On Knowledge and Communication Complexity in Distributed Systems

Daniel Pfleger and Ulrich Schmid

TU Wien, Treitlstrasse 3, 1040 Vienna, Austria {dpfleger,s}@ecs.tuwien.ac.at

Abstract. This paper contributes to exploring the connection between epistemic knowledge and communication complexity in distributed systems. We[1] focus on Action Models, a well-known variant of dynamic epistemic logic, which allows to cleanly separate the state of knowledge of the processes and its update due to communication actions: Exactly like the set of possible global states, the possible actions are described by means of a Kripke model that specifies which communication actions are indistinguishable for which process. We first show that the number of connected components in the action model results in a lower bound for communication complexity. We then apply this result, in the restricted setting of a two processor system, for determining communication complexity lower bounds for solving a distributed computing problem $\mathcal{P}$: We first determine some properties of the action model corresponding to any given protocol that solves $\mathcal{P}$, and then use our action model communication complexity lower bounds. Finally, we demonstrate our approach by applying it to synchronous distributed function computation and to a simple instance of consensus in directed dynamic networks.

Keywords: distributed systems, dynamic epistemic logic, communication complexity

1 Introduction

Our paper is concerned with the idea to infer the communication complexity for solving a general distributed computing problem $\mathcal{P}$ from the epistemic knowledge that must be attained by the processes to solve $\mathcal{P}$. More specifically, we take a first step to bridge Hintikka's *epistemic logic* [14], variants of which have very successfully been applied in distributed computing already [3,4,13], and Yao's communication complexity [23]. In this seminal work, Yao introduced methods for deriving communication complexity lower bounds for distributed function computation in a system of 2 processes.

Epistemic logic [14] allows to formally reason about knowledge and belief in static multi-agent systems. It relies on a Kripke model M that describes the possible global states ("possible worlds") of the system, where certain atomic propositions (facts like "variable x_i of process p_i is zero") hold true or not, along with an indistinguishability relation $s \sim_a s'$ between global states s, s' for every agent (= process) a. Knowledge of some fact ϕ about the system in global state s is primarily captured by a modal *knowledge operator* K_a, used in formal expressions like $M, s \models K_a\phi$, which captures the intuition that, being in the global state s, process a knows ϕ if ϕ holds in every global state s' that is indistinguishable from s for a. We will use the term *epistemic model* to refer to this type of Kripke models, which focus entirely on a given "static" knowledge state of the processes.

Dynamic Epistemic Logic (DEL) [8, 12] also allows to incorporate communication-induced knowledge gain into the formal reasoning. We will focus on a variant of DEL called *Action Models* [8], which are particularly suitable for our purpose. Action Models can be used to describe possible communication events that may occur at certain times in an execution of some algorithm. Formally, this is modeled by applying a sequence of (arbitrary) communication *action models* $\mathtt{AM}_1, \mathtt{AM}_2, \ldots$

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

to an initial epistemic model M_0, which results in a sequence of epistemic models $M_0, M_1, \ldots$ that describes the evolution of knowledge in the execution. Every $\mathtt{AM}_k$ is represented by an *independent* Kripke model here, which is orthogonal to the epistemic model M_{k-1} it is applied to. By means of a well-defined product $\otimes$ (see Sec. 2.1), this leads to the epistemic model $M_k = M_{k-1} \otimes \mathtt{AM}_k$. This abstraction is particularly suitable for modeling synchronous systems, as $\mathtt{AM}_k$ can be used to express all the possible communication in round k here.

Our contributions: (1) We first exploit the clean separation of epistemic models and action models to infer a natural lower bound for the number of bits that some process a must receive in any protocol that faithfully implements a given action model $\mathtt{AM}$. It is closely related to the number of partitions in a's indistinguishability relation in $\mathtt{AM}$. (2) Restricting our attention to systems of 2 processes, we then infer a communication complexity lower bound for solving a problem $\mathcal{P}$ by (i) determining the properties of the action model corresponding to any protocol that correctly solves $\mathcal{P}$, and (ii) inferring a communication complexity lower bound from this via the result of (1). We apply our approach both to distributed function computation in synchronous systems [23] and to consensus under message adversaries [1, 5, 22].

Related work: Van Ditmarsch et. al. [8] provide a comprehensive introduction into Dynamic Epistemic Logic, including Action Models. Fagin et. al. [11] introduces the powerful runs and systems framework, which allows to reason about knowledge in general distributed systems. Halpern and Moses [13] use this framework to reason the role of (various forms of) common knowledge in solving distributed consensus. They also elaborate on the *Muddy Children* problem, which is very closely related to the *Cheating Husbands* problem [20] that we use for illustrating Action Models in Sec. 4.

Communication complexity lower bounds deserve much to the seminal work [23] by Yao, which studies distributed function computation for two processes. Among the techniques used for deriving lower bounds is the *fooling set* method [18], which became quite popular, see e.g. [7]. The alternative *rank lower bound* technique has been introduced in [19]. Among many other works, [9] and [10] generalized the two-player setting to multi-party communication complexity. Indeed, Yao's paper [23] has also sparked quite some interest in the distributed systems community, which led to very interesting lower bounds based on information-theoretic. A few examples, among many possible others, are symmetry breaking in chains and rings [7] and lower bounds for all pair shortest paths [15]. However, unlike our results, these approaches are usually tied to the specific problem $\mathcal{P}$ at hand and do not use epistemic logic.

We are not aware of much work on the relation between communication complexity and epistemic models. Somewhat similar to our work is [6], 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. Alechina et. al. [2] 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. Since the communication complexity in this work is defined as the number of formulas, rather than the number of bits, however, it cannot be compared to our approach.

Paper organization: We start by defining our system model in Sec. 2, followed by an introduction to the relevant basics of Action Models (Sec. 2.1) and Communication Complexity (Sec. 3). In Sec. 4, we demonstrate how Action Models work by means of the well-known Cheating Husbands problem [20], and explain the connection between communication complexity and the number of partitions in action models. Sec. 5 elaborates further on the connection between Action Models and *Protocol Trees* introduced in [23]. Our main results can be found in Sec. 5.2, along with two applications in Sec. 5.3 (details for consensus in directed dynamic networks had to be relegated

to Appendix A, however). Some conclusions and directions of future work in Sec. 6 round off our paper.

2 Model

We consider synchronous message passing systems only. Such systems are modeled as a set Π of n processes with unique identifiers, which are reliable and operate in lock-step rounds $r = 1, 2, \ldots$. The processes are modeled as state machines and connected by point-to-point communication links. We consider both reliable links and unreliable links controlled by a message adversary [1, 5, 22], which determines the links that successfully deliver the message sent over it in a round. More specifically, at the beginning of round r, all processes send out a message to every other process (and to themselves). Rounds are communication-closed, in the sense that each message sent in round r can only be delivered in r. The message adversary determines which message is indeed be delivered to the intended receiver. After this message exchange, all processes simultaneously perform an instantaneous local computation step that terminates round r.

Note that, in the case of reliable links, the guarantee that all the messages sent in round r will be delivered by the end of round r allows *communication by time*: if a process a did not receive the message from process b by the end of round r, then a *knows* that b did not send this message. Conversely, if a sends a message to b at the beginning of round r, a *knows* at the end of round r that b received this message. Note that Ben-Zvi and Moses [4] modeled communication by time via explicitly sending a virtual NULL-message instead of a real one.

Regarding the connection of action models and the synchronous model, we assume that a single action model $\mathtt{AM}_r$ is applied in each round r. For every possible communication pattern in round r, which is determined (i) by the protocol and (ii) by the message adversary, it contains a corresponding action. Clearly, two actions are indistinguishable for process a, if it receives the same messages. Note carefully that the synchrony assumption makes sure that every process knows that the action model $\mathtt{AM}_r$ is to be applied, even if it does not receive a single message in round r.

2.1 Knowledge and Action Models

Usually, distributed computing problems also involve global constraints, like agreement in distributed consensus (see Sec. 5.3). The actions of a single process in a distributed 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 typically 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, frameworks like [8, 11] allow to formally reason about knowledge in such systems. We utilize a variant of Dynamic Epistemic Logic [12, 21], namely, Action Models, which are well-suited for the simple synchronous systems considered in our paper. The following Def. 1 to 4 and 7 will define epistemic models and action models, as well as the semantics of action model logic. Illustrating examples can be found in Sec. 4 (Fig. 2 and Fig. 3).

Definition 1 (Kripke model, see [8], Definition 2.6) *A* Kripke model M *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*.* $V : P \to 2^S$*,* P *a set of atomic propositions (also called atoms), is a valuation function for each proposition* $p \in P$*. For any proposition* $p \in P$*,* $V(p) \subseteq S$ *is exactly the set of states in which* p *is true.*

In our context, the epistemic states $s \in S$, denoted (M, s), are the possible global states of the distributed system, and R_a is interpreted as an indistinguishability relation for process a, thus is denoted by $\sim_a$ in the sequel.

The following definition formally defines the Kripke model of the possible actions. It is independent of the epistemic model in Def. 1 it is applied to, except for the precondition function that governs which actions are applicable in which epistemic state.

Definition 2 (Action Model, see [8], 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 following syntax is used to formally specify action model knowledge formulas. Basically, it consists of formulas ϕ related to the epistemic state (the "possible worlds") of the system, and formulas involving the application of some action α. The detailed semantics is given in Def. 7 below.

Definition 3 (Syntax of action model logic, see [8], Definition 6.3) *Given processes A and atoms P, the language $\mathcal{L}_{\mathcal{KC}\otimes}(A, P)$ is the union of* formulas *$\phi \in \mathcal{L}^{stat}_{\mathcal{KC}\otimes}(A, P)$ and* pointed action models *$\alpha \in \mathcal{L}^{act}_{\mathcal{KC}\otimes}(A, P)$ defined by:*

$\phi ::= p \mid \neg\phi \mid (\phi \wedge \phi) \mid K_a\phi \mid E_B\phi \mid C_B\phi \mid [\alpha]\phi$

$\alpha ::= (\mathtt{M}, \mathtt{s}) \mid (\alpha \cup \alpha)$

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. $\alpha \cup \alpha'$ denotes a non-deterministic choice *between α and α'.*

Action models can be composed: To apply two different actions $(\mathtt{M}, \mathtt{s})$ and $(\mathtt{M}', \mathtt{s}')$ to some epistemic state (M, s) subsequently, one can either apply them one after the other or determine their composition $(\mathtt{M}'', \mathtt{s}'') = (\mathtt{M}; \mathtt{M}', (\mathtt{s}, \mathtt{s}'))$ and apply the resulting action $(\mathtt{M}'', \mathtt{s}'')$ in a single step.

Definition 4 (Composition of action models, see [8], 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:*

$\mathtt{S}'' = \mathtt{S} \times \mathtt{S}'$

$(\mathtt{s}, \mathtt{s}') \sim''_a (\mathtt{t}, \mathtt{t}')$ *iff* $\mathtt{s} \sim_a \mathtt{t}$ *and* $\mathtt{s}' \sim'_a \mathtt{t}'$

$\mathtt{pre}''((\mathtt{s}, \mathtt{s}')) = \langle \mathtt{M}, \mathtt{s} \rangle \mathtt{pre}'(\mathtt{s}')$

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

To change the static epistemic status starting in an epistemic model M, one applies an action model $\mathtt{M}$, resulting in a new epistemic modem M':

Definition 5 (Application of an action model) *We define the application of action model $\mathtt{M}$ on epistemic model M, resulting in M', $M' = (M \otimes \mathtt{M})$, as $M' = \langle S', \sim', V' \rangle$ with:*

$S' = \{(s, \mathtt{s}) \mid s \in S, \mathtt{s} \in \mathtt{S}, \text{and } M, s \models \mathtt{pre}(\mathtt{s})\}$

$(s, \mathtt{s}) \sim'_a (t, \mathtt{t})$ *iff* $s \sim_a t$ *and* $\mathtt{s} \sim_a \mathtt{t}$

$(s, \mathtt{s}) \in V'(p)$ *iff* $s \in V(p)$

Note that our complexity results will primarily rely on the axiom $(s, \mathtt{s}) \sim'_a (t, \mathtt{t})$ iff $s \sim_a t$ and $\mathtt{s} \sim_a \mathtt{t}$, which implies that the application of two *distinguishable* actions $\mathtt{s} \not\sim_a \mathtt{t}$ to indistinguishable epistemic states $s \sim_a t$ causes distinguishable epistemic states $(s, \mathtt{s}) \not\sim'_a (t, \mathtt{t})$.

To define the semantics of common knowledge, the last ingredient of our Action Models, we need to introduce the *reflexive transitive closure* of a relation R. It allows to express facts and formulas ϕ that are commonly known to a subset of the processes, in the sense that "every process knows that every process knows that every process knows ... ϕ.". We define "everybody in group B knows ϕ" ($E_B\phi$), as a syntactic equivalence $E_B\phi = \bigwedge_{a\in B} K_a\phi$.

Definition 6 *The* reflexive transitive closure *of a relation R is the smallest relation R^* such that: (i) $R \subseteq R^*$, (ii) for all x, y, and z: $xR^*y \wedge yR^*z \Rightarrow xR^*z$ (transitivity), (iii) for all x, xR^*x (reflexivity).*

We can now give the formal semantics of Action Model logic. It specifies the meaning of both the operations for reasoning about knowledge in the epistemic model and the application of action models.[2]

Definition 7 (Semantics of action model logic, see [8], 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 $\phi \in \mathcal{L}^{stat}_{\mathcal{KC}\otimes}$ and $\alpha \in \mathcal{L}^{act}_{\mathcal{KC}\otimes}$. Furthermore let A be a set of processes and P a set of atoms, while $a \in A$, $B \subseteq A$ and $p \in P$.*

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

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

In Sec. 4, we will use the Cheating Husbands problem [20] to exemplify how the Action Model semantics works in practice; particular instantiations can be found in Sec. 5.3.

3 Communication Complexity Basics

In [23], Yao considers two processes p_0 and p_1, which jointly solve the problem of evaluating the non-constant function $f : X \times Y \to Z$, where X, Y and Z are arbitrary finite sets. Herein, the input $x \in X$ is only known to p_0, whereas the input $y \in Y$ is only known to p_1. Clearly, p_0 and p_1 have to communicate with each other in order to solve the problem. Yao's communication model assumes that all communication links are reliable and 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 of computing f is the least number of bits 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. In fact, Yao assumes that the process that can compute $f(x, y)$ first sends a special NULL message to the other process and stops. It is also assumed that the processes both know the identity of themselves and the other process a priori. Note carefully that this allows the design of an asymmetric protocol, where some agreed-upon process, say, p_0 sends the first bit.

[2] Please observe the different fonts in our notation: in $s \sim_a t$, $\sim_a$ is taken from the epistemic model M, while in $\mathtt{s} \sim_a \mathtt{t}$, $\sim_a$ is from the action model $\mathtt{M}$.

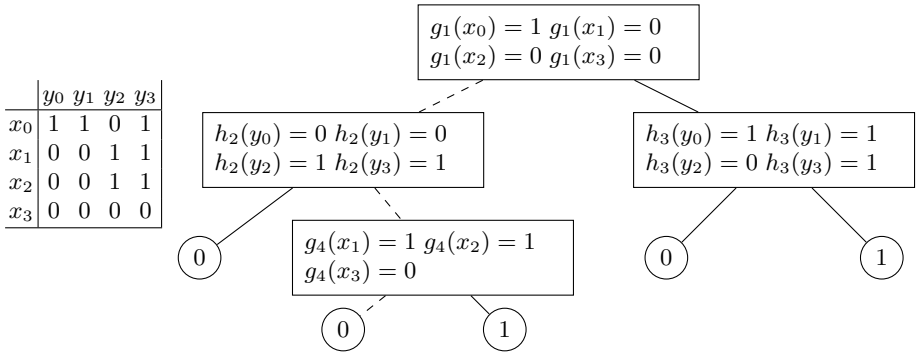
The following definition introduces the convenient notion of *protocol trees*, which uniquely describe the possible executions of a given protocol.

Definition 8 (Protocol trees, see [16], 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* $g_v : X \to \{0,1\}$ *or by a function* $h_v : Y \to \{0,1\}$*, and each leaf is labeled with an element* z *of* Z*. The root* r *is labeled by* $g_1 : X \to \{0,1\}$*. Intuitively,* g_k *(resp.* h_k*) gives the bit sent by* p_0 *(resp.* p_1*) in round* k*.*

The cost of the protocol $\mathcal{P}$ *on input* (x,y) *is the length of the path taken on input* (x,y)*, denoted by* $\mathcal{D}_{\mathcal{P}}(f)$*. 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.*

The cost of a problem f *is the minimal cost of any protocol* $\mathcal{P}$ *that computes* f*, denoted by* $\mathcal{D}(f)$*.*

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 g_v (resp. h_v), to determine the value of the next communicated bit. Fig. 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 by the protocol tree next to it.



	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

Fig. 1: An example of a function $f(x,y)$ and one possible protocol tree for computing it. The dashed path corresponds to some input in $\{x_3\} \times \{y_2,y_3\}$. The left (resp. right) branch from each node v corresponds to sending bit 0 (resp. 1).

As witnessed by the leaves in the protocol tree, a protocol $\mathcal{P}$ can be seen as a way to partition the set of possible inputs $X \times Y$ to multiple subsets leading to the same communication pattern. The left (resp. right) child of a node corresponds to the case where the bit sent by the node's corresponding process is 0 (resp. 1). For example, the sequence of communicated bits for any input pair in $\{x_3\} \times \{y_2,y_3\}$, corresponding to the dashed path in Fig. 1, is $(0,1,0)$. This leads to the crucial notion of *rectangles*:

Definition 9 (Rectangles, see [16], Proposition 1.13) *A partition* $R \subseteq X \times Y$ *is a rectangle iff:* $(x,y) \in R$ *and* $(x',y') \in R \Rightarrow (x,y') \in R$*. A rectangle is* f-monochromatic *iff for all* $(x,y) \in R$ *the result of* $f(x,y) = z$ *is the same.*

By exploiting the close relation between monochromatic rectangles and leaf, one can prove the following Cor. 1 [16, 23]:

Corollary 1 (Lower Bound, see [16], Corollary 1.17) *If any set of* f*-monochromatic rectangles induced by* f *has size at least* t*, then* $\log_2 t \leq D(f)$*.*

In the remaining paper, we will use a similar model as in [23], with the following two main differences: (i) Each process can send an arbitrary number of bits in every round. (ii) We consider symmetric function computation (sometimes without communication by time), i.e., once an algorithm for computing $f(x, y)$ terminates, the result must be commonly known by both p_0 and p_1.

4 Communication Complexity of Action Models

We use the *Cheating Husbands* problem [20] to illustrate the connection between knowledge and communication complexity. Herein, the women of a city ruled by a queen want to get rid of unfaithful husbands. It is common knowledge that each of the women knows the fidelity-status of the husbands of all other women, but does not know whether or not her own husband is unfaithful. The women are not allowed to discuss their husbands fidelity with each other. The left model in Fig. 2 depicts the initial epistemic model M_{CH} for three women (a, b, c). Each state is labeled with the atomic proposition (abc), $a, b, c \in \{0, 1\}$, $i = 1(0)$ interpreted as "husband of i is unfaithful (faithful)".

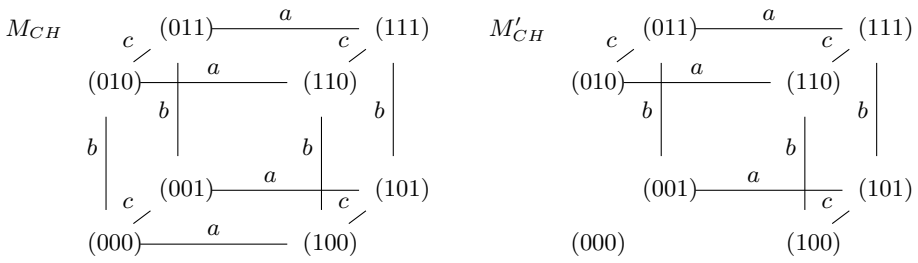


Fig. 2: Example Cheating Husbands: Left: The initial Kripke model M_{CH} for three wives a, b, c. Right: The epistemic model M'_{CH} reached after applying $\mathtt{AM}_{pub}$ or $\mathtt{AM}_{priv}$.

In the original problem considered in [20], it is common knowledge among the women that the queen will publicly announce whether there is at least one unfaithful husband or not. We can model this announcement using the actions $\neg\mathtt{t}$ ("the queen does not announce anything") and ≥ 1 ("the queen publicly announces that there is at least one unfaithful husband"). It is well known [20] that, in this scenario, the women can find all the unfaithful husbands by a synchronous protocol, which requires every woman who gets to know her husband is unfaithful some day must shoot him at midnight.

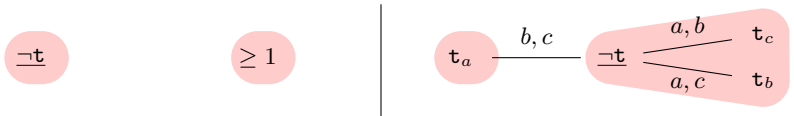
There is a variant of this problem, which also allows a correct solution: Here it is common knowledge that, iff there is exactly one unfaithful husband, the queen tells his wife privately that her husband is unfaithful on some a priori known day. All the other women will never hear anything from the queen, and in no other case the queen announces anything. This can be modeled by the actions $\neg\mathtt{t}$ ("the queen does not announce anything") and t_i for each woman i ("the queen tells woman i that her husband is unfaithful"). It can be shown that the women are also able to shoot all the unfaithful husbands, using the same protocol.

The two action models $\mathtt{AM}_{pub}$ for the public announcement and $\mathtt{AM}_{priv}$ for the private announcement are depicted in Fig. 3. Applying $\mathtt{AM}_{pub}$ and $\mathtt{AM}_{priv}$ on the initial epistemic model, the resulting epistemic model is the same, depicted in Fig. 2 (right). Still, in the public scenario $\mathtt{AM}_{pub}$, the queen "sends out" a single bit ("There is no / at least one unfaithful husband.") to each woman, summing up to n bits in total for n women. In the private scenario $\mathtt{AM}_{priv}$, though, the queen only needs to send a message ("You!") to a single woman in some special cases, and sends nothing in most other

cases. Since we are a synchronous setting, however, every woman knows —via communication by time— that, if the queen sent her a message, she would have received it by midnight of the a priori known day. So, effectively, the queen sends a single bit ("Your husband is unfaithful") to at most one woman. Consequently, the communication complexity in the public scenario is higher than in the private one.

One may conjecture that this difference is related to the information complexity of the *a priori knowledge*: the communication complexity probably decreases with increasing a priori knowledge. Exploring this relation is a very interesting research question but still out of reach.[3] We therefore focus on the relation between communication complexity and the number of possible "knowledge-changing" events in an execution, which are neatly encapsulated in the action models: in essence, an action model just defines the possible observations of the global system state every single process can make.

Considering a single woman a, it is apparent from Fig. 3 that both of the action models are *partitioned* regarding the indistinguishability of a.



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

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

Fig. 3: Action models for the two scenarios of Cheating Husbands. The actions are denoted by: $\neg\mathtt{t}$: the queen does not make a statement, ≥ 1: the queen publicly announces that there is at least one unfaithful husband, $\mathtt{t}_i$: the queens tells i privately that her husband is unfaithful. The partitioning regarding a is depicted in red.

Definition 10 (Partitions of action models) *An Action Model* $\mathtt{AM} = \langle S, \sim, \mathtt{pre}\rangle$ *is* partitioned regarding process a *if the underlying indistinguishability graph, consisting only of edges corresponding to* $\sim_a$*, is partitioned. I.e., there are sets of nodes* V_i*, such that for* $i \neq j$ $V_i \cap V_j = \emptyset$, $\bigcup_i V_i = V$ *and* $v \in V_i$, $v' \in V_j \Rightarrow (v, v') \notin E$.
The number of those partitions is denoted by N^a_{AM}.

Our claim is that there is a strong connection between the communication complexity, more specifically, the number of bits received by process a, and the number of such partitions $N^a_{\mathtt{AM}}$ in the action model $\mathtt{AM}$. In Fig. 3, both action models $\mathtt{AM}_{pub}$ and $\mathtt{AM}_{priv}$ partition into two partitions, for each woman i. Indeed, if the queen does not announce anything in $\mathtt{AM}_{priv}$, woman a does NOT know that the action has been $\neg\mathtt{t}$, she only knows that the action has been in the partition $\{\neg\mathtt{t}, \mathtt{t}_b, \mathtt{t}_c\}$. In Fig. 3, if ≥ 1 (in $\mathtt{AM}_{pub}$) or $\mathtt{t}_a$ (in $\mathtt{AM}_{priv}$) occurs, woman a of course immediately knows the action itself, without receiving any additional information. In the case of $\neg\mathtt{t}$ (in $\mathtt{AM}_{pub}$) or an action other than $\mathtt{t}_a$ (in $\mathtt{AM}_{priv}$), she learns the partition by not receiving anything, i.e., via communication by time.

Since every woman has to be able to identify the actual partition the current action is in, according to the semantics of $\otimes$ in Def. 5, the number of these partitions determines a lower bound on the number of bits received by a woman in some scenario, i.e., a worst-case lower bound: 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.

[3] We note, however, that our findings do support this claim, as the action model is common a priori knowledge and clearly more complex in the private than in the public scenario, cp. Fig. 3.

Note that this does NOT contradict our above observation that, in $\mathtt{AM}_{priv}$, the queen sends a message to at most one woman. In more detail, in action $\mathtt{t}_a$ in $\mathtt{AM}_{priv}$, the queen *actively* sends a $bit = 1$ to woman a. In actions $\neg\mathtt{t}$, $\mathtt{t}_b$, $\mathtt{t}_c$, the queen does not *actively* send anything to woman a, but it does so passively via communication by time: as in [4], we model this by virtually sending a NULL message.

Definition 11 *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 (communication by time).*

Note that multiple active bits can be sent from a to b in a round, while a NULL message counts as a single passive bit only.

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

Definition 12 *The worst-case cost $D^a(\mathtt{AM})$ of the application of an action model $\mathtt{AM}$ for 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 protocol 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 $D^a(\mathtt{AM})$. Of course, this implies that we can only guarantee that $D^a(\mathtt{AM})$ bits are sent in *some* scenario, not in *any* scenario. Even worse, we cannot assume that the action causing the worst-case cost $D^a(\mathtt{AM})$ for process a is also causing the worst-case cost $D^b(\mathtt{AM})$ for process b. Therefore, defining the total worst-case cost $D(\mathtt{AM})$ of the application of an action model $\mathtt{AM}$ as the sum of $D^a(\mathtt{AM})$ 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 $D^a(\mathtt{AM})$:

Lemma 1 *In a synchronous system with processes A, the worst-case cost $D^a(\mathtt{AM})$ of the application of an action model $\mathtt{AM}$ for process $a \in A$ satisfies $\log_2(N^a_{\mathtt{AM}} - 1) \leq D^a(\mathtt{AM})$, where $N^a_{\mathtt{AM}}$ is the number of partitions regarding a in $\mathtt{AM}$.*

Proof (Proof by contradiction). Suppose there exists an action model $\mathtt{AM}$ such that $D^a(\mathtt{AM}) < \log_2(N^a_{\mathtt{AM}} - 1)$ for some process $a \in A$. Then $2^{D^a(\mathtt{AM})} + 1 < N^a_{\mathtt{AM}}$. Obviously, by receiving $D^a(\mathtt{AM})$ active bits with value 0 or 1, a can distinguish at most $2^{D^a(\mathtt{AM})} + 1$ (including the single passive bit) partitions of $\mathtt{AM}$. Since $2^{D^a(\mathtt{AM})} + 1 < N^a_{\mathtt{AM}}$, 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 $(\mathtt{AM}, \mathtt{s})$ at epistemic state (M, s) results in (M', s'), and consider the following scenario: (i) $s_0 \sim_a s_1$ in epistemic model M, (ii) $\mathtt{s}_0 \in P_0$, $\mathtt{s}_1 \in P_1$ in action model $\mathtt{AM}$ applicable to s_0 respectively s_1 ($\mathtt{s}_0 \not\sim_a \mathtt{s}_1$, but P_0 and P_1 indistinguishable by a). Such a scenario always exists, as one can choose $s_1 = s_0$ as well. By the semantics of Action Models (Def. 5 and 7), we must have $(s_0, \mathtt{s}_0) \not\sim_a (s_1, \mathtt{s}_1)$ in epistemic model M'. Since a cannot distinguish between (the actions in) P_0 and P_1, however, we inevitably have $(s_0, \mathtt{s}_0) \sim_a (s_1, \mathtt{s}_1)$, providing the required contradiction. Thus a has to receive at least $\log_2(N^a_{\mathtt{AM}} - 1)$ active bits during the application of $(\mathtt{AM}, \mathtt{s})$.

Lem. 2 provides a lower bound for $D^a(\mathtt{AM})$ in the case in which communication by time cannot be used, e.g., when communication is unreliable. Its proof is almost identical to the proof of Lem. 1.

Lemma 2 *In a system with processes A, the worst-case cost of $D^a(\mathtt{AM})$ of the application of an action model $\mathtt{AM}$ for process $a \in A$ satisfies $\log_2(N^a_{AM}) \leq D^a(\mathtt{AM})$, where N^a_{AM} is the number of partitions regarding a in $\mathtt{AM}$.*

So far, we only considered the application of a single action model. For 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 is to sum up the communication complexity of single round action models. Unfortunately, this would not provide a tight lower bound on the overall communication complexity of $\mathcal{A}$: while the worst-case execution of $\mathcal{A}$ may include the worst-case scenario of some round r action model $\mathtt{AM}_r$, it does not necessarily include the worst-case scenario of action model $\mathtt{AM}_{r'}$ in round r'. Fortunately, however, Def. 4 provides a way to alleviate 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 $D^a(\mathtt{AM})$ given by Def. 12 is tied to the communication complexity for applying a given action model $\mathtt{AM}$, i.e., of an algorithm $\mathcal{A}$ that *faithfully* implements a given $\mathtt{AM}$. Obviously, this is not equivalent to the communication complexity for solving a specific problem, since the lower bounds for $D^a(\mathtt{AM})$ established in Lem. 1 and Lem. 2 are tied to a specific action model. In the following section, we will address the communication complexity of a given problem $\mathcal{P}$, by considering action models that are optimal for $\mathcal{P}$.

5 Action Models and Protocol Trees

In this section, we will restrict our attention to distributed function computation in synchronous systems of 2 processes. Clearly, all actions correspond to messages sent by one of the two processes here, and each action can be distinguished from any other action by both processes at the end of a round.

5.1 Action models of protocol trees

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 $\mathtt{s}$ "a sends x to b" in round r from the action t "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. $\neg x$.[4] On the other hand, since b can distinguish the actions $\mathtt{s}$ and $\mathtt{t}$, 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 $K_b \neg x$.

Note, however, that the fact that all actions are distinguishable for every process is only valid because we have just two processes: In a system with e.g. three processes, it would be possible that 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

[4] To be precise, this is only true if x is a *preserved formula* (as introduced in [8]), which requires x to be propositional or positive knowledge (but not $x = \neg K_a \phi$, for example). Thus we will also restrict ourselves to algorithms in which preconditions of actions only involve preserved formulas, which is essentially a non-restriction for distributed algorithms.

that are separated for *all* processes: Each such partition consists of (potentially multiple) epistemic states in which the result of f must be 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 input value), the initial epistemic model $M = (S, \sim, V)$ is not partitioned, but rather a hypercube like in Fig. 2. The terminal epistemic model is the result of applying the composed action model (for all rounds) to the initial epistemic model. Thus, the required partitioning of the terminal epistemic model can only result from some partitioning of the composed action model.

Definition 13 *An algorithm $\mathcal{A}$ is defined by a set of action models $\{\mathit{AM}_1, \ldots, \mathit{AM}_k\}$, such that a single action model AM_i is applicable in round i of the synchronous execution. The action model AM_i partitions into $t_{\mathit{AM}_i} \geq 1$ disjoint partitions (identically for both processes p_0, p_1).*

Definition 14 *The composed action model of the first k rounds (CAM_k) is the composition of the action models $\mathit{AM}_1, \ldots, \mathit{AM}_k$, inductively defined as $\mathit{CAM}_1 = \mathit{AM}_1$ and $\mathit{CAM}_k = (\mathit{CAM}_{k-1}; \mathit{AM}_k)$. Every $\mathit{CAM}_k = (S_{\mathit{CAM}_k}, \sim_{\mathit{CAM}_k}, \mathit{pre}_{\mathit{CAM}_k})$ partitions into t_{CAM_k} disjoint partitions, where $P_{k,i}$ denotes the i-th partition of CAM_k, consisting of actions $S_{k,i} \subseteq S_{\mathit{CAM}_k}$.*

Clearly, if $\mathcal{A}$ computes f in m rounds, the relevant composed action model is $\mathtt{CAM}_m$. Observe 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 $M' = M \otimes \mathtt{CAM}_k$, as required.

We can now define the protocol tree corresponding to the action model for algorithm $\mathcal{A}$:

Definition 15 *The protocol tree $\mathcal{T}_{\mathcal{A}} = (P, E)$ of an algorithm $\mathcal{A}$, starting at the root vertex v that represents the initial epistemic model $M = (S, \sim, V)$, is defined as:*
$P = \{v\} \cup \{P_{k,i} \mid P_{k,i}$ for some i is a partition of CAM_k, $k \in \{1, m\}\}$
$E = \{(v, P_{1,j}) \mid P_{1,j}$ a partition of $\mathit{CAM}_1\} \cup \{(P_{k,i}, P_{k+1,j}) \mid \exists s \in S_{k,i}, t \in S_{\mathit{AM}_{k+1}} : (s, t) \in S_{k+1,j}$ for some i and j, and $k \in \{1, m-1\}\}$.

Informally, Def. 15 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 $\mathtt{P}_{k+1,j}$ on levels k and $k+1$ if and only if there is an action $\mathtt{s} \in \mathtt{S}_{k,i}$ which is a prefix of an action $(\mathtt{s}, \mathtt{t})$ of $\mathtt{S}_{k+1,j}$, with $\mathtt{t} \in \mathtt{S}_{\mathtt{AM}_{k+1}}$ an action of $\mathtt{AM}_{k+1}$. The following Lem. 3 proves that $\mathcal{T}_{\mathcal{A}}$ is indeed a tree.

Lemma 3 *$\mathcal{T}_{\mathcal{A}}$ is a tree.*

Proof. First we show that each node in $\mathcal{T}_{\mathcal{A}}$ (except v) has at least one predecessor: To do so, we prove that there is no node at level k that 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 single root v of the tree. For the other nodes, assume that there is a node on some level $k > 1$ that 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 Def. 4.

Finally, we 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 obtain $\mathtt{s}_1 \not\sim_{p_0} \mathtt{s}_2 \wedge \mathtt{s}_1 \not\sim_{p_1} \mathtt{s}_2$. 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 Def. 4, however.

So far, we only considered the protocol tree $\mathcal{T}_\mathcal{A}$, which 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, this must induce a partitioning of the resulting epistemic model, i.e., the leaves in $\mathcal{T}_\mathcal{A}$, in order to correctly compute $f(x, y)$ at both processes. The following Lem. 4 shows that the $\mathtt{CAM}_m$ and the resulting $\mathcal{T}_\mathcal{A}$ of a correct solution must induce rectangles at the leafs of $\mathcal{T}_\mathcal{A}$.

Lemma 4 *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 $\mathtt{AM}_1, \ldots, \mathtt{AM}_m$ (resulting in the composed action model $\mathtt{CAM}_m$) and the corresponding protocol tree $\mathcal{T}_\mathcal{A}$. Then, every rectangle corresponds to at least one partition in the final epistemic model $M' = M \otimes \mathtt{CAM}_m$, 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. Suppose that 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 $k = 0$, 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 $\mathtt{AM}_1$ 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 $\mathtt{AM}_1$ 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 $\mathtt{AM}_1$. By the analogous argument, it follows that also the action for (x_2, y_1) in $\mathtt{AM}_1$ is in the very same partition of $\mathtt{AM}_1 = \mathtt{CAM}_1$.

For the induction step, assume that the execution of $\mathcal{A}$ for (x_1, y_1) resp. (x_2, y_2) reached some node $\mathtt{P}_{k,i}$ 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 $\mathtt{P}_{k+1,j}$ corresponding to a partition in $\mathtt{CAM}_{k+1} = (\mathtt{CAM}_k; \mathtt{AM}_{k+1})$. 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 $\mathtt{CAM}_{k+1}$ on the initial epistemic model: the resulting epistemic models are equivalent. Thus, by the same argument as before (only using $\mathtt{AM}_{k+1}$ instead of $\mathtt{AM}_1$), it follows that all the actions on the inputs have to be in the same partition in $\mathtt{AM}_{k+1}$ and hence in $\mathtt{CAM}_{k+1}$. Consequently, all the inputs lead to the same node on level $k + 1$ as asserted.

5.2 Communication complexity lower bounds

In this section, we will prove a lower bound on the number of bits received by the processes during the worst-case execution of a given algorithm $\mathcal{A}$ for computing a function $f(x, y)$, using the action model representation of Sec. 5.1. In the following, $\mathcal{D}^i = \mathcal{D}^i(\mathcal{A})$ denotes the maximum number of bits received by p_i in any execution of $\mathcal{A}$ for computing f, and $\mathcal{D} = \mathcal{D}(\mathcal{A})$ is the maximum total

number of bits received system-wide. We start with two communication complexity lower bounds for $\mathcal{D}^0$ and $\mathcal{D}^1$ (the proof of Lem. 6 is analogous to the proof of Lem. 5):

Lemma 5 *$\mathcal{D}^0 \geq \log_2(R_X)$, where R_X denotes the maximum number of rectangles in any row of M_f, the matrix defining f.*

Proof. Assume that the input of p_0 is some fixed value $x_i \in X$ such that there are R_X rectangles in the row of M_f corresponding to x_i. Suppose in contradiction that $log_2(R_X) > \mathcal{D}^0$, hence $R_X > 2^{\mathcal{D}^0}$. 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 are at least two rectangles R_0, R_1, which cannot be distinguished by p_0. According to Lem. 4, R_0 and R_1 correspond to some leaves ℓ_0 and ℓ_1 of $\mathcal{T}_\mathcal{A}$, which in turn 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 the proof of Lem. 1, let M be the initial epistemic model and $M' = M \otimes \mathtt{M}$ be the result of applying $\mathcal{A}$'s action model $\mathtt{M}$ to M. From Lem. 4 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 and only y_i is arbitrary, we can infer the existence of the following scenario here: (i) $s_0 \sim_{p_0} s_1$ in M, (ii) $\mathtt{s}_0 \in P_0$, $\mathtt{s}_1 \in P_1$ in action model $\mathtt{M}$ applicable to s_0 respectively s_1 (P_0 and P_1 indistinguishable by p_0), and (iii) $(s_0, \mathtt{s}_0) \not\sim_{p_0} (s_1, \mathtt{s}_1)$ in M'. Since p_0 cannot distinguish between the states in partitions P_0 and P_1, however, we inevitably have $(s_0, \mathtt{s}_0) \sim_{p_0} (s_1, \mathtt{s}_1)$, providing the required contradiction.

Lemma 6 *$\mathcal{D}^1 \geq \log_2(R_Y)$, where R_Y denotes the maximum number of rectangles in any column of M_f, the matrix defining f.*

The following Thm. 1 finally establishes a lower bound on $\mathcal{D}$ for a given algorithm $\mathcal{A}$ for computing f. It relies on a lower bound on the number of partitions $t_{\mathtt{CAM}_m}$ of the composed action model $\mathtt{CAM}_m$ for $\mathcal{A}$, and can hence be viewed as an action model analogon for Cor. 1.

Theorem 1 *The maximum total number of bits $\mathcal{D}$ received by the processes in any execution of any algorithm $\mathcal{A}$ that computes $f(x, y)$ in m rounds has the lower bound $log_2\ t_{\mathtt{CAM}_m} \leq \mathcal{D}$, where $t_{\mathtt{CAM}_m}$ is the number of partitions of the composed action model of $\mathcal{A}$ after m rounds. It satisfies $t_{\mathtt{CAM}_m} \geq t$, where t is the number of monochromatic rectangles of $f(x, y)$.*

Proof. Suppose $t_{\mathtt{CAM}_m} < t$, i.e., there are less leaves in $\mathcal{T}_\mathcal{A}$ than there are monochromatic rectangles of $f(x, y)$. Then, there are two rectangles $\mathcal{R}_1, \mathcal{R}_2$ that lead to the same leaf. However, this contradicts Lem. 4, as every leaf corresponds to a single rectangle.

By Lem. 2, a lower bound for the worst-case cost $\mathcal{D}^a = \mathcal{D}^a(\mathtt{CAM}_m)$ regarding process a is $\log_2 t^a_{\mathtt{CAM}_m}$, where $t^a_{\mathtt{CAM}_m}$ is the number of partitions of $\mathtt{CAM}_m$ regarding a. Additionally, every process has to be able to distinguish all the partitions of $\mathtt{CAM}_m$, else the result of $f(x, y)$ would not be common knowledge. Thus, $t^a_{\mathtt{CAM}_m} = t_{\mathtt{CAM}_m}$ and hence $\log_2 t_{\mathtt{CAM}_m} \leq \mathcal{D}^a$. Since trivially $\mathcal{D}^a \leq \mathcal{D}$, we can conclude that $log_2\ t \leq log_2\ t_{\mathtt{CAM}_m} \leq \mathcal{D}$.

5.3 Application examples

We now demonstrate how to apply our approach by means of two simple examples. We first consider distributed function computation, using the function $f(x, y)$ and the protocol $\mathcal{A}$ given in Fig. 1, where the processes send a single bit in each round alternatingly. Recall that $\mathcal{A}$ is optimal in terms of communication complexity.

The corresponding action models, for the 3 rounds of algorithm $\mathcal{A}$, are given in Fig. 4. An action of the form e.g. (x_0, x_1) encodes that p_0 sends the information that its input value is either x_0 or x_1

to p_1. An expression like $K_i x_0$ in a precondition formula means that, in the appropriate epistemic state, p_i knows that x_0 is the input to p_0. Self-loops in the indistinguishability relation of the action models are denoted by *loops*, in the following.

$\mathtt{AM}_1$:	$\mathtt{S}_{\mathtt{AM}_1} = \{(x_0), (x_1, x_2, x_3)\}$ $\sim_{p_i} = loops$ $\mathtt{pre}(x_0) = K_0 x_0$ $\mathtt{pre}((x_1, x_2, x_3)) = K_0(x_1 \vee x_2 \vee x_3)$	(x_0)	(x_1, x_2, x_3)
$\mathtt{AM}_2$:	$\mathtt{S}_{\mathtt{AM}_2} = \{(y_0, y_1, y_3), y_2, (y_0, y_1), (y_2, y_3)\}$ $\sim_{p_i} = loops$ $\mathtt{pre}((y_0, y_1, y_3)) = K_1(x_0 \wedge (y_0 \vee y_1 \vee y_3))$ $\mathtt{pre}(y_2) = K_1(x_0 \wedge y_2)$ $\mathtt{pre}((y_0, y_1)) = K_1((x_1 \vee x_2 \vee x_3) \wedge (y_0 \vee y_1))$ $\mathtt{pre}((y_2, y_3)) = K_1(x_1 \vee x_2 \vee x_3) \wedge (y_2 \vee y_3))$	(y_0, y_1, y_3) (y_0, y_1)	(y_2) (y_2, y_3)
$\mathtt{AM}_3$:	$\mathtt{S}_{\mathtt{AM}_2} = \{x_3, (x_1, x_2)\}$ $\sim_{p_0} = loops$ $\mathtt{pre}(x_3) = K_0(x_3 \wedge (y_2 \vee y_3))$ $\mathtt{pre}((x_1, x_2)) = K_0((x_1 \vee x_2) \wedge (y_2 \vee y_3))$	(x_3)	(x_1, x_2)

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

$\mathtt{S}_{\mathtt{CAM}_3} = \{\mathtt{s}_0 = (x_0, (y_0, y_1, y_3)), \mathtt{s}_1 = (x_0, y_2),$ $\mathtt{s}_2 = ((x_1, x_2, x_3), (y_0, y_1)),$ $\mathtt{s}_3 = (((x_1, x_2, x_3), (y_2, y_3)), x_3),$ $\mathtt{s}_4 = (((x_1, x_2, x_3), (y_2, y_3)), (x_1, x_2))\}$	$\mathtt{s}_0$	$\mathtt{s}_1$
$\sim_{p_i} = loops$	$\mathtt{s}_2$	$\mathtt{s}_3$
$\mathtt{pre}((x_0, (y_0, y_1, y_2))) = K_0 x_0 \wedge K_1(y_0 \vee y_1 \vee y_3)$		
$\mathtt{pre}((x_0, y_2)) = K_0 x_0 \wedge K_1 y_2$		$\mathtt{s}_4$
$\mathtt{pre}((x_1, x_2, x_3), (y_0, y_1)) = K_0(x_1 \vee x_2 \vee x_3) \wedge K_1(y_0 \vee y_1)$		
$\mathtt{pre}(((x_1, x_2, x_3), (y_2, y_3)), x_3) = K_0 x_3 \wedge K_1(y_2 \vee y_3)$		
$\mathtt{pre}(((x_1, x_2, x_3), (y_2, y_3)), (x_1, x_2)) = K_0(x_1 \vee x_2) \wedge K_1(y_2 \vee y_3)$		

Fig. 4: Precondition functions (left) and action models (right) for the optimal algorithm $\mathcal{A}$ for f, given in Fig. 1.

The corresponding protocol tree $\mathcal{T}_\mathcal{A}$ and the rectangles corresponding to $\mathcal{T}_\mathcal{A}$ are depicted in Fig. 5. It is apparent that there are 5 completely separated partitions in $\mathtt{CAM}_3$ corresponding to 5 leaves in the protocol tree $\mathcal{T}_\mathcal{A}$. Thm. 1 thus reveals that $\mathcal{D} \geq \log_2(5)$. Alternatively, since $\mathcal{A}$ follows the original Yao protocol, we can also directly apply Cor. 1. It confirms that $\mathcal{A}$ has to communicate at least $\log_2(5) \leq 3$ bits to compute f. And indeed, in the corresponding protocol tree (of height 3), there are paths where 1 bit is sent/received in each of the 3 rounds.

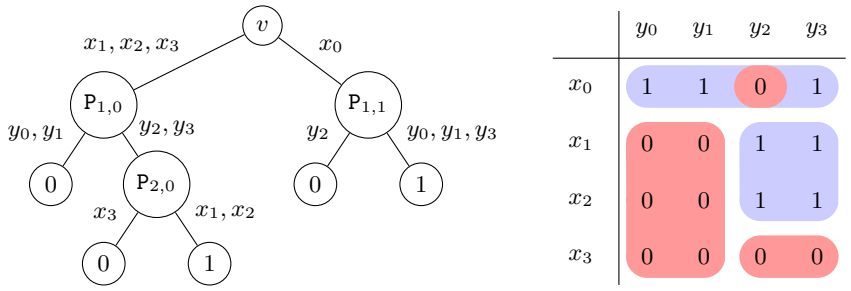


Fig. 5: The protocol tree $\mathcal{T}_\mathcal{A}$ for function f defined in Fig. 1.

As our second application, we will use the approach developed in the previous sections to obtain lower bounds for the communication complexity $\mathcal{D}$ for distributed consensus in directed dynamic networks controlled by a message adversary.

In the *consensus* problem [17], 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 in our model, where processes cannot fail but communication is unreliable, 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 $y_q = v$.

We restrict our attention to the very simple directed dynamic network made up of two synchronous processes $p_0,\ p_1$. The communication graph $\mathcal{G}^r$ of each round r is controlled by an omniscient *message adversary* (MA) here [1,5,22], which determines which messages are delivered resp. lost in round r: it chooses a sequence $\sigma = \mathcal{G}^1, \mathcal{G}^2, \ldots$ of graphs $\mathcal{G}^r \in \{\leftarrow, \leftrightarrow, \rightarrow\}$ for any round r. We will focus on the message adversary $MA_{\leftrightarrow^2}$ here, which may generate all graph sequences not starting with $\mathcal{G}^1 = \mathcal{G}^2 = \leftrightarrow$. There is a simple algorithm $\mathcal{A}$ solving consensus under $MA_{\leftrightarrow^2}$.

Lacking space forced us to relegate the detailed modeling and analysis to Appendix A. In a nutshell, we pursued two different approaches there:
(1) One can consider consensus as the distributed computation of a function $f(x, y, \sigma)$, where the result depends on the inputs of p_0 (x) and p_1 (y) and on the particular graph sequence σ chosen by the MA.
(2) In order to directly apply our approach, we had to address the problem 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. Consequently, the actual result of $f(x, y, \sigma)$ depends on x, y and σ, but also on the choices made by the algorithm $\mathcal{A}$. We solved this problem by partitioning the function $f(x, y, \sigma)$ into multiple functions $f_i(x, y)$, which can be treated independently. Since every $f_i(x, y)$ led to the (trivial) lower bound $\mathcal{D} \geq 1$, we obtained the same (trivial) lower bound $\mathcal{D} \geq 1$ as in (1).

6 Conclusions

We established a relation between the number of partitions in the composed action model of a synchronous distributed algorithm $\mathcal{A}$ and the number of bits received by some process in a worst case execution. For the restricted case of deterministic distributed function computation among 2 processes, we also provided a lower bound for the total communication complexity of any correct solution algorithm. We provided two simple applications of our approach, which confirmed an already known communication complexity lower bound for distributed function computation and even reached out to consensus in directed dynamic networks under a message adversary. Part of our current work is devoted to the shortcomings of our current approach, most notably, the restriction to 2 processes.

References

1. Afek, Y., Gafni, E.: Asynchrony from synchrony. In: Frey, D., Raynal, M., Sarkar, S., Shyamasundar, R., Sinha, P. (eds.) Distributed Computing and Networking, Lecture Notes in Computer Science, vol. 7730, pp. 225–239. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-35668-1_16, `http://dx.doi.org/10.1007/978-3-642-35668-1_16`
2. Alechina, N., Logan, B., Nguyen, H.N., Rakib, A.: Verifying time, memory and communication bounds in systems of reasoning agents. Synthese **169**(2), 385–403 (2009)

3. Ben-Zvi, I., Moses, Y.: Beyond Lamport's happened-before: On the role of time bounds in synchronous systems. In: Lynch, N., Shvartsman, A. (eds.) Distributed Computing, Lecture Notes in Computer Science, vol. 6343, pp. 421–436. Springer Berlin / Heidelberg (2010)
4. Ben-Zvi, I., Moses, Y.: Beyond Lamport's happened-before: On time bounds and the ordering of events in distributed systems. J. ACM **61**(2), 13:1–13:26 (Apr 2014). https://doi.org/10.1145/2542181, http://doi.acm.org/10.1145/2542181
5. Biely, M., Robinson, P., Schmid, U., Schwarz, M., Winkler, K.: Gracefully degrading consensus and k-set agreement in directed dynamic networks. Theoretical Computer Science **726**, 41–77 (2018). https://doi.org/https://doi.org/10.1016/j.tcs.2018.02.019, http://www.sciencedirect.com/science/article/pii/S0304397518301166
6. Cyriac, A., Krishnan, K.M.: Lower bound for the communication complexity of the russian cards problem. CoRR **abs/0805.1974** (2008), http://arxiv.org/abs/0805.1974
7. Dinitz, Y., Moran, S., Rajsbaum, S.: Bit complexity of breaking and achieving symmetry in chains and rings. J. ACM **55**(1), 3:1–3:28 (Feb 2008). https://doi.org/10.1145/1326554.1326557, http://doi.acm.org/10.1145/1326554.1326557
8. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic. Springer Netherlands (2008). https://doi.org/10.1007/978-1-4020-5839-4
9. Dolev, D., Feder, T.: Multiparty communication complexity. In: Foundations of Computer Science, 1989., 30th Annual Symposium on. pp. 428–433. IEEE (1989)
10. Draisma, J., Kushilevitz, E., Weinreb, E.: Partition arguments in multiparty communication complexity. CoRR **abs/0909.5684** (2009), http://arxiv.org/abs/0909.5684
11. Fagin, R., Moses, Y., Halpern, J., Vardi, M.: Reasoning About Knowledge. MIT Press (2003), https://books.google.at/books?id=xHmlRamoszMC
12. Gerbrandy, J.D.: Dynamic epistemic logic. Institute for Logic, Language and Computation (ILLC), University of Amsterdam (1997)
13. Halpern, J.Y., Moses, Y.: Knowledge and common knowledge in a distributed environment. J. ACM **37**(3), 549–587 (1990). https://doi.org/http://doi.acm.org/10.1145/79147.79161
14. Hintikka, J.: Knowledge and belief: an introduction to the logic of the two notions. Contemporary philosophy, Cornell University Press (1962), https://books.google.de/books?id=N28OAAAAIAAJ
15. Holzer, S., Wattenhofer, R.: Optimal distributed all pairs shortest paths and applications. In: Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing. pp. 355–364. PODC '12, ACM, New York, NY, USA (2012). https://doi.org/10.1145/2332432.2332504, http://doi.acm.org/10.1145/2332432.2332504
16. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press (1997), https://books.google.at/books?id=yiV6pwAACAAJ
17. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**(3), 382–401 (July 1982)
18. Lipton, R.J., Sedgewick, R.: Lower bounds for vlsi. In: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing. pp. 300–307. STOC '81, ACM, New York, NY, USA (1981). https://doi.org/10.1145/800076.802482, http://doi.acm.org/10.1145/800076.802482
19. Mehlhorn, K., Schmidt, E.M.: 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. pp. 330–337. STOC '82, ACM, New York, NY, USA (1982). https://doi.org/10.1145/800070.802208, http://doi.acm.org/10.1145/800070.802208
20. Moses, Y., Dolev, D., Halpern, J.Y.: Cheating husbands and other stories: A case study of knowledge, action, and communication. Distributed Computing **1**(3), 167–176 (1986). https://doi.org/10.1007/BF01661170, http://dx.doi.org/10.1007/BF01661170
21. Plaza, J.: Logics of public communications. Synthese **158**(2), 165–179 (Sep 2007). https://doi.org/10.1007/s11229-007-9168-7, https://doi.org/10.1007/s11229-007-9168-7
22. Schwarz, M., Winkler, K., Schmid, U.: Fast consensus under eventually stabilizing message adversaries. In: Proceedings of the 17th International Conference on Distributed Computing and Networking. pp. 7:1–7:10. ICDCN '16, ACM, New York, NY, USA (2016). https://doi.org/10.1145/2833312.2833323, http://doi.acm.org/10.1145/2833312.2833323
23. Yao, A.C.: Some complexity questions related to distributive computing (preliminary report). In: Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA. pp. 209–213 (1979). https://doi.org/10.1145/800135.804414, http://doi.acm.org/10.1145/800135.804414

A Lower bounds for consensus in dynamic networks

In this appendix, we provide the details for applying our approach to consensus under message adversary $MA_{\leftrightarrow^2}$ that had been dropped from the main body of our submission due to lacking space.

There is a simple algorithm $\mathcal{A}$ for solving consensus under $MA_{\leftrightarrow^2}$: In round 1, p_0 and p_1 both send their initial value x resp. y to each other. In round 2, each process sends the last value it received (or its initial value if it did not receive any value in round 1). Once a process received the same value it sent or does not receive a message in round 2, it decides the last value sent and terminates. It is easy to show that it indeed solves consensus under $MA_{\leftrightarrow^2}$. Note that, in a straightforward implementation of $\mathcal{A}$, the processes would receive 3 bits in total (i.e., system-wide) in some execution.

A.1 First approach: Explicitly incorporating σ in $f(x, y, \sigma)$

We can consider consensus as the distributed computation of a function $f_{\mathcal{A}}(x, y, \sigma)$ that depends on the inputs of p_0 (x) and p_1 (y), but also on the particular graph sequence σ and the choices of the solution algorithm $\mathcal{A}$ for resolving the freedom offered by the validity property. So unlike in classic distributed function computation, where $f(x, y)$ is given and every solution algorithm must compute the same function, $f_{\mathcal{A}}(x, y, \sigma)$ inherently depends on $\mathcal{A}$ here.

Fig. 6 shows $f_{\leftrightarrow^2}(x, y, \sigma)$ for $\mathcal{A}$, as well as the action model $\mathtt{AM}_k$ for each round k: The single actions are given in the form $(v_0, v_1, \mathcal{G})$, with $v_i \in \{0, 1\}$ the value sent by p_i and $\mathcal{G} \in \{\leftarrow, \leftrightarrow, \rightarrow\}$ the graph chosen by $MA_{\leftrightarrow^2}$ in the current round. The precondition for an action $(v_0, v_1, \mathcal{G})$ is $K_0 v_0 \wedge K_1 v_1$.

	$y=0$		$y=1$	
$x=$	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

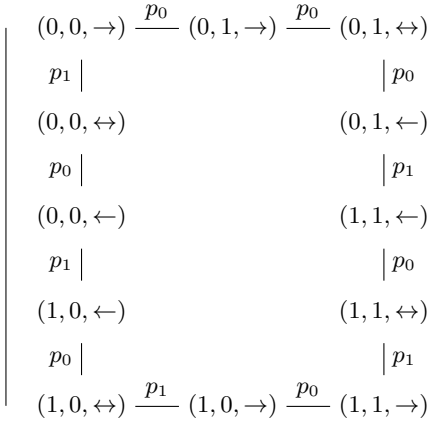


Fig. 6: Left: A function $f_{\leftrightarrow^2}(x, y, \sigma)$ defining the outcome of consensus in the 2 process case with adversary $MA_{\leftrightarrow^2}$. Right: The Action Model $\mathtt{AM}_k$ for each round k of algorithm $\mathcal{A}$. Note that the indistinguishability relation $\sim$ is replaced by straight lines here to improve readability.

The resulting composed action model after two rounds, $\mathtt{CAM}_2 = (\mathtt{AM}_1; \mathtt{AM}_2)$, is depicted in Fig. 7 and has 32 actions split up in 2 partitions. Recall that the composition must explicitly omit combining $(.,.,\leftrightarrow) \in \mathtt{AM}_1$ and $(.,.,\leftrightarrow) \in \mathtt{AM}_2$, as this is the forbidden graph sequence for $MA_{\leftrightarrow^2}$. The protocol tree $\mathcal{T}_{cons}$ corresponding to $\mathcal{A}$ is depicted in Fig. 8.

In $\mathcal{T}_{cons}$ resp. $\mathtt{CAM}_2$, we observe $t_{\mathtt{CAM}_2} = 2$ leaves resp. partitions. Recalling Thm. 1, we observe the lower bound $\mathcal{D}_{\mathcal{A}}(f_{\leftrightarrow^2}) \geq 1$. Obviously, this is a trivial lower bound for the number of received bits in the system for solving consensus.

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	$(1,1,\leftarrow),(1,1,\leftarrow)$

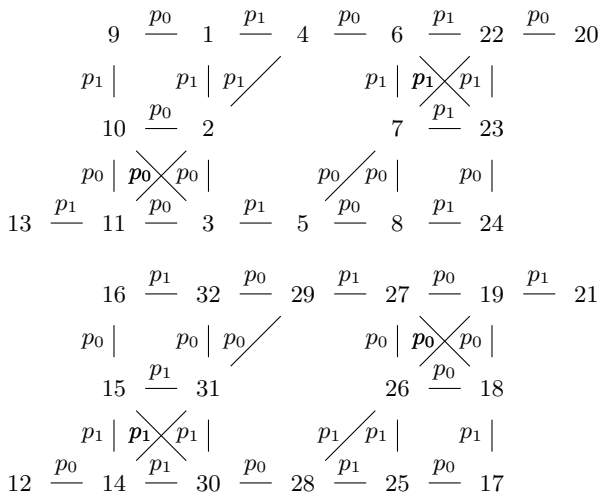


Fig. 7: The graphical representation of $\texttt{CAM}_2$. The actions in the upper partition all lead to the decision 0, the ones in the lower partition to a decision 1. They hence correspond to monochromatic rectangles.

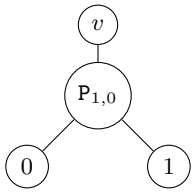


Fig. 8: The protocol tree $\mathcal{T}_{cons}$ corresponding to the action model $\texttt{CAM}_2 = (\texttt{AM}_1;\texttt{AM}_2)$ defined in Fig. 6 that solve consensus by computing the function $f_{\leftrightarrow^2}$. $P_{1,0}$ represents the single partition of $\texttt{AM}_1$ representing Fig. 6, 0 resp. 1 the two partitions of Fig. 7.

A.2 Second approach: Multiple distributed function computations

Alternatively, we can apply our approach also directly, by "splitting up" $f_{\leftrightarrow^2}(x,y,\sigma)$ into several independent distributed function computations: we just 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 by our approach. The worst-case communication complexity of the original algorithm is then determined as 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 Fig. 6 is $\{(\leftrightarrow,\leftarrow),(\rightarrow,\leftarrow),(\rightarrow,\leftrightarrow),(\rightarrow,\rightarrow)\}$. Clearly the partitions P^1_{00} and P^1_{11} are empty due to validity. The decision in P^0_{00} and P^1_{11} is given by validity and independent of the graph sequence, thus there is no need to consider them any further. These eliminations leave only the non-trivial sets P^0_{01}, P^1_{01}, P^0_{10} and P^1_{10}.

We now use these 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 Fig. 6, 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 $P^1_{01}\cap P^1_{10}=P^0_{xy}\cap P^1_{xy}=\emptyset$. 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 determines the actual function $f'(x,y)$ that has to be calculated.

Fig. 9 shows the function $f_{\leftrightarrow^2}(x, y, \sigma)$ partitioned in the two-dimensional matrices $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$. Partitions P^0_{00} and P^1_{11} are marked blue, while the other partitions are marked orange ($f'_{\leftrightarrow^2}$) resp. red ($f''_{\leftrightarrow^2}$).

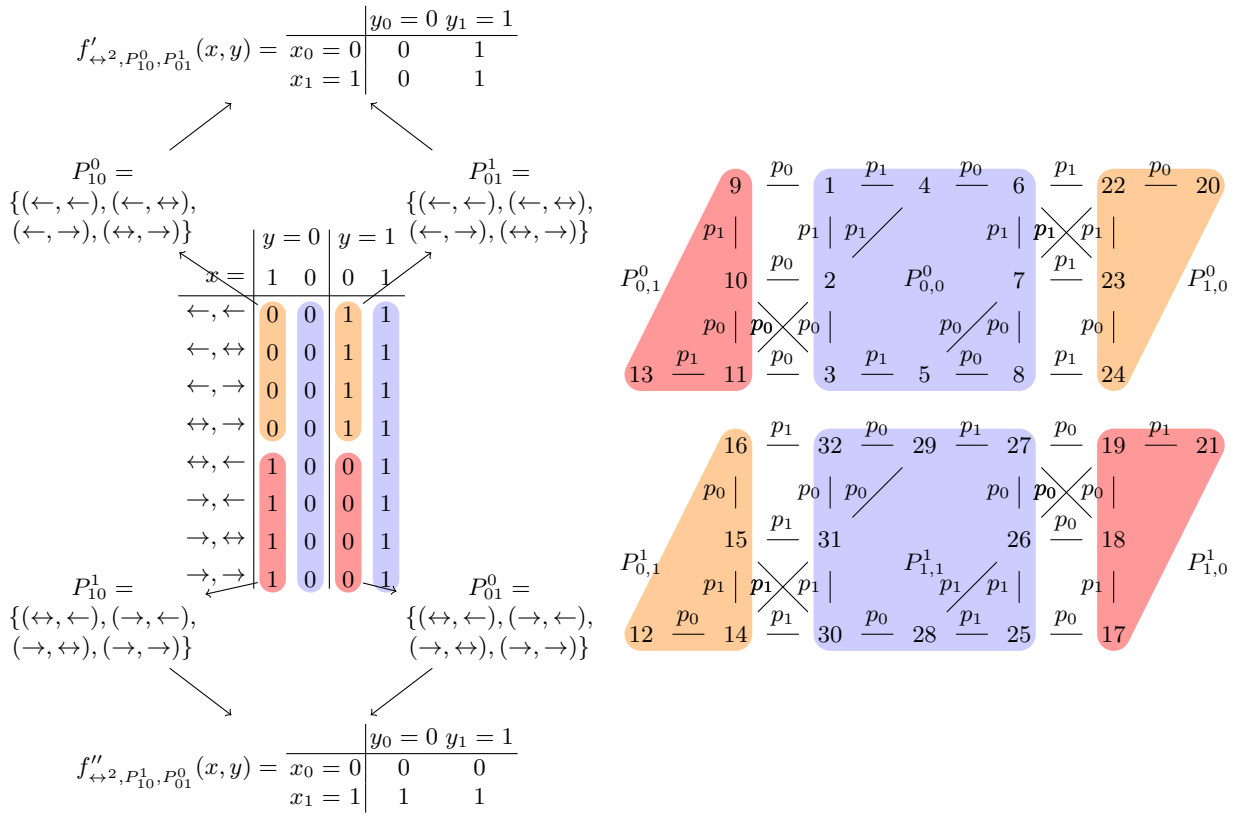


Fig. 9: Partitioning of three-dimensional function $f_{\leftrightarrow^2}$ into two two-dimensional functions $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$. The trivial partitions for P^0_{00} and P^1_{11} are marked blue, while the partitions relevant for the creation of the functions are marked 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 Sec. 5.2 can be used again.

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 and hence all the techniques of Sec. 5.2. More specifically, in order to analyze the communication complexity of $\mathcal{A}$, we analyze the involved functions $f'_{\leftrightarrow^2}$ and $f''_{\leftrightarrow^2}$ independently and take the maximum. Note that, in each of those matrices of our example, 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 to apply our method. Fig. 9 depicts the partitions of $f_{\leftrightarrow^2}$ as well as the partitions of $\texttt{CAM}_2$. We see that in both cases $\texttt{CAM}_2$ partitions to two partitions. Thus, the protocol tree for the two-dimensional functions is the same as in Fig. 8, so $t_{\texttt{CAM}_2} = 2$ again. Thm. 1 again reveals the trivial lower bound 1 already obtained by the first method.