $\varepsilon \in \alpha$ splits the SST into two partitions. As soon as a process, running Algorithm 5.1, recognizes, that the actual graph sequence σ is not in α, it checks whether it is in the left (yellow) or right (green) partition and decides x_q respectively x_p.

In Algorithm 5.1, we change notation from processes p and q to p_0 and p_1 for indexing convenience. Furthermore, we use $poss_i$ as an array of sets, which contains the local estimates of process p_i. I.e., the set $poss_i[k]$ contains all the graphs p_i thinks possible in round k. Thus, p_i can distinguish the actual graph sequence from the forbidden sequences (in the set cut), if for each graph sequence $c \in cut$ there is some round k such that $\mathcal{G}_c^k \notin poss_i[k]$. Once p_i can distinguish the actual graph sequence from the forbidden sequences (in Line 24), it checks whether its estimate is between $\leftarrow^r$ and cut (in the loop starting at Line 25). If this is the case, p_i decides x_1 else it decides x_0.

Corollary 5.4.3. Algorithm 5.1 is a *full graph history* algorithm.

Proof. This follows directly from Definition 5.4.5. □

Theorem 5.4.2. *Algorithm 5.1 solves consensus under message adversary MA_α.*

Termination. Due to Definition 5.4.8, there is either some fair graph sequence σ or two forever indistinguishable unfair graph sequences ε, ε' that MA_α is not allowed to choose. The set α, containing the forbidden fair and unfair sequences, is input to Algorithm 5.1 via the parameters $fair$ and $unfair$. In Line 2 and Line 4, each process p chooses the same (pair of) sequence(s) for cut, i.e., cut is either a single fair graph sequence σ or a pair of indistinguishable unfair sequences ε, ε'.
Due to Corollary 5.4.3, we can use Definition 5.4.7 and Lemma 5.4.12 to conclude that each process running Algorithm 5.1 can at some round r distinguish the actual graph sequence σ chosen by MA_α from all the graph sequences in cut. Thus, the statement in Line 24 eventually evaluates to $true$ in some round r, leading to a decision in either Line 28 or Line 26. Note that the loop in Line 25 terminates, since there is a fixed number of prefixes of length r (exactly $3^r - 1$). □

Validity. In Line 8, a process p_i always sends its initial value x_i, thus p_{1-i} cannot receive a value different from x_i. p_i has to decide x_0 or x_1. One of the values is p_i's initial value, the other is either $\perp$ or p_{1-i}'s initial value. W.l.o.g. assume $x_0 = \perp$ at p_1, which is only possible if the actual graph sequence $\sigma = \leftarrow^*$. By Lemma 5.4.9, $\leftarrow^*$ is x_1-valent, thus p_1 cannot decide any other value than x_1 in this case. Thus, p_i cannot decide any value other than x_0 or x_1 and validity holds. $\square$

Agreement. W.l.o.g. assume that p_0 is the first process to decide. Let us denote the graph sequence chosen by the adversary by σ. Since, in each round r, p_0 and p_1 are uncertain about at most a single round r'_0 (resp. r'_1) $\leq r$ in σ_r (Lemma 5.4.6), $poss_i^r = \{\sigma_r, \varepsilon_r^i\}$ for process p_i and $\sigma_r \sim_i \varepsilon_r^i$, thus $\varepsilon_r^0 \sim_0 \sigma_r \sim_1 \varepsilon_r^1$ ($i \in \{0,1\}$). Due to Lemma 5.4.7, ε_r^0 and ε_r^1 are neighbors to σ_r and by Lemma 5.4.1, one of ε_r^0 and ε_r^1 is the left neighbor of σ_r, while the other one is its right neighbor.

To prove agreement, we look at the two cases p_0 decides x_0 and p_0 decides x_1. p_0 can only decide in some round r_0 if all the estimated prefixes $poss_0^{r_0}$ of length r_0 are either on the left or on the right side of *cut*. We have to prove that if, in round r_0, p_0's estimated prefixes are w.l.o.g. left of *cut*, then, in some round r_1, p_1's estimated prefixes are also left of *cut*. To do so, we need to distinguish the situation in which $\varepsilon_{r_0}^0$ is the left neighbor of σ_{r_0} (and $\varepsilon_{r_0}^1$ the right neighbor of σ_{r_0}) from the situation in which $\varepsilon_{r_0}^0$ is the right neighbor of σ_{r_0} (and $\varepsilon_{r_0}^1$ the left neighbor of σ_{r_0}).

Case 1: p_0 decides x_1 in round r_0.
Suppose in contradiction that p_1 decides x_0 in round r_1, $r_0 \leq r_1$.

To contradict this assumption, we have to prove that in no round $r \geq r_0$ some prefix c_r (of any $c \in cut$) is contained in the indistinguishability chain between $\leftarrow^r$ and $\{\varepsilon_r^1, \sigma_r\}$.

In Algorithm 5.1, p_0 decides x_1 in Line 26 there is a chain:
(i) $\leftarrow^{r_0} = \gamma_{r_0}^0 \sim_{y_0} \gamma_{r_0}^1 \sim_{y_1} \cdots \sim_{y_{k-1}} \gamma_{r_0}^k = \varepsilon_{r_0}^1 \sim_1 \sigma_{r_0} \sim_0 \varepsilon_{r_0}^0$ or
(ii) $\leftarrow^{r_0} = \gamma_{r_0}^0 \sim_{y_0} \gamma_{r_0}^1 \sim_{y_1} \cdots \sim_{y_{k-1}} \gamma_{r_0}^k = \varepsilon_{r_0}^0 \sim_0 \sigma_{r_0} \sim_1 \varepsilon_{r_0}^1$,
with $\gamma_{r_0}^\ell$ prefixes of length r_0 and y_i some process p_0 or p_1, such that $\forall c \in cut\colon c_{r_0} \notin \{\gamma_{r_0}^0, \ldots, \gamma_{r_0}^k\}$ and $\forall c \in cut\colon c_{r_0} \notin \{\varepsilon_{r_0}^0, \sigma_{r_0}\}$. Thus $\forall c \in cut\colon c_{r_0}$ right of $\varepsilon_{r_0}^0$ and, by Lemma 5.4.4 and Definition 5.4.3, for each $r \geq r_0\colon \forall c \in cut\colon c_r$ right of ε_r^0.

For case (i) clearly p_1 and decides x_1 in Line 26, since it is guaranteed that no $c \in cut$ is on the chain from $\leftarrow^{r_0}$ to $\varepsilon_{r_0}^1$. Thus p_1 also decides x_1 in r_0, since it did not decide earlier.

For case (ii) we know, by termination, that there is some round $r_1 \geq r_0$ such that p_1 decides. Suppose p_1 decides x_0, thus it decides in Line 28. Then there has to be a chain:
$\leftarrow^{r_1} = \gamma'^0_{r_1} \sim_{y_0} \gamma'^1_{r_1} \sim_{y_1} \ldots c'_{r_1} \cdots \sim_{y_{k-1}} \gamma'^\ell_{r_1} = \varepsilon_{r_1}^0 \sim_0 \sigma_{r_1} \sim_1 \varepsilon_{r_1}^1$.

The fact that in this chain c'_{r_1} is left of $\varepsilon_{r_1}^0$ contradicts the fact that c'_{r_0} is right of $\varepsilon_{r_0}^0$, by Lemma 5.4.4.

Case 2: p_0 decides x_0 in round r_0.
Suppose in contradiction that p_1 decides x_1 in round r_1, $r_0 \leq r_1$. In Algorithm 5.1, p_0 decides x_0 in Line 28 if it did not decide earlier. This is only the case if $\forall c \in cut\colon c \notin$

$\{\sigma_{r_0}, \varepsilon^0_{r_0}\}$ and there is a chain:
(i)$\leftarrow^{r_0}= \gamma^0_{r_0} \sim_{y_0} \gamma^1_{r_0} \sim_{y_1} \dots c_{r_0} \dots \sim_{y_{k-1}} \gamma^k_{r_0} = \varepsilon^0_{r_0} \sim_0 \sigma_{r_0} \sim_1 \varepsilon^1_{r_0}$ or
(ii)$\leftarrow^{r_0}= \gamma^0_{r_0} \sim_{y_0} \gamma^1_{r_0} \sim_{y_1} \dots c_{r_0} \dots \sim_{y_{k-1}} \gamma^k_{r_0} = \varepsilon^1_{r_0} \sim_1 \sigma_{r_0} \sim_0 \varepsilon^0_{r_0}$.

For case (i) p_1 clearly decides x_0 in Line 28, since there is a $c \in \mathit{cut}$, such that $c \in \{\gamma^0_{r_0}, \dots, \gamma^k_{r_0}\}$.

For case (ii) we know, by termination, that there is some round $r_1 \geq r_0$ such that p_1 decides. Suppose p_1 decides x_1. Then it passed Line 25 and decided in Line 26, because there is a chain:
$\leftarrow^{r_1}= {\gamma'}^0_{r_1} \sim_{y_0} {\gamma'}^1_{r_1} \sim_{y_1} \dots \sim_{y_{k-1}} {\gamma'}^\ell_{r_1} = \varepsilon^1_{r_1} \sim_1 \sigma_{r_1} \sim_0 \varepsilon^0_{r_1}$
such that there is no $\forall c \in \mathit{cut}$: $c_{r_1} \notin \{{\gamma'}^0_{r_1}, \dots {\gamma'}^\ell_{r_1}\}$. The fact that there is no such c contradicts chain (ii) and Lemma 5.4.4, since there is a graph sequence $c \in \mathit{cut}$ such that c_{r_0} is left of $\varepsilon^1_{r_0}$. □

This completes the correctness proof of Algorithm 5.1. Finally, we will present our main theorem: That the restrictions, met by message adversary MA_α are indeed necessary and sufficient for solving consensus. We start with the following technical lemma:

Lemma 5.4.13. *Pick an arbitrary initial configuration C. Let $\sigma, \varepsilon \in MA_\alpha$ and assume p_0, p_1 have decided in the executions (C, σ_t) and (C, ε_t). If there is an indistinguishability chain from σ_t to ε_t consisting of feasible prefixes ρ^i_t, i.e., $\rho^i_t \neq \delta_t$ for all $\delta \in \alpha$, then, in a correct consensus algorithm $\mathcal{A}$, p_0, p_1 have decided the same value in (C, σ_t) and (C, ε_t).*

Proof. Suppose this is not the case. Then the decision values on (C, σ_t) and (C, ε_t) differ from each other, i.e., $\mathit{dec}(C, \sigma_t) \neq \mathit{dec}(C, \varepsilon_t)$.

We proceed by induction on $\ell \geq 1$ to show that then there is a sequence $V_1, \dots, V_\ell$ of non-empty sets of feasible graph sequence prefixes of length r_ℓ where at least one process has not decided and V_i contains all the prefixes of the members of V_{i+1}. By König's Lemma, this contradicts that $\mathcal{A}$ is a correct consensus algorithm.

Consider the graph sequences $\gamma(0) = \sigma$ and, for $\ell \geq 1$, $\sigma(\ell) = \gamma(\ell - 1)$. In the following we will show by induction on $\ell \geq 1$, that there are graph sequences $\gamma(\ell)$, such that in $(C, \gamma(\ell)_{r_\ell})$ at least one process has not decided. We further set $V_\ell = \{\gamma(\ell)_{r_\ell}\}$ and for $0 < i < \ell$ add $\gamma(\ell)_{r_i}$ to V_i. As there are only finitely many prefixes of each length r_i, $|V_{r_i}|$, and thus the node degree, is finite.

For the base case $\ell = 1$, let $r_1 = t$. Suppose there is no graph sequence $\gamma(1)$, such that at least one process has not decided in $(C, \gamma(1)_{r_1})$. Then, by the properties of the SST, there are three prefixes $\rho^{i-1}_{r_1} \sim_{p_i} \rho^i_{r_1} \sim_{p_j} \rho^{i+1}_{r_1}$, with $i, j \in \{p_0, p_1\}, i \neq j$ such that $\mathit{dec}(C, \rho^{i-1}_{r_1}) = \mathit{dec}(C, \sigma_{r_1})$ and $\mathit{dec}(C, \rho^{i+1}_{r_1}) = \mathit{dec}(C, \varepsilon_{r_1})$. Due to indistinguishability p_i has decided $\mathit{dec}(C, \sigma_{r_1})$ on $(C, \rho^i_{r_1})$, while p_j has decided $\mathit{dec}(C, \varepsilon_{r_1})$ on $(C, \rho^i_{r_1})$. This contradicts the supposition that $\mathcal{A}$ solves consensus.

The induction step follows since, for all admissible executions $(C, \gamma(\ell))$, there is a round $r_{\ell+1}$ such that both processes have decided in $(C, \gamma(\ell)_{r_{\ell+1}})$. Otherwise $\mathcal{A}$ would not solve consensus for MA_α. Note that we don't know when the processes decide in $(C, \gamma(\ell))$, thus $r_{\ell+1}$ is not necessarily equal to $r_\ell + 1$. $\square$

Lemma 5.4.14. *Let $\gamma \in S$ be an unfair sequence and $\sigma, \varepsilon \in MA_\alpha$ such that σ is left of γ, ε is right of γ and, for any round r, γ_r is reachable from σ_r and ε_r via an indistinguishability chain of prefixes that does not include any δ_r with $\delta \in S \setminus \{\gamma\}$. Pick any input assignment C. If p_0, p_1 have decided in the executions (C, σ_t) and (C, ε_t) of a correct consensus algorithm $\mathcal{A}$, then they have decided on the same value.*

Proof. By definition of unfair graph sequences, and as there is an indistinguishability chain from σ_t to ε_t not containing any prefix δ_t of some $\delta \in S \setminus \{\gamma\}$, there is a feasible unfair sequence γ' with $\gamma_r \sim_{p_i} \gamma'_r$ for all rounds r and some process p_i. W.l.o.g. let γ' be left of γ. Since $\gamma' \in MA_\alpha$, there is a round s such that both p_0, p_1 have decided some value v in (C, γ'_s). Lemma 5.4.13 shows that the decision of (C, γ'_s) and (C, σ_t) is on the same value v. Thus p_i decides v also on (C, γ_s).

By the structure of the SST, we can extend γ_s to a fair sequence γ'' to the right of γ, since its forever indistinguishable counterpart γ' is to its left. Since p_i decided v on $(C, \gamma_s = \gamma''_s)$ and since $\mathcal{A}$ solves consensus, there is a round u such that also p_j have decided v on (C, γ''_u).

Again by invoking Lemma 5.4.13 we conclude that both processes also decided v on (C, ε_t).

The case where γ' is right of γ is analogous: Now there is a fair extension γ'' of γ_s left of γ. The decision must be the same on (C, ε_t) and (C, γ'_s) as well as on (C, σ_t) and (C, γ''_u). $\square$

Theorem 5.4.3. *Consensus is impossible under MA_α if S contains no fair graph sequence and no pair of unfair graph sequences.*

Proof. Under the input assignment C where $x_0 = 0$ and $x_1 = 1$, by validity and termination, p_0, p_1 have decided 0 in $(C, \rightarrow^t)$ and 1 in $(C, \leftarrow^t)$ for some round t. Since S includes no fair sequence, both executions are admissible. Since the path from $\rightarrow^t$ to $\leftarrow^t$ contains no fair execution nor a pair of unfair executions, repeated application of Lemma 5.4.14 shows that the same decision is reached in $(C, \rightarrow^t)$ and $(C, \leftarrow^t)$, a contradiction. $\square$

Theorem 5.4.4 (Necessary and Sufficient Consensus Message Adversary)**.** *Consensus is solvable under a message adversary $MA_\alpha = MA_u \setminus S$ if and only if S is a set of graph sequences, such that*

- $\exists \sigma \in S\colon$ *σ is a fair graph sequence, or*
- $\exists \varepsilon, \varepsilon' \in S\colon$ *ε, ε' are unfair graph sequences such that $\exists p\colon \forall r\colon \varepsilon_r \sim_p \varepsilon'_r$.*

Proof $\Rightarrow$. See Theorem 5.4.3. $\square$

Proof $\Leftarrow$. By Definition 5.4.8, MA_α fulfills exactly the conditions. Since Algorithm 5.1 solves consensus under MA_α, which is proven in Theorem 5.4.2, consensus is solvable if a message adversary MA_α fulfills the given properties. $\square$

CHAPTER 6

Conclusions and Open Questions

To solve consensus in directed dynamic networks, n synchronous processes connected via unreliable directed links shall decide on a common value. The unreliability of the communication links is modeled by a message adversary, which determines whether or not a message sent by a process in a round is delivered. One of the unsolved questions in this area is to precisely characterize a strongest message adversary: The strongest message adversary is the least restricted message adversary under which deterministic consensus is solvable.

This thesis develops the first steps of two very different approaches to answer this question. The first one combines the knowledge-based approach with communication complexity in the following way: First, determine the knowledge a process has to gather to decide a value. Second, use this knowledge to compute a lower bound on the number of messages a process has to receive, and finally, use this communication complexity to find a message adversary which allows to send exactly the needed bits.
We developed an important ingredient for such an approach in Chapter 4, where we established a connection between *Action Models*, describing the actions observed by the processes in some round and used to update the epistemic state of the system, and communication complexity: Theorem 4.3.1 proves a lower bound on the number of bits, which have to be received during the application of an action model. Furthermore, we reduced deterministic consensus to distributed function computation and developed two different methods for determining the number of bits that have to be received by algorithms computing a function $f(x, y)$ using two processes with input x resp. y.

Once this lower bound had been proved, we tried to use this connection to establish a lower bound on communication complexity for solving consensus in a directed dynamic network with 2 processes. During our investigations, however, we encountered the problem that there is a message adversary $\Diamond\text{STABLE}_D(2)$, which allows consensus to be solved but for which the communication complexity is unbounded (Theorem 5.3.1). Thus, it is impossible to give a lower bound on the communication complexity for consensus in directed dynamic networks in general.

Nevertheless, we found necessary and sufficient conditions for a message adversary that allows consensus for $n = 2$ processes to be solved by using a different approach. It turns out that such a message adversary has to exclude certain graph sequences (at least one fair or a pair of unfair ones), which effectively breaks the otherwise existing indistinguishability chain between all sequences. We defined a message adversary MA_α, which exactly fulfills those conditions, and developed an algorithm that solves consensus under this adversary. Moreover we proved that any message adversary that does not fulfill these conditions does not allow to solve consensus.

Open Questions There are two main open questions remaining:

- We found a method to determine a lower bound on communication complexity for an algorithm computing a function $f(x, y)$ in the 2-process case. As already discussed at the end of Chapter 4, there are two important open tasks:
  - Extend the method to the n-process case ($n > 2$).
  - Extend the method for dynamic networks under general message adversaries.
- We found necessary and sufficient conditions for solving consensus in a dynamic network with 2 processes.
  - Can these conditions be redefined to be used to get such conditions for the system with n processes?
  - What exactly are such conditions for the n process system?

List of Figures

List of Algorithms

Bibliography

[ADG84] Chagit Attiya, Danny Dolev, and Joseph Gil. Asynchronous byzantine consensus. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 119–133, New York, NY, USA, 1984. ACM.

[ALNR09] Natasha Alechina, Brian Logan, Hoang Nga Nguyen, and Abdur Rakib. Verifying time, memory and communication bounds in systems of reasoning agents. *Synthese*, 169(2):385–403, 2009.

[Aum76] Robert J Aumann. Agreeing to disagree. *The annals of statistics*, pages 1236–1239, 1976.

[Bal99] Alexandru Baltag. A logic of epistemic actions. In *Electronic) Proceedings of the FACAS workshop, held at ESSLLI'99*, 1999.

[BM04] Alexandru Baltag and Lawrence S. Moss. Logics for epistemic programs. *Synthese*, 139(2):165–224, 2004.

[BMS98] Alexandru Baltag, Lawrence S. Moss, and Slawomir Solecki. The logic of public announcements, common knowledge, and private suspicions. In *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '98, pages 43–56, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[Bra84] Gabriel Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 154–162, New York, NY, USA, 1984. ACM.

[BRS12] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in directed dynamic networks. In *Proceedings of the 19th International Conference on Structural Information and Communication Complexity*, SIROCCO'12, pages 73–84, Berlin, Heidelberg, 2012. Springer-Verlag.

[BRS$^+$15] Martin Biely, Peter Robinson, Ulrich Schmid, Manfred Schwarz, and Kyrill Winkler. *Gracefully Degrading Consensus and k-Set Agreement in Directed*

Dynamic Networks, pages 109–124. Springer International Publishing, Cham, 2015.

[BT83] Gabriel Bracha and Sam Toueg. Resilient consensus protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 12–26, New York, NY, USA, 1983. ACM.

[BZM14] Ido Ben-Zvi and Yoram Moses. Beyond lamport's happened-before: On time bounds and the ordering of events in distributed systems. *J. ACM*, 61(2):13:1–13:26, April 2014.

[CBS09] Bernadette Charron-Bost and André Schiper. The Heard-Of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, April 2009.

[CFQS12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

[CG13] Étienne Coulouma and Emmanuel Godard. A characterization of dynamic networks where consensus is solvable. In *Proceedings Structural Information and Communication Complexity - 20th International Colloquium (SIROCCO'13), Springer LNCS 8179*, pages 24–35, 2013.

[CGM13] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. Good, better, best! - unbeatable protocols for consensus and set consensus. *CoRR*, abs/1311.6902, 2013.

[CK08] Aiswarya Cyriac and K. Murali Krishnan. Lower bound for the communication complexity of the russian cards problem. *CoRR*, abs/0805.1974, 2008.

[DF89] Danny Dolev and Tomás Feder. Multiparty communication complexity. In *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 428–433. IEEE, 1989.

[DFF$^+$82] Danny Dolev, Michael J Fischer, Rob Fowler, Nancy A Lynch, and H Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[DKW09] Jan Draisma, Eyal Kushilevitz, and Enav Weinreb. Partition arguments in multiparty communication complexity. *CoRR*, abs/0909.5684, 2009.

[DMR08] Yefim Dinitz, Shlomo Moran, and Sergio Rajsbaum. Bit complexity of breaking and achieving symmetry in chains and rings. *J. ACM*, 55(1):3:1–3:28, February 2008.

[FHMV95] R. Fagin, J.Y. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge.* MIT Press, 1995.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[Ger97] Jelle Douwe Gerbrandy. *Dynamic epistemic logic.* Institute for Logic, Language and Computation (ILLC), University of Amsterdam, 1997.

[GG97] Jelle Gerbrandy and Willem Groeneveld. Reasoning about information change. *Journal of Logic, Language and Information*, 6(2):147–169, 1997.

[HH89] Jaakko Hintikka and Merrill B. Hintikka. *Reasoning about Knowledge in Philosophy: The Paradigm of Epistemic Logic*, pages 17–35. Springer Netherlands, Dordrecht, 1989.

[Hin62] J. Hintikka. *Knowledge and belief: an introduction to the logic of the two notions.* Contemporary philosophy. Cornell University Press, 1962.

[HM90] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. (3):549–587, 1990.

[KN97] Eyal Kushilevitz and N. Nisan. *Communication Complexity.* Cambridge University Press, 1997.

[KOM11] Fabian Kuhn, Rotem Oshman, and Yoram Moses. Coordinated consensus in dynamic networks. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '11, pages 1–10, New York, NY, USA, 2011. ACM.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[Lew69] David Lewis. *Convention: A philosophical study.* Harvard University Press, 1969.

[LS81] Richard J. Lipton and Robert Sedgewick. Lower bounds for vlsi. In *Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, STOC '81, pages 300–307, New York, NY, USA, 1981. ACM.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[MDH86] Yoram Moses, Danny Dolev, and Joseph Y. Halpern. Cheating husbands and other stories: A case study of knowledge, action, and communication. *Distributed Computing*, 1(3):167–176, 1986.

[Moo80] Robert C Moore. *Reasoning about knowledge and action.* SRI International Menlo Park, CA, 1980.

[MS82] Kurt Mehlhorn and Erik M. Schmidt. Las vegas is better than determinism in vlsi and distributed computing (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, pages 330–337, New York, NY, USA, 1982. ACM.

[Pla89] Jan Plaza. Logics of public communications. In *Proceedings of the 4th International Symposium on Methodologies for Inteligent Systems*, pages 201–2016, 1989.

[Pla07] Jan Plaza. Logics of public communications. *Synthese*, 158(2):165–179, Sep 2007.

[SW89] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proc. 6th Annual Symposium on Theor. Aspects of Computer Science (STACS'89)*, LNCS 349, pages 304–313, Paderborn, Germany, February 1989. Springer-Verlag.

[SWK09] Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility results and lower bounds for consensus under link failures. Technical report, 2009.

[SWS16] Manfred Schwarz, Kyrill Winkler, and Ulrich Schmid. Fast consensus under eventually stabilizing message adversaries. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 7:1–7:10, New York, NY, USA, 2016. ACM.

[vD99] Hans van Ditmarsch. The logic of knowledge games: Showing a card, 1999.

[vD02] Hans P. van Ditmarsch. Descriptions of game actions. *Journal of Logic, Language and Information*, 11(3):349–365, 2002.

[vDvdHK08] Hans van Ditmarsch, Wiebe van der Hoek, and Berteld Kooi. *Dynamic Epistemic Logic.* Springer, 2008.

[WSS16] Kyrill Winkler, Manfred Schwarz, and Ulrich Schmid. Consensus in directed dynamic networks with short-lived stability. *CoRR*, abs/1602.05852, 2016.

[Yao79] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM.