Propositional Gossip Protocols

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Joseph Livesey

April 2023

Abstract

Gossip protocols are programs which can be used by a group of agents to synchronise information which is known by each. We assume each agent holds a unique piece of information which is known as a secret, with the goal of the protocol to reach a situation where all agents are experts, i.e. where each agent knows every secret. Distributed epistemic gossip protocols use epistemic formulas in the component programs for the agents. In this thesis we shall study one of the simplest classes of such protocols: propositional gossip protocols, in which the calls made by agents are determined only by the set of secrets the agent currently knows. Propositional gossip protocols are simple and quick to execute, due to their guards being evaluated in linear time, making them potentially well suited for small devices with limited memory and computational capabilities. This raises many natural questions about conditions necessary for a propositional gossip protocol to be correct, i.e. always terminated in the all-expert state. In this thesis we shall show that such a protocol can be correct only if for any two agents, at least one of them can call the other at some stage in the protocol. In other words, the underlying undirected communication graph must be complete. Furthermore, we shall show that for any such protocol with $n \geq 4$ agents, at least $2n - 2$ calls are required in the worst case. We continue to study the complexity of checking the correctness of such a protocol, as well as the related sub problems of termination and partial correctness, showing that these are coNP-complete problems. We move on to show how this characterization changes when fairness constraints are imposed on the call scheduler used, showing that many more protocols of different structures become viable, as the requirement of the underlying undirected communication graph to be complete is no longer necessary. We continue again to investigate the complexity of checking the correctness of these protocols with fair schedulers, showing the problem of correctness again to be coNP-complete. Finally, we shall look at the topic of simulation, in which we look at if two propositional gossip protocols can have such similarities that one may be able to replicate all call sequences from the other. This could allow for seemingly more complex protocols to be replaced with less computationally demanding protocols, whilst achieving the same results. We find that the problem of checking if a protocol may simulate another protocol is coNP-complete.

Acknowledgements

I would like to thank my supervisors Dr. Dominik Wojtczak and Prof. Sven Schewe for their support and guidance through my studies, as well as all the staff associated with the Department of Computer Science at the University of Liverpool.

I would like to thank my examiners Dr. Patrick Totzke and Prof. Krzysztof R. Apt for taking the time to suggest improvements to this thesis.

Thank you David Abrams and others for reading over my thesis checking for improvements.

Thank you to all my friends particularly Bobby Smith, Calum Forster and Matt Burt for their support and encouragement.

I would like to thank all my family, Carole, Alex and Penny Livesey, Cassie and Pete Seymour, and John Downes for supporting me and providing me with accommodation. I would like to thank Chris and Kath Dean for their support. I would like to thank my grandparents Bernice and Florence for their support. I would also like to thank all my extended family, for their constant encouragement.

I am thankful for being supported by EPSRC NPIF PhD studentship.

Finally I would like to thank my fiancée Emma Dean, I could not have completed this without your constant support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many aspects of technology require devices to communicate, passing on information between them. Examples of this could be a network of self-driving cars, where being able to make observations and relay this information is essential. The network relies on information being known throughout the network, in order for a decision to be made. Knowing the car ahead of you is about to slow down to make a turn, or has detected a pedestrian on a crossing, is critical information for other cars to know and be able to react appropriately. In this thesis we will investigate protocols for such situations when limited to the setting of the gossip problem, in which agents aim to spread information between themselves via calls. We will further restrict this to the propositional setting, where the decision by an agent to make a call is based only on the secrets currently known by this agent, which would potentially allow all information to be spread even in scenarios where computational power and memory is low, for example in a swarm of robots or a sensor network.

Gossip protocols have the aim of spreading information through a network in the form of point-to-point communications, which are referred to as calls. This can be thought of as telephone calls between the agents of the network. Each call is between a pair of agents, one-to-one, and in this call all secrets known by either agent are exchanged, meaning all secrets known by either agent before the call are known by both agents after the call has been made. A protocol can be thought of as a set of rules which determines when certain calls become active. When more than one call can be made at the same time it is typically assumed that a scheduler would decide the order in which the calls are made. Hence, we end up with many possible call sequences, determined by the possible calls given by the rules of the protocol, and the scheduler would choose which call sequence is executed.

Such protocols were successfully used in a number of domains, for instance communication networks [17], computation of aggregate information [22], and data replication [24]. For a more recent account see [20] and [23]. One of the early results established by a number of authors in the seventies, e.g.,[29], is that for n agents, $2n - 4$ calls are necessary and sufficient for all agents to become experts when every agent can communicate with any other agent. A communication graph is the graph formed by creating a node corresponding to each agent, and an edge corresponding to each call, going between the corresponding nodes related to the agents in the call. When such a communication graph is not complete, $2n - 3$ calls may be needed [11] but are sufficient for any connected communication graph [16]. However, all such protocols considered in these papers were centralised.

In [10] a dynamic epistemic logic was introduced in which gossip protocols could be expressed as formulas. These protocols rely on agents' knowledge and are distributed, so they are *distributed epistemic gossip protocols*. This also means they can be seen as special cases of knowledge-based programs introduced in [14].

In [2] a simpler modal logic was introduced that is sufficient to define these protocols and to reason about their correctness. This logic is interesting in its own right and was subsequently studied in further papers such as [3]. In this thesis, we are going to focus on its simplest propositional fragment.

Propositional gossip protocols are a particular type of epistemic gossip protocol in which all guards are propositional. This means that calls being made by each agent are dependent only on the secrets that the agent has had access to. Clearly, this can lead to states where multiple calls are possible at the same time. Here, a scheduler would decide which call takes priority. Throughout this thesis, we assume the scheduler is demonic unless otherwise mentioned, and it picks the order of calls in a way such that the protocol fails, or to maximise the number of calls made before termination. In other words, we study these gossip protocols in their worst-case scenario. In [9], many challenging open problems about general as well as propositional gossip protocols were listed.

In this thesis we begin by investigating the required communication graph, and show that its undirected communication graph must be complete, which resolves Problem 5 from [9] for undirected communication graphs, because for any complete graph a correct propositional gossip protocol is already known (namely, LNS protocol, see Example 2.1.3).

We then solve Problem 7 from [9], which asks to show that no correct propositional gossip protocol exists that always terminates within $2n - 2$ steps for $n \geq 4$ agents. Finally we will look at the complexity of checking the correctness of a given propositional gossip

protocol, showing it is a coNP-complete problem.

1.1 Gossip Spreading

In general, the aim of a communication protocol is to spread information through a network in such a way that all agents in the network learn all secrets in the network.

We can imagine this as a situation where three work colleagues have all being working separately on a group project. They all have an answer to their specific part of the problem, and need to share this between all three members of the team via phone calls. Each time a call is made it is between two people, and in this call all answers known by either person are passed to the other person. We may take the people to be the agents of a network, looking to spread the information known to each other agent in the network. We can consider the protocol to be successful if every agent learns every secret. In our three agent example, this can be achieved in a variety of ways in just three calls. For example, the call sequences *ab.ac.bc* and *ab.bc.ac* will both lead us to a position where all agents know all secrets, as shown in Table 1.1 and Table 1.2. When we say *ab*, this means that agent *a* calls agent *b*, and these agents exchange all known secrets. By inspection, it is clear to see that a total of three calls is both necessary and sufficient for all three agents to become experts. It should be noted however that while three calls are enough, not all call sequences of length three guarantee success. Trivially, if the same call is repeated indefinitely, say, *ab*, then we will never reach a situation where all agents are experts, no matter how many calls are made. Other less trivial examples using three calls can be seen, such as the sequence *ab.ba.cb*, as demonstrated in Table 1.3. It should be noticed, however, that using this same set of calls but in a different order, such as *ab.cb.ba*, would have lead to success, as can be seen in Table 1.4, thus highlighting the effect of the order of calls made.

Table 1.1: Secrets known by agents for call sequence ab.ac.bc

	a	b	c
	A	B	C
ab	AB	AB	C
ac	ABC	AB	ABC
bc	ABC	ABC	ABC

Table 1.2: Secrets known by agents for call sequence ab.bc.ac

	a	b	c
	A	B	C
ab	AB	AB	C
bc	AB	ABC	ABC
ac	ABC	ABC	ABC

Table 1.3: Secrets known by agents for call sequence ab.ba.cb

	a	b	c
	A	B	C
ab	AB	AB	C
ba	AB	AB	C
cb	AB	ABC	ABC

Table 1.4: Secrets known by agents for call sequence ab.cb.ba

	a	b	c
	A	B	C
ab	AB	AB	C
cb	AB	ABC	ABC
ba	ABC	ABC	ABC

When dealing with just three agents we have so few options for calls that the end result can be clear from the outset; let us move on to looking at the case with four agents. We are now in the scenario with four colleagues wanting to share information, not just three.

We may assume that it makes sense to start with a call sequence which worked for three agents, and then expand this. For instance, we may start with the sequence *ab.ac.bc*, and then have each of these agents calling *d*, for example with the sequence *ad.bd.cd*, to give us the call sequence *ab.ac.bc.ad.bd.cd*, which will lead us to a situation where each agent is an expert, as shown in Table 1.5. This achieves the goal in six calls. Naturally, the question

of "is this the fewest calls possible to achieve this state?" arises. The answer in this case is no, we can indeed do better than six calls, in fact, only four calls are necessary. Let us begin with the calls *ab.cd*. This puts us in the situation where all four agents require just two more secrets, which they can learn in the next call. By choosing these options carefully, we are successful in just four calls. For example, the call sequence *ab.cd.ac.bd* is successful in just four calls as can be seen in Table 1.6.

Table 1.5: Secrets known by agents for call sequence ab.ac.bc.ad.bd.cd

	a	b	c	d
	A	B	C	D
ab	AB	AB	C	D
ac	ABC	AB	ABC	D
bc	ABC	ABC	ABC	D
ad	ABCD	ABC	ABC	ABCD
bd	ABCD	ABCD	ABC	ABCD
cd	ABCD	ABCD	ABCD	ABCD

Table 1.6: Secrets known by agents for call sequence ab.cd.ac.bd

	a	b	c	d
	A	B	C	D
ab	AB	AB	C	D
cd	AB	AB	CD	CD
ac	ABCD	AB	ABCD	CD
bd	ABCD	ABCD	ABCD	ABCD

It was also found that four is the fewest number of calls in which all four agents will become experts [29]. This can be confirmed with case analysis. Let us consider the first two calls:

There may be an agent involved in the first call which is also involved in the second call, such as *ab.ac*. It should be pointed out that it does not matter who initiates the call, or the labels, as we would still end up with the same distribution of secrets. Now, in each call the number of agents which know a secret can increase by at most one, as initially

only one agent is familiar with D, (d itself), at least three more calls are required for each agent to learn D and hence become experts.

There may be no agent involved in the first call which is also involved in the second call, such as *ab.cd*. At this point we shall make a trivial, yet important, observation: with each call made, at most two new agents may become experts. This is clearly true, as only agents involved in the call may learn new information, and therefore have a chance of becoming experts due to the call. Hence, as after two calls we have no experts, at least two more calls are required.

Thus, we have found that four calls are needed to make all agents experts when we start with four agents, but, is there a general formula for this? Yes there is. When dealing with n agents, at least $2n-4$ calls are required. This has been shown in [29]. Earlier we showed that when dealing with $n=3$ agents, at least three calls were needed, and hence $2n-3$, however, when dealing with $n \geq 4$ agents, $2n-4$ calls is also sufficient.

How can we show that $2n-4$ calls is also sufficient? We have already shown how to do this for $n=4$, but what happens when we have more agents than this? We will use a very similar method to the $n=4$ case, utilizing the call sequence *ab.cd.ac.bd*.

We shall begin by labeling four of the agents a, b, c, d. For every agent which is not a, b, c or d, we will have this agent initiate a call with a. This would involve $n-4$ calls, and lead to a knowing every secret apart from B, C and D. We now follow this with the call sequence *ab.cd.ac.bd*. n calls have now been made, whilst a, b, c and d are now all experts. To finish, we may have all non-experts call a again, this is another $n-4$ calls, and hence we reach the all expert situation in $2n-4$ calls. This is demonstrated on six agents in Table 1.7, leading to the all expert situation in eight calls with the call sequence $ea.fa.ab.cd.ac.bd.ea.fa$.

We may again use this example to highlight the importance of the order of calls in this sequence. We quickly see that this is not the only call sequence with eight calls leading to the all expert situation, for example we may swap the first and second call, to have the call sequence $fa.ea.ab.cd.ac.bd.ea.fa$. As can be seen in Table 1.8, this does indeed lead to the all expert situation in eight calls. Many other call sequences leading to the all expert situation in eight calls are also possible.

Table 1.7: Secrets known by agents for call sequence ea.fa.ab.cd.ac.bd.ea.fa

	a	b	c	d	e	f
	A	B	C	D	E	F
ea	AE	B	C	D	AE	F
fa	AEF	B	C	D	AE	AEF
ab	ABEF	ABEF	C	D	AE	AEF
cd	ABEF	ABEF	CD	CD	AE	AEF
ac	ABCDEF	ABEF	ABCDEF	CD	AE	AEF
bd	ABCDEF	ABCDEF	ABCDEF	ABCDEF	AE	AEF
ea	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF	AEF
fa	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF

Table 1.8: Secrets known by agents for call sequence fa.ea.ab.cd.ac.bd.ea.fa

	a	b	c	d	e	f
	A	B	C	D	E	F
fa	AF	B	C	D	E	AF
ea	AEF	B	C	D	AEF	AF
ab	ABEF	ABEF	C	D	AEF	AF
cd	ABEF	ABEF	CD	CD	AEF	AF
ac	ABCDEF	ABEF	ABCDEF	CD	AEF	EF
bd	ABCDEF	ABCDEF	ABCDEF	ABCDEF	AEF	AF
ea	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF	AF
fa	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF	ABCDEF

When we design such protocols and can determine the order of calls ourselves, we have shown that $2n - 4$ calls are required, and sufficient, for n agents to become experts when $n \geq 4$. However, let us go back to our example of work colleagues sharing results, assuming we have four members in the team ($n = 4$). We have seen that the call sequence *ab.cd.ac.bd* would lead to all these colleagues knowing all the information in four calls. Yet, imagine this scenario from the perspective of a. a starts off by calling b, and then needs to call c after c has called d, but how would a know if this cd call had already taken place? Simply, they wouldn't; they would make the call to c without knowing whether c had already called d or not. If cd has already been made then we can have all colleagues knowing all the information in four calls, yet if it has not, we would begin with the call sequence *ab.ac*. Now, at least five calls would be required in total, as a, b and c must all still learn D. When looking at the gossip problem through this different perspective of individual agents, the problem changes significantly.

1.2 Epistemic Gossip Spreading

As mentioned, in the real world there may be no way to coordinate the calls in an optimal manner without outside influence. With four colleagues it is possible for them all to become experts in just four calls, but there would need to be outside input to guarantee this by ensuring the calls take place in the correct order. In practice this may not be the case; the colleagues would have to decide whether to make a call or not based on information known to them. From this point on a scheduler would decide the order in which calls take place, considering the available calls at any given time in the protocol. This leads to protocols being defined in such a way that an agent can use only its own knowledge to determine if it wishes to make a call, and to which agent. Further, it gives us the idea of correctness. If we wish to have a protocol which will always lead to termination in a state where all agents are experts, this must be the case no matter which calls are chosen to take place in what order by the scheduler. For example, a well known correct gossip protocol, proposed in [10], is described in Example 1.2.1.

Example 1.2.1. *In [10] a correct gossip protocol, called* Hear My Secret *(HMS in short), was proposed. In the syntax of [2] it has the following program for agent i:*

$$*[[]_{j \in \mathsf{G}} \neg K_i F_j I \to ij].$$

Where F_aS is an atomic formula, which we read as 'agent a is familiar with the secret S' and $K_a\phi$ is a compound formula i.e., 'agent a knows the formula ϕ is true'. Hence, informally, agent i calls agent j if agent i does not know whether j is familiar with his secret. □

1.3 Overview of the Literature

Much work has been done on epistemic gossip protocols. The various types of calls used in [10] and [2] were presented in a uniform framework in [3], where in total 18 types of communication were considered and compared w.r.t. their epistemic strength. In this thesis we study the synchronous communication setting, calls are made and received at the same time, but where agents are not aware of the calls they do not participate in, nor can they synchronize their actions precisely to enable calls to take place at a certain time as there is no global clock. In [5], and its full version [8], it was shown that the semantics and truth of this logic were decidable when the restriction to disallow the nesting of modalities was added. Building upon these results it was proven in [5] that the distributed gossip protocols, the guards of which are defined in this logic, are implementable, that their partial correctness (if upon termination of a protocol, all agents know all secrets) is decidable, and in [7] that termination (if a protocol will terminate) and two forms of fair (when limitations are placed on the repetition of calls) termination of these protocols are decidable, as well. Building on that, [34] showed decidability of the full logic for various variants of the gossiping model. Further, in [4] the computational complexity of this was studied and in [6] an extension with the common knowledge operator was considered and analogous decidability results were established there.

Despite how simple this logic seems to be, there remain natural open problems about it and the gossip protocols defined using it. These problems were discussed at length in [9], where partial results were presented. Open problems listed there regarding axiomatisations for knowledge in gossiping logic were subsequently solved in [34].

Centralised gossip protocols were studied in [18] and [19]. These had the goal to achieve higher-order shared knowledge. This was investigated further in [12], where optimal protocols for various versions of such a generalised gossip problem were given. These protocols depend on various parameters, such as the type of the communication graph or communication. Additionally, different gossip problems which contained some negative goals, for example that certain agents must not know certain secrets, were studied. Such

problems were further studied in [13] with temporal constraints, i.e., a given call has to (or can only) be made within a given time interval. In [30] it was found that arbitrary goals of higher-order logic cannot be obtained when using epistemic gossip protocols, by proving the statement "everyone knows that everyone knows that everyone knows all secrets" is unsatisfiable.

The number of calls needed to reach the desired all expert situation in the distributed but synchronous setting was studied in [31]. In [33, 32] the expected time of termination of several gossip protocols for complete graphs was studied. Dynamic distributed gossip protocols were studied in [35], in which the calls allow the agents to transmit the links as well as share secrets. These protocols were characterised in terms of the class of graphs for which they terminate. Various dynamic gossip protocols were proposed and analysed in [36]. In [15] these protocols were analysed empirically by embedding them in a network programming language NetKAT [1].

1.4 Aim of the Thesis

In this thesis we will be investigating propositional gossip protocols. The aim is to add insight on several problems.
Question 1: What is the required communication graph for a correct propositional gossip protocol?
Question 2: Is there a lower bound on the number of calls required in a correct propositional gossip protocol?
Question 3: What is the complexity of checking if a propositional gossip protocol is correct?
Question 4: How does the introduction of fair schedulers change the results of Questions 1 and 3?
Question 5: What is the complexity of checking if one propositional gossip protocol may simulate another?

We shall begin in Chapter 2 with an introduction to the gossiping logic, giving examples before moving on to tackle these questions. We shall investigate Question 1 in Chapter 3, and Question 2 in Chapter 4, whilst using results from Chapter 3 to help us tackle this question. We look at Question 3 in Chapter 5. In Chapter 6 we begin to look at fairness, and hence looking at Question 4. This continues in Chapter 7. In Chapter 8 we look at Question 5 with regards to simulation of gossip protocols, before finally concluding the thesis in Chapter 9, summarising results and giving ideas for extensions of this work.

The bulk of Chapters 3 and 5 were published in [26]. Most of the work in Chapter 4 was published in [25]. The majority of Chapters 6 and 7 were published in [27].

Chapter 2

Gossiping Logic

2.1 Gossiping Logic

We recall here the framework of [2], which we restrict to the propositional setting. We assume a fixed set G of $n \geq 3$ ***agents*** and stipulate that each agent holds exactly one ***secret***, and that there exists a bijection between the set of agents and the set of secrets. We use it implicitly by denoting the secret of agent a by A, of agent b by B, etc. We denote by Sec the set of all secrets.

The propositional language $\mathcal{L}$ is defined by the following grammar:

$$\phi ::= F_a S \mid \neg\phi \mid \phi \wedge \phi,$$

where $S \in \mathsf{Sec}$ and $a \in \mathsf{G}$. We will distinguish the following sublanguage $\mathcal{L}^a$, where $a \in \mathsf{G}$ is a fixed agent, which disallows all F_b operators for $b \neq a$.

$F_a S$ is an atomic formula, which we read as 'agent a is familiar with the secret S'. Note that in [2], a compound formula $K_a\phi$, i.e., 'agent a knows the formula ϕ is true', was used. Dropping $K_a\phi$ from the logic simplifies greatly its semantics and the execution of a gossip protocol, while still being capable of defining a rich class of protocols. Below we shall freely use other Boolean connectives that can be defined using $\neg$ and $\wedge$ in a standard way. We shall use the following formula

$$Exp_i := \bigwedge_{S \in \mathsf{Sec}} F_i S,$$

that denotes the fact that agent i is an ***expert***, i.e., he is familiar with all the secrets.

Each ***call***, written as ab or sometimes $a \circ b$, concerns two different agents, the ***caller***, a, and the ***callee***, b. After a call, the caller and the callee learn each other's secrets. Calls are denoted by c, d, etc.

In what follows we focus on call sequences. Unless explicitly stated each call sequence is assumed to be finite. The empty sequence is denoted by ϵ. We use $\mathbf{c}$ to denote a call sequence and $\mathbf{\Delta}$ to denote the set of all finite call sequences. Given call sequences $\mathbf{c}$ and $\mathbf{d}$ and a call c we denote by $\mathbf{c}.\mathsf{c}$ the outcome of adding c at the end of the sequence $\mathbf{c}$ and by $\mathbf{c}.\mathbf{d}$ the outcome of appending the sequences $\mathbf{c}$ and $\mathbf{d}$. We say that $\mathbf{c}'$ is an ***extension*** of a call sequence $\mathbf{c}$ if for some call sequence $\mathbf{d}$ we have $\mathbf{c}' = \mathbf{c}.\mathbf{d}$.

To describe what secrets the agents are familiar with, we use the concept of a ***gossip situation***. It is a sequence $\mathsf{s} = (\mathsf{Q}_a)_{a \in \mathsf{G}}$, where $\{A\} \subseteq \mathsf{Q}_a \subseteq \mathsf{Sec}$ for each agent a. Intuitively, Q_a is the set of secrets a is familiar with in the gossip situation s. The ***initial gossip situation*** is the one in which each Q_a equals $\{A\}$ and is denoted by root. It reflects the fact that initially each agent is familiar only with his own secret. Note that an agent a is an expert in a gossip situation s iff $\mathsf{Q}_a = \mathsf{Sec}$.

Each call transforms the current gossip situation by modifying the sets of secrets the agents involved in the call are familiar with as follows. Consider a gossip situation $\mathsf{s} := (\mathsf{Q}_d)_{d \in \mathsf{G}}$ and a call ab. Then

$$ab(\mathsf{s}) := (\mathsf{Q}'_d)_{d \in \mathsf{G}},$$

where $\mathsf{Q}'_a = \mathsf{Q}'_b = \mathsf{Q}_a \cup \mathsf{Q}_b$, and for $c \notin \{a, b\}$, $\mathsf{Q}'_c = \mathsf{Q}_c$.

So the effect of a call is that the caller and the callee share the secrets they are familiar with.

The result of applying a call sequence to a gossip situation s is defined inductively as follows:

$$\epsilon(\mathsf{s}) := \mathsf{s}, \;\; (\mathsf{c}.\mathbf{c})(\mathsf{s}) := \mathbf{c}(\mathsf{c}(\mathsf{s})).$$

Example 2.1.1. We will use the following concise notation for gossip situations. Sets of secrets will be written down as lists. e.g., the set $\{A, B, C\}$ will be written as ABC. Gossip situations will be written down as lists of secrets separated by a comma. e.g., if there are three agents, a, b and c, then $\mathsf{root} = A, B, C$ and the gossip situation $(\{A, B\}, \{A, B\}, \{C\})$ will be written as AB, AB, C.

Let $\mathsf{G} = \{a, b, c\}$. Consider the call sequence $ac.bc.ac$. It generates the following

successive gossip situations starting from root:

$$A, B, C \overset{ac}{\Longrightarrow} AC, B, AC \overset{bc}{\Longrightarrow} AC, ABC, ABC \overset{ac}{\Longrightarrow} ABC, ABC, ABC.$$

Where $\alpha \overset{\mathsf{c}}{\Longrightarrow} \alpha'$ iff $\alpha' = \mathsf{c}(\alpha)$ for gossip situations α and α'.

Definition 2.1.2. *Consider a call sequence $\mathbf{c} \in \mathbf{\Delta}$. We define the satisfaction relation $\models$ inductively as follows:*

$$\begin{aligned} \mathbf{c} \models F_a S \quad &\textit{iff} \quad S \in \mathbf{c}(\mathsf{root})_a, \\ \mathbf{c} \models \neg\phi \quad &\textit{iff} \quad \mathbf{c} \not\models \phi, \\ \mathbf{c} \models \phi_1 \wedge \phi_2 \quad &\textit{iff} \quad \mathbf{c} \models \phi_1 \textit{ and } \mathbf{c} \models \phi_2. \end{aligned}$$

So a formula $F_a S$ is true after the call sequence $\mathbf{c}$ whenever secret S belongs to the set of secrets agent a is familiar with in the situation generated by the call sequence $\mathbf{c}$ applied to the initial situation root. Hence $\mathbf{c} \models Exp_a$ iff agent a is an expert in $\mathbf{c}(\mathsf{root})$.

By a ***propositional component program***, in short a ***program***, for an agent $a \in \mathsf{G}$ we mean a statement of the form

$$*[[]_{j=1}^{m_a} \psi_j^a \to \mathsf{c}_j^a],$$

where each $\psi_j^a \to \mathsf{c}_j^a$ (which we will call a ***rule***), is such that

- agent a is the caller in the call c_j^a,
- $\psi_j^a \in \mathcal{L}^a$ (which we will call a ***guard*** of this rule),

and $m_a \geq 0$ is the number of rules agent a has. If the guard of a rule is true then the corresponding agent, call, and the rule is said to be ***active***.

Intuitively, $*$ denotes a repeated execution of the rules, one at a time, where each time non-deterministically an active rule is selected.

Consider a ***propositional gossip protocol***, P, that is a parallel composition of the propositional component programs $*[[]_{j=1}^{m_a} \psi_j^a \to \mathsf{c}_j^a]$, one for each agent $a \in \mathsf{G}$.

The ***computation tree*** of P is a directed tree defined inductively as follows. Its nodes are call sequences and its root is the empty call sequence ϵ. Further, if $\mathbf{c}$ is a node and

for some rule $\psi_j^a \to \mathsf{c}_j^a$ we have $\mathbf{c} \models \psi_j^a$, then $\mathbf{c}.\mathsf{c}_j^a$ is a node that is a direct descendant of $\mathbf{c}$. Intuitively, the arc from $\mathbf{c}$ to $\mathbf{c}.\mathsf{c}_j^a$ records the effect of the execution of the call c_j^a performed after the call sequence $\mathbf{c}$ took place.

By a ***computation*** of a gossip protocol P we mean a maximal rooted path in its computation tree. In what follows we identify each computation with the unique call sequence it generates. We say that the gossip protocol P is ***partially correct*** if for all leaves $\mathbf{c}$ of the computation tree of P, and all agents a, we have $\mathbf{c} \models Exp_a$, i.e., if each agent is an expert in the gossip situation $\mathbf{c}(\mathsf{root})$. We say furthermore that P ***terminates*** if all its computations are finite and say that P ***is correct*** if it is partially correct and terminates. The agents and possible calls of a given protocol P can be thought of as nodes (agents) and edges (calls) of its ***(directed) communication graph***. In this communication graph, each agent corresponds to exactly one node, with an edge existing between the two nodes corresponding to agents a and b iff ab is a call in P. In the directed communication graph, this would correspond to a directed edge from the node corresponding to a.

In [10] the following correct propositional gossip protocol, called *Learn New Secrets* (LNS in short), for complete directed graphs was proposed.

Example 2.1.3 (LNS protocol)**.** *The following program is used by agent i:*

$$*[[]_{j\in\mathsf{G}} \neg F_i J \to ij].$$

Informally, agent i calls agent j if agent i is not familiar with j's secret.

We now define two new propositional gossip protocols that we will show to be correct. We call the first one the *Learn Next Secret (LXS)* protocol. In this protocol, agents are only able to call agents with a higher "index", which for instance can be their phone number or name, with the corresponding total order ($>$) on G. Just like in the LNS protocol, agents are only able to call another agent if they are not familiar with their secret. Finally, it is required that an agent can make a call to another agent only if he is already familiar with all the secrets of agents with the index value lower than the agent to be called. Formally, this protocol can be defined as follows.

Example 2.1.4 (LXS protocol)**.** *The following program is used by agent i:*

$$*[[]_{\{j\in\mathsf{G}\mid j>i\}} \neg F_i J \wedge \bigwedge_{\{m\in\mathsf{G}\mid m<j\}} F_i M \to ij].$$

Informally, agent i calls agent j if agent i is not familiar with j's secret and j's index is higher according to some total order $>$, whilst being familiar with the secrets of all agents with lower index.

Note that although the communication graph of LXS protocol is not complete, its undirected communication graph is. We now prove that this protocol is correct.

Theorem 2.1.5. *The Learn Next Secret (LXS) protocol is a correct propositional gossip protocol.*

Proof. Termination can be seen as each call made ensures the caller learns at least one secret, hence the protocol terminates, as we have a finite number of secrets to be learned, and a finite number of agents learning them.

It remains to be shown that when the protocol terminates, all agents are experts.

Assume the protocol is not correct. As we know the protocol terminates, this must mean that when we terminate at least one agent is not an expert. Let a be the agent with the lowest index which is not an expert. Now, consider the agent with the lowest index whose secret agent a is not familiar with.

If that agent b has higher index than a, then by definition of the LXS protocol, ab would be available; a contradiction. So the index of b has to be lower than the index of a. Therefore, by definition of the LXS protocol, a could not make any calls before termination.

Now, all agents with lower index than a, including b, must be experts, and hence are familiar with secret A. This means that a was called by an agent say c. But for ca to be available, c must be familiar with all secrets corresponding to agents of index lower than a, including B. Hence when a was first called by c, a learned B; a contradiction. □

We now define the second new protocol, *Learn New Secrets with a special one (LNSwS1)*, where there is a special agent whose secret acts as a guard that all the other agents wait for before making their calls. Formally, we define it as follows where the special agent and its secret is denoted by x and X, respectively.

Example 2.1.6 (LNSwS1 protocol)**.** *The following program is used by the special agent x:*

$$*[[]_{\{j\in\mathsf{G}\}}\neg F_x J \to xj].$$

The following program is used by agents $i \in \mathsf{G} \setminus \{x\}$:

$$*[[]_{\{j\in\mathsf{G}\setminus\{x\}\}}F_i X \wedge \neg F_i J \to ij].$$

Informally, agent x calls agent j if agent x is not familiar with j's secret. For every other agent i, i calls agent j if agent i is not familiar with j's secret, but is familiar with X.

Again, the communication graph of LNSwS1 protocol is not complete, but its undirected communication graph is. We also now show this protocol's correctness.

Theorem 2.1.7. *The LNSwS1 protocol is a correct propositional gossip protocol.*

Proof. Termination can be seen as each call made ensures the caller learns at least one secret, hence the protocol terminates, as we have a finite number of secrets to be learned, and a finite number of agents learning them.

It remains to be shown that when the protocol terminates, all agents are experts.

Assume the protocol is not correct. As we know the protocol terminates, this must mean that when we terminate at least one agent, say y, is not an expert. As x is always happy to initiate a call with any agent it is not familiar with the secret of, as does any agent which is familiar with X, any agent which is not an expert when the protocol terminates must not be familiar with X.

However, by definition of the LNSwS1 protocol, every agent which initiates a call must be familiar with X, and hence every agent which has been involved in any call must be familiar with X. Yet, x is an expert, and so must be familiar with Y. This means that y must have been involved in a call, and hence must be familiar with X; a contradiction. □

Later in this thesis we also consider two variants of fair termination, but we first have to recall the notion of fairness. An infinite computation is ***rule-fair*** (resp. ***agent-fair***) if all rules (resp. agents) that are active after infinitely many prefixes of this computation are selected infinitely often. By definition, a finite computation is rule-fair and agent-fair. We say that a gossip protocol P ***rule-fairly terminates*** (resp. ***agent-fairly terminates***) if all its rule-fair (resp. agent-fair) computations are finite. By definition, every finite computation is fair. We do not define agent-fair nor rule-fair partial correctness, because they are equivalent to partial correctness. A protocol which rule-fairly (resp. agent-fairly) terminates and is partially correct is said to be ***correct under rule-fairness*** (resp. ***correct under agent-fairness***).

In Chapter 3, we are going to study for which classes of communication graphs propositional gossip protocol which are correct under agent-fairness or rule-fairness exist.

Chapter 3

The Required Communication Graph of a Correct Propositional Gossip Protocol

3.1 Introduction

A natural starting point while investigating propositional gossip protocols is to ask for which network structures would they be applicable. The question we aim to investigate is - what is the required communication graph for a correct propositional gossip protocol? Making progress with this problem will give insight to which settings propositional gossip protocols may be applied to give a terminating (all expert) situation. In doing this we will also discover other interesting and useful properties, which will be essential to solving other problems relating to propositional gossip protocols. In Chapter 2, we had three examples of correct propositional gossip protocols. From a quick inspection we notice that the undirected communication graph for each of these protocols is complete. We also see that this undirected communication graph being complete is not sufficient for a propositional gossip protocol to be correct. An example of this is given in Example 3.1.1. In this protocol, every call is available to all agents at all times and hence the undirected communication graph (and in fact the directed communication graph) is complete, however, it is clear that this protocol will never terminate and therefore is not correct.

Example 3.1.1 (Call All Agents protocol)**.** *The following program is used by agent i:*

$$*[[]_{\{j\in \mathsf{G}\}} F_i I \to ij].$$

Informally, any agent may call any other agent at any time.

Ultimately, in this chapter we will show the result that there is no correct propositional gossip protocol whose undirected communication graph is not complete.

3.2 Properties of Correct Propositional Gossip Protocols

We begin by introducing some useful results, regarding propositional gossip protocols in general, as well as correct propositional gossip protocol.

Lemma 3.2.1 (Call Removal)**.** *Consider a propositional gossip protocol P. Let $\mathbf{c}.\mathbf{d}$ be a prefix of a computation of P such that $\mathbf{c}.\mathbf{d} \not\models F_a B$. Let $\mathbf{d}'$ be $\mathbf{d}$ where all calls that involve an agent familiar with B are removed, then $\mathbf{c}.\mathbf{d}'$ is also a prefix of a computation of P and, moreover, $(\mathbf{c}.\mathbf{d})(\mathsf{root})_a = (\mathbf{c}.\mathbf{d}')(\mathsf{root})_a$.*

Proof. It suffices to show that we can remove the last such call in $\mathbf{d}$, because that clearly preserves the $\mathbf{c}.\mathbf{d}' \not\models F_a B$ property and then we can repeat this procedure until no such calls are left in $\mathbf{d}$.

Let $\mathbf{d} = \mathbf{d}_1.cd.\mathsf{c}_1.\mathsf{c}_2 \dots \mathsf{c}_k$, where cd is the last call that involves an agent that is familiar with B, i.e., $\mathbf{c}.\mathbf{d}_1 \models F_c B \vee F_d B$. Straight from the definition of the outcome of the cd call, for all agents $x \notin \{c, d\}$, $\mathbf{c}.\mathbf{d}_1(\mathsf{root})_x = (\mathbf{c}.\mathbf{d}_1.cd)(\mathsf{root})_x$. At the same time, agents c, d cannot be involved in any of the calls $\mathsf{c}_1, \dots, \mathsf{c}_k$. Therefore, we also have for all agents $x \notin \{c, d\}$, $(\mathbf{c}.\mathbf{d}_1.\mathsf{c}_1)(\mathsf{root})_x = (\mathbf{c}.\mathbf{d}_1.cd.\mathsf{c}_1)(\mathsf{root})_x$, and by induction $(\mathbf{c}.\mathbf{d}_1.\mathsf{c}_1 \dots \mathsf{c}_i)(\mathsf{root})_x = (\mathbf{c}.\mathbf{d}_1.cd.\mathsf{c}_1 \dots \mathsf{c}_i)(\mathsf{root})_x$ for all $i \leq k$. Note that $a \notin \{c, d\}$, because $\mathbf{c}.\mathbf{d} \not\models F_a B$, so $(\mathbf{c}.\mathbf{d})(\mathsf{root})_a = (\mathbf{c}.\mathbf{d}')(\mathsf{root})_a$ holds as desired.

Now consider the guard ϕ_i associated with the call c_i where $i \leq k$. By assumption on P, ϕ_i is a propositional formula built out of the atomic formulas of the form $F_x S$ where $x \notin \{c, d\}$ is the agent making the call c_i. We already showed that $(\mathbf{c}.\mathbf{d}_1.\mathsf{c}_1 \dots \mathsf{c}_{i-1})(\mathsf{root})_x = (\mathbf{c}.\mathbf{d}_1.cd.\mathsf{c}_1 \dots \mathsf{c}_{i-1})(\mathsf{root})_x$, so the truth of these atomic formulas is not affected by the removal of the call cd from $\mathbf{d}$. This shows that we have $\mathbf{c}.\mathbf{d}_1.cd.\mathsf{c}_1 \dots \mathsf{c}_{i-1} \models \phi_i$ iff $\mathbf{c}.\mathbf{d}_1.\mathsf{c}_1 \dots \mathsf{c}_{i-1} \models \phi_i$ and so c_i can also be made by the protocol P after $\mathbf{c}.\mathbf{d}_1.\mathsf{c}_1 \dots \mathsf{c}_{i-1}$. □

Example 3.2.2 (An example of Lemma 3.2.1 applied to LNS on 4 agents)**.** *Consider LNS on 4 agents a,b,c,d, we shall refer to this protocol in this example as P. We can take the prefix of a computation of P ac.cd.cb.ad and apply Lemma 3.2.1 to get the call sequence ac.cd.ad, which indeed is a valid prefix of a computation of P.*

Lemma 3.2.1 states that if in a prefix of a computation of P, an agent a is not familiar with B, then there exists a prefix of a computation of P such that all calls involving agents familiar with B have been removed, yet a is familiar with the same secrets in both instances. It is extremely useful in helping us adapt possible call sequences for a given protocol in order to generate other possible call sequences. We establish now what the correctness of a propositional protocol implies for the order of calls.

Lemma 3.2.3 (Initiation)**.** *Consider any call sequence $\mathbf{c}$ which is a prefix of a computation of a correct propositional gossip protocol P such that $\mathbf{c} \models F_a B$. There does not exist a call sequence $\mathbf{d}$ such that $\mathbf{c}.\mathbf{d}.ab$ is a prefix of a computation of P. (In other words, agent a will never call agent b if agent a is already familiar with B.)*

Proof. Suppose such a call sequence $\mathbf{d}$ exists. If $\mathbf{c}.\mathbf{d}(\mathsf{root})_b \subseteq \mathbf{c}.\mathbf{d}(\mathsf{root})_a$, then we have $\mathbf{c}.\mathbf{d}(\mathsf{root})_a = \mathbf{c}.\mathbf{d}.ab(\mathsf{root})_a$ and so the guard ϕ of the call ab is still true after ab is made. As a result, ab could be repeated indefinitely after $\mathbf{c}.\mathbf{d}$; a contradiction with the assumption that P is terminating.

Otherwise, there exists a secret, X, such that $\mathbf{c}.\mathbf{d} \models F_b X \wedge \neg F_a X$ before the call ab takes place. Let us now remove all calls from $\mathbf{c}.\mathbf{d}$ that involve agents that are familiar with the secret X, which results in a call sequence $\mathbf{c}'.\mathbf{d}'$. Lemma 3.2.1 then implies that $\mathbf{c}'.\mathbf{d}'$ is also a prefix of a computation of P and $\mathbf{c}.\mathbf{d}(\mathsf{root})_a = \mathbf{c}'.\mathbf{d}'(\mathsf{root})_a$, so the call ab can still be made after $\mathbf{c}'.\mathbf{d}'$. At the same time, $\mathbf{c}'.\mathbf{d}'(\mathsf{root})_b \subsetneq \mathbf{c}.\mathbf{d}(\mathsf{root})_b$, because agent b is no longer familiar with secret X and possibly other secrets as well.

If there is still a secret Y left such that $\mathbf{c}'.\mathbf{d}' \models F_b Y \wedge \neg F_a Y$ then we again remove all calls from $\mathbf{c}'.\mathbf{d}'$ that involve agents that are familiar with the secret Y, which results in a call sequence $\mathbf{c}''.\mathbf{d}''$. We keep repeating this process until we reach a call sequence $\mathbf{c}^\Omega.\mathbf{d}^\Omega$ such that $\mathbf{c}^\Omega.\mathbf{d}^\Omega(\mathsf{root})_b \subseteq \mathbf{c}^\Omega.\mathbf{d}^\Omega(\mathsf{root})_a$ and $\mathbf{c}^\Omega.\mathbf{d}^\Omega.ab$ is prefix of a computation of P, because $\mathbf{c}.\mathbf{d}(\mathsf{root})_a = \mathbf{c}^\Omega.\mathbf{d}^\Omega(\mathsf{root})_a$. Just like before, we arrive to a contradiction, because the call ab can now be repeated indefinitely. □

Already these two lemmas allow us to show non-existence of a correct propositional gossip protocol for a wide range of natural communication graph classes. The first undi-

rected graph class that we consider is the star graph, i.e., when communication is only possible via a single central agent. This was already shown in [9], however our proof is much simpler.

Theorem 3.2.4. *There is no correct propositional protocol whose undirected communication graph is a star graph (a tree in which all nodes apart from one, have vertex degree one) with at least 3 agents, where a tree is a connected graph with no cycles.*

Proof. Suppose such a protocol P exists. From Lemma 3.2.3 each agent, apart from the central agent, is involved in at most one call, as otherwise the protocol will not be correct. Therefore, the non-central agent involved in the first call will not have any further calls, and so will never become an expert. □

We can also show the results that should the undirected communication graph be a cycle graph with at least 5 agents then no correct propositional gossip protocol exists.

Theorem 3.2.5. *Suppose that the undirected communication graph is a cycle graph with at least 5 agents. Then no correct propositional protocol exists.*

Proof. Suppose such a correct propositional protocol P exists. From Lemma 3.2.3 each agent is involved in at most two calls: one with each of the agent's neighbours. Hence, after an agent has had 2 calls it must be an expert, because otherwise P would not be correct. Let agent a and agent b be involved in the first call ab, which have opposite side neighbours z and c respectively.

Any computation starting with $ab.az$ leads to failure of P as then a has no further possible calls and is not an expert. Note that if $ab.\mathbf{c}.az$, where call sequence $\mathbf{c}$ does not involve agent a, is a prefix of some computation of P then so is $ab.az$. This is because $ab.az(\mathsf{root})_a = ab.\mathbf{c}.az(\mathsf{root})_a$ (i.e., agent a is familiar with the same secrets after ab as after $ab.\mathbf{c}$), and so the call az may be made immediately after ab. Note that Lemma 3.2.3 excludes the possibility that ba follows ab, so $ab.\mathbf{c}.za$, for some $\mathbf{c}$, must take place instead and at that point agent z is not familiar with A.

Note that $ab.\mathbf{c}.za \models Exp_a$, because za is the last call a can be involved in. But then also $ab.\mathbf{c}.za \models Exp_z$. Hence, z must have been in a call with its opposite neighbour y to learn all other secrets to pass onto agent a, which in turn had to be in a call with its opposite neighbour x to learn X. But now, after a call yz is made, y has been involved in all possible calls but is not an expert, since after yz, agent z (and so also y) cannot be familiar with A as otherwise he would not be able to call agent a later. □

We now proceed to show that there is no correct propositional protocol whose undirected communication graph is not complete. Note that if there are only two agents then this statement is trivial. In the case of three agents, an undirected connected graph with a missing link is a star graph so the statement follows from Theorem 3.2.4. The proof of this statement in the general case is quite complex, so we break it down into several lemmas. In all these lemmas, we make the assumption that a correct protocol P exists where there is no edge in either direction between two agents denoted by a and b. Theorem 3.2.9 will later show how this assumption leads to a contradiction.

Lemma 3.2.6. *Consider a correct propositional protocol P. Suppose that the connected undirected communication graph is not complete, with no edge between two agents, denoted a and b. There exists a computation of P such that agent b learns A by receiving a call from another agent.*

Proof. We prove this by contradiction and so assume instead that in all computations agent b learns A by calling another agent.

Let us pick a computation where b is familiar with the greatest number of secrets before learning A. In other words, if $\mathbf{c}.bc$ is a prefix of a computation of P where b learns A during the call bc, we require the cardinality of $\mathbf{c}(\mathsf{root})_b$ to be as large as possible when compared to other such prefixes of computations of P. This is well-defined as this value is an integer between 1 and $|\mathsf{G}|$ and agent b has to learn A in every computation as P is correct.

We know that $\mathbf{c} \models \neg F_b C$, because otherwise bc would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with C in $\mathbf{c}$ and obtain $\mathbf{c}'$. Due to Lemma 3.2.1, $\mathbf{c}'.bc$ is still a prefix of a computation of P. Note that c is not familiar with A (nor any other secret apart from his own for that matter) after $\mathbf{c}'$, because all his calls were removed. At the same time we know that $\mathbf{c}(\mathsf{root})_b = \mathbf{c}'(\mathsf{root})_b$. If P is indeed correct, then it has to be possible to extend $\mathbf{c}'.bc$ to a prefix $\mathbf{c}'.bc.\mathbf{d}.bd$ of a computation of P, for some $\mathbf{d}$ and d, such that b finally learns A during bd. (Note that due to our original assumption, it cannot be db.) But then $\mathbf{c}(\mathsf{root})_b$ is smaller than $\mathbf{c}'.bc.\mathbf{d}(\mathsf{root})_b$, because the latter includes at least one more secret (namely C); this is a contradiction with the pick of the prefix $\mathbf{c}.bc$ as the prefix where b is familiar with the greatest number of secrets before learning A. □

Lemma 3.2.7. *Consider a correct propositional protocol P. Suppose that the connected undirected communication graph is not complete, with no edge between two agents, denoted*

a and b. For any call sequence $\mathbf{c}$ *without a-calls (any calls involving agent a) and any agent* $c \in \mathsf{G} \setminus \{a, b\}$*, if* $\mathbf{c}.ca$ *is a prefix of a computation of* P *such that* $\mathbf{c} \models F_c B$ *(i.e., a learns B from c), then* $\mathbf{c}.\mathbf{d}.bc$ *is also a prefix of a computation of* P *for some call sequence* $\mathbf{d}$*.*

Proof. Let us pick any prefix of a computation of P $\mathbf{c}.ca$ where $\mathbf{c}$ is without a-calls (any calls involving agent a), such that $\mathbf{c} \models F_c B$. Note that after $\mathbf{c}.ca$, all agents that are familiar with A (agents a and c, only) are all familiar with B. As in every call all secrets are exchanged, any extension of $\mathbf{c}.ca$ would also have this property.

Due to Lemma 3.2.3, no agent that is familiar with A would call b after $\mathbf{c}.ca$, because he is already familiar with B. Moreover, b will never call a (missing link). Let us assume he does not call c either. Then, as b must learn A eventually, he must call a different agent. From here the proof follows similarly as in Lemma 3.2.6, but with an initial call sequence $\mathbf{c}.ca$.

Let us pick a computation that starts with $\mathbf{c}.ca$ where b is familiar with the greatest number of secrets before learning A. In other words, if $\mathbf{c}.ca.\mathbf{d}.bd$ is a prefix of a computation of P where b learns A during the call bd for some $d \in \mathsf{G} \setminus \{a, b, c\}$, we require the cardinality of $(\mathbf{c}.ca.\mathbf{d})(\mathsf{root})_b$ to be as large as possible when compared to other such prefixes of computations of P. This is well-defined as this value is an integer between 1 and $|\mathsf{G}|$.

We know that $\mathbf{c}.ca.\mathbf{d} \models \neg F_b D$, because otherwise bd would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with D in $\mathbf{d}$ and obtain $\mathbf{d}'$. Due to Lemma 3.2.1, $\mathbf{c}.ca.\mathbf{d}(\mathsf{root})_b = \mathbf{c}.ca.\mathbf{d}'(\mathsf{root})_b$, so $\mathbf{c}.ca.\mathbf{d}'.bd$ is also a prefix of a computation of P.

Note that d is not familiar with A after $\mathbf{c}.ca.\mathbf{d}'$, because $\mathbf{c}$ and $\mathbf{d}'$ have no calls involving agents familiar with A. Hence $\mathbf{c}.ca.\mathbf{d}'.bd \models \neg F_b A$. So if P is indeed correct, then it has to be possible to extend $\mathbf{c}.ca.\mathbf{d}'$ to a prefix $\mathbf{c}.ca.\mathbf{d}'.bd.\mathbf{e}.be$ of a computation of P, for some call sequence $\mathbf{e}$ and agent e, such that b finally learns A during be. (Note that it cannot be eb, because no agent familiar with A would call b after $\mathbf{c}.ca$.) But then $\mathbf{c}.ca.\mathbf{d}(\mathsf{root})_b$ is smaller than $\mathbf{c}.ca.\mathbf{d}'.bd.\mathbf{e}(\mathsf{root})_b$, because the latter includes at least one more secret (namely D); this is a contradiction with the pick of the prefix $\mathbf{c}.ca.\mathbf{d}.bd$ as the prefix where b is familiar with the greatest number of secrets before learning A. In conclusion, it must be possible for b to call c after $\mathbf{c}.ca$.

We have shown so far that $\mathbf{c}.ca.\mathbf{d}.bc$ has to be a prefix of a computation of P for some call sequence $\mathbf{d}$. Note that $\mathbf{c}.ca.\mathbf{d} \not\models F_b C$ due to Lemma 3.2.3. We can then remove all

calls of agents familiar with C from the suffix $ca.\mathbf{d}$ of $\mathbf{c}.ca.\mathbf{d}$, to obtain $\mathbf{c}.\mathbf{d}'$. Then due to Lemma 3.2.1, $\mathbf{c}.\mathbf{d}'.bc$ is also a prefix of a computation of P. □

Using very similar techniques we can show the following.

Lemma 3.2.8. *Consider a correct propositional protocol P. Suppose that the connected undirected communication graph is not complete, with no edge between two agents, denoted a and b. For any call sequence $\mathbf{c}$ without a-calls and any agent $c \in \mathsf{G} \setminus \{a, b\}$, if $\mathbf{c}.ca$ is a prefix of a computation of P such that $\mathbf{c} \models F_c B$, and d is the last agent in a call with c before call ca takes place, then $\mathbf{c}.ca.\mathbf{d}.da$ is also a prefix of a computation of P for some $\mathbf{d}$.*

Proof. Let us pick any prefix of a computation of P without a-calls, $\mathbf{c}.ca$, such that $\mathbf{c} \models F_c B$ and the last call of c before ca is either cd or dc. Note that after $\mathbf{c}.ca$, all agents that are familiar with A (agents a and c, only) are also familiar with D. As in every call all secrets are exchanged, any extension of $\mathbf{c}.ca$ would also have this property.

Due to Lemma 3.2.3, no agent that is familiar with A in any extension of $\mathbf{c}.ca$ will call d, because he would already be familiar with D. Moreover, d will never call c, due to Lemma 3.2.3, and let us assume he does not call a either. Then, as d must learn A eventually, he must call a different agent (if there are no such agents then we immediately get a contradiction and d has to call a to learn A). From here the proof follows similarly as in Lemma 3.2.7.

Let us pick a computation that starts with $\mathbf{c}.ca$ where d is familiar with the greatest number of secrets before learning A. In other words, if $\mathbf{c}.ca.\mathbf{d}.de$ is a prefix of a computation of P where d learns A during the call de for some $e \in \mathsf{A} \setminus \{a, b, c, d\}$, we require the size of $(\mathbf{c}.ca.\mathbf{d})(\mathsf{root})_d$ to be the largest possible. This is well-defined as this value is an integer between 1 and $|\mathsf{A}|$.

We know that $\mathbf{c}.ca.\mathbf{d} \models \neg F_d E$, because otherwise de would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with E in $\mathbf{d}$ and obtain $\mathbf{d}'$. Due to Lemma 3.2.1, $\mathbf{c}.ca.\mathbf{d}(\mathsf{root})_d = \mathbf{c}.ca.\mathbf{d}'(\mathsf{root})_d$, so $\mathbf{c}.ca.\mathbf{d}'.de$ is also a prefix of a computation of P.

Note that e is not familiar with A after $\mathbf{c}.ca.\mathbf{d}'$, because $\mathbf{c}$ and $\mathbf{d}'$ have no a-calls. Hence $\mathbf{c}.ca.\mathbf{d}'.de \models \neg F_d A$. So if P is indeed correct, then it has to be possible to extend $\mathbf{c}.ca.\mathbf{d}'.de$ to a prefix $\mathbf{c}.ca.\mathbf{d}'.de.\mathbf{e}.df$ of a computation of P, for some call sequence $\mathbf{e}$ and agent f, such that d finally learns A during df. (Note that it cannot be fd, because no agent familiar with A would call d in any extension of $\mathbf{c}.ca$.) But then $\mathbf{c}.ca.\mathbf{d}(\mathsf{root})_d$ is smaller

than $\mathbf{c}.ca.\mathbf{d}'.de.\mathbf{e}(\mathsf{root})_d$, because the latter includes at least one more secret (namely E); this is a contradiction with the pick of the prefix $\mathbf{c}.ca.\mathbf{d}.de$ as the prefix where d is familiar with the greatest number of secrets before learning A. In conclusion, it must be possible for d to call a after $\mathbf{c}.ca$. □

We now have all the ingredients needed to prove the main result of this chapter.

Theorem 3.2.9. *The undirected communication graph for every correct propositional gossip protocol is complete.*

Proof. Suppose such a correct propositional protocol P exists and there are two agents, say a and b, which cannot call each other.

From Lemma 3.2.6 there exists an agent $c \in \mathsf{G} \setminus \{a, b\}$ and a prefix $\mathbf{c}.ca$ of a computation of P such that $\mathbf{c} \models F_c B$. We know that $\mathbf{c} \not\models F_c A$, because otherwise ca would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with A in $\mathbf{c}$ and obtain $\mathbf{c}'$. Due to Lemma 3.2.1, $\mathbf{c}(\mathsf{root})_c = \mathbf{c}'(\mathsf{root})_c$, so $\mathbf{c}'.ca$ is also a prefix of a computation of P, with a not yet having been in a call until ca takes place.

Since $\mathbf{c}'.ca$ is a prefix of a computation of P where $\mathbf{c}'$ has no a-calls and $\mathbf{c}' \models F_c B$ then from Lemma 3.2.7 we get that $\mathbf{c}'.\mathbf{d}.bc$ is also a prefix of a computation of P for some $\mathbf{d}$. Therefore, $\mathbf{c}'$ cannot have bc nor cb call due to Lemma 3.2.3.

Note that there has to be at least one c-call in $\mathbf{c}'$, because $\mathbf{c}' \models F_c B$. We already excluded bc and cb. It cannot be ac nor ca either as ca takes place after $\mathbf{c}'$. Therefore, the last agent to be in a c-call in $\mathbf{c}'$ is some $d \in \mathsf{G} \setminus \{a, b, c\}$. From Lemma 3.2.8 we get that $\mathbf{c}'.ca.\mathbf{e}.da$ must also be a prefix of a computation of P for some $\mathbf{e}$. Note that also $\mathbf{c}' \models F_d B$, because when the call between d and c takes place, c already has to be familiar with B.

We know that $\mathbf{c}'.ca.\mathbf{e} \models \neg F_d A$, because otherwise da would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with A in $\mathbf{e}$ and obtain $\mathbf{e}'$. Due to Lemma 3.2.1, $\mathbf{c}'.ca.\mathbf{e}(\mathsf{root})_d = \mathbf{c}'.ca.\mathbf{e}'(\mathsf{root})_d$, so $\mathbf{c}'.ca.\mathbf{e}'.da$ is also a prefix of a computation of P. As $\mathbf{e}$ has had all calls to agents familiar with A removed, $\mathbf{e}'$ contains no c-calls. Hence, $\mathbf{c}'(\mathsf{root})_c = \mathbf{c}'.\mathbf{e}'(\mathsf{root})_c$, and so $\mathbf{c}'.\mathbf{e}'.ca$ is also a prefix of a computation of P. (As $\mathbf{e}'$ contains no call to or from agents familiar with A, $\mathbf{e}'$ can now occur before ca, because none of the calls can involve c or a.) As ca does not change the set of secrets familiar to d, and da does not change the set of secrets familiar to c, we get that both $\mathbf{c}'.\mathbf{e}'.ca.da$ and $\mathbf{c}'.\mathbf{e}'.da.ca$ are also prefixes of computations of P.

As $\mathbf{c}'.\mathbf{e}'.ca$ is prefix of a computation of P, $\mathbf{c}'.\mathbf{e}'$ does not have any a-calls and $\mathbf{c}'.\mathbf{e}' \models$

F_cB then from Lemma 3.2.7 we get that $\mathbf{c}'.\mathbf{e}'.\mathbf{f}.bc$ is also a prefix of a computation of P for some $\mathbf{f}$.

We know that $\mathbf{c}'.\mathbf{e}'.\mathbf{f} \models \neg F_bC$, because otherwise bc would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with C in $\mathbf{f}$ and obtain $\mathbf{f}'$. Due to Lemma 3.2.1, $\mathbf{c}'.\mathbf{e}'.\mathbf{f}(\mathsf{root})_b = \mathbf{c}'.\mathbf{e}'.\mathbf{f}'(\mathsf{root})_b$, so $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc$ is also a prefix of a computation of P.

Note that $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc(\mathsf{root})_d = \mathbf{c}'.\mathbf{e}'(\mathsf{root})_d$, because all calls of agents familiar with C were removed from $\mathbf{f}'$ and d is familiar with C already after $\mathbf{c}'$. Hence, $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc.da$ is also a prefix of a computation of P, because da can take place immediately after $\mathbf{c}'.\mathbf{e}'$.

Now due to Lemma 3.2.7 we get that $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc.da.\mathbf{g}.bd$ is also a prefix of a computation of P, because $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc$ does not have any a-calls and $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc \models F_dB$, because $\mathbf{c}' \models F_dB$. We now get a contradiction with Lemma 3.2.3, because in this prefix b calls d even though b is already familiar with D after the bc call in $\mathbf{c}'.\mathbf{e}'.\mathbf{f}'.bc.da.\mathbf{g}.bd$. □

3.3 Conclusion

In this chapter we have begun to analyse the structure of propositional gossip protocols. We have produced useful results in the form of Lemma 3.2.1 and Lemma 3.2.3, which not only help us understand the structure of correct propositional gossip protocols, but propositional gossip protocols in general. Using these results we managed to rule out possible structures for correct propositional gossip protocols, culminating in showing the main result of this chapter, that the undirected communication graph for every correct propositional gossip protocol is complete in Theorem 3.2.9.

Chapter 4

Minimal Number of Calls for Correct Propositional Gossip Protocols

4.1 Introduction

We shall now improve the lower bound on the minimal number of calls required in the worst-case by a correct propositional gossip protocol to terminate in the all-expert situation, improving on the $2n-4$ calls shown for the telephone problem in [29] and thus solving Problem 7 as stated in [9]. To do this, we will be combining results from Chapter 3 on the structure of propositional gossip protocols as well as introducing new results, culminating in Theorem 4.1.1.

We shall introduce the notation $\overline{ab}$, to indicate a call between agents a and b has taken place, but not necessarily initiated by a. Hence this may be either ab or ba, as both would lead to the same gossip situation.

Theorem 4.1.1. *Every correct propositional protocol for $n > 3$ agents generates a computation of length $\geq 2n-2$.*

In other words, Theorem 4.1.1 says that the height of the computation tree of every correct propositional protocol is at least $2n-2$. In order to prove this theorem, we first need to establish many intermediate partial results and analyse different call scenarios.

We shall first begin by showing that at least $2n-2$ calls are needed when we have just 4 agents. We do this in Lemma 4.2.1. This ensures that for the rest of the chapter we only need to focus on situations with at least 5 agents. We then introduce a useful result in Lemma 4.2.2, which states that for a protocol with n agents to correctly terminate in m calls, every agent must be involved in a call after at most $m-n+2$ calls. Furthermore, after $m-n+p$ calls, each secret must be familiar to at least p agents. This is a vitally important result in ensuring we have a method of determining we cannot get a correct protocol in fewer than $2n-2$ from certain gossip situations.

From this background we begin looking at the different possible cases, beginning by building up necessary call structures in a correct propositional gossip protocol. This begins in Lemma 4.3.1, where the idea of a ***CC (cycle component)*** and ***NCC (not cycle component)*** is introduced. This CC is the minimal set of agents which have been involved in a call in a prefix of the form $\overline{ab}.\overline{bc}.\mathbf{c}.ac$ for some three different agents a, b, c and a call sequence $\mathbf{c}$, where no calls involving agents b or c are made in $\mathbf{c}$. This is shown to exist for any correct propositional gossip protocol in 4.3.1. NCC is the set of agents not yet involved in a call at this stage. Further, NCC may form its own cycle component, we refer to this as ***NCC-CC (NCC cycle component)***.

Using this set up, we shall go through 9 different possible cases, showing each possible structure leads to a call sequence of at least $2n-2$ calls being possible. These cases start with the premise that CC has been formed, before analysing the 9 different possible continuations of the computation. Examples of these cases are that all agents have been involved in a call (Lemma 4.4.1) or that in NCC at least two further instances of cycle components can form (Lemma 4.4.14). Finally we shall draw all of these cases together, allowing us to prove Theorem 4.1.1.

4.2 Foundations to Improving the Lower Bound

We begin with some useful results to help narrow down our options and hence simplify the proof of Theorem 4.1.1. We begin by proving the statement for 4 agents. This strengthens Theorem 6 of [9].

Lemma 4.2.1. *Every correct propositional protocol for 4 agents generates a computation of length* ≥ 6.

Proof. In Lemma 5 in [9] it was shown that every gossip protocol has a computation that

starts with the same agent being involved in its first two calls. By relabelling the names of the agents, we can assume that the first call is between a and b, followed by a call between b and c, for distinct agents a, b, c, hence we reach the situation (AB, BAC, CAB, D).

We also know the protocol must allow the call ac to take place at some point. This is because for a to learn C, either ac or ad must be where this secret is learned, as every other agent familiar with C is also familiar with A, by Lemma 3.2.3. However, if a learns C via ad, this call must be already available in the situation (AB, BAC, CAB, D) in which case a does not learn C, and hence must learn C via ac. If the next call is ac we are done, as 3 calls are needed for 3 agents to learn D.

Therefore, assume ac is not a currently available call. ac must be available later, however it is not yet. If d must call either b or c before a, then d will have learned A, and therefore will not initiate a call with a. a currently may not initiate a call with b or c. Therefore, in this scenario, either ad or da is available. After this call, it must be true that a may initiate a call with c. However, we have seen that after ab, either da or ad must be available. This leads to symmetry and the previous argument holds to show that once b has learned C, bd must be an available call.

This leads to two other possible scenarios, where either a or b is involved in the first 3 calls made. Therefore both $(ABCD, BA, CABD, DAB)$ and $(AB, BACD, CAB, DABC)$ are possible. Consider the first instance. For this to be correct either all 3 calls are available, hence 6 can be made, or just bc and dc. Similarly, in the second instance we must have ad and cd available. Hence, DAB is a guard for dc, and CAB is a guard for cd. Hence, we have shown (ABD, BAC, CAB, DAB) is a possible situation after 3 calls, in which all 3 calls are available to the scheduler. The scheduler would simply initiate cd (or dc) first. We are in the situation $(ABD, BAC, CABD, DABC)$ with ac and bd available to us. After these two calls all 4 agents are experts however it has taken 6 calls. □

Due to Lemma 4.2.1, in the rest of this chapter we will assume that $n > 4$. We now notice a basic fact about gossip situations if we hope to have a successful protocol with only a certain number of calls remaining.

Lemma 4.2.2 (Conversation)**.** *For a protocol on n agents to correctly terminate in m calls, every agent must be involved in a call after at most $m - n + 2$ calls. Furthermore, after $m - n + p$ calls, each secret must be familiar to at least p agents.*

Proof. For an agent a and its secret A, each call can increase the number of agents that are familiar with A by at most 1. If a has not yet been involved in any calls, then the only

agent which is familiar with A is a. If after $m-n+2$ calls, a has not yet been involved in a call, then a is the only agent which is familiar with A. However, only $n-2$ calls remain for $n-1$ agents to learn A, which is impossible.

Similarly, if after $m-n+p$ calls, fewer than p agents are familiar with A, then $n-p$ calls remain for at least $n-p+1$ agents to learn A. Again, this is impossible as at most one agent can learn A in each call. □

4.3 Basic Call Structure of a Correct Propositional Protocol

We will now show a necessary possible structure of a correct propositional protocol. Having a starting structure which must be possible for a correct propositional gossip protocol enables us to use this as the basis for our proof of Theorem 4.1.1.

Lemma 4.3.1. *Every correct propositional protocol has a prefix of a form* $\overline{ab}.\overline{bc}.\mathbf{c}.ac$ *for some three different agents* a, b, c *and a call sequence* $\mathbf{c}$*, without any calls involving agents* b *or* c*, such that no agent* x *where* $\overline{ab}.\overline{bc}.\mathbf{c} \not\models F_aX$ *is involved in a call in* $\mathbf{c}$*. We will later refer this minimal set of agents which have been involved in a call in this prefix as* **CC (cycle component)** *and agents not involved in a call in this prefix as* **NCC (not cycle component)**.

Proof. In Lemma 5 in [9] it was shown that every gossip protocol has a computation that starts with the same agent being involved in its first two calls. By relabeling the names of the agents, we can assume that the first call is $\overline{ab}$, i.e. between a and b, followed by a call $\overline{bc}$, i.e. between b and c, hence we get the situation where a is initially involved in a call with an agent b, who is then involved in a call with an agent c.

Now, every agent which is familiar with C is also familiar with A. Hence by Lemma 3.2.3, no agent which is familiar with C will ever initiate a call with a, and so a must learn C in a call initiated by a himself. Also by Lemma 3.2.3, a will never have a further call with b. Now, assume ac is never available. Agent a must learn C from initiating a call with an agent, say d.

Let us pick a computation where a is familiar with the greatest number of secrets before learning C. In other words, if $\mathbf{c}.ad$ is a prefix of a computation which starts with a call between a and b, followed by a call between b and c, and where a learns C during the call ad, we require the size of $\mathbf{c}(\mathsf{root})_a$ to be the largest possible. This is well-defined as this value is an integer between 1 and n.

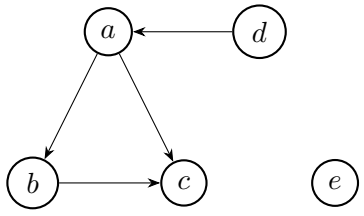

Figure 4.1: Directed communication graph for the call sequence $ab.bc.da.ac$ in example 4.3.2

We know that $\mathbf{c} \models \neg F_a D$, because otherwise ad would not be possible due to Lemma 3.2.3. We can then remove all calls of agents familiar with D in $\mathbf{c}$ and obtain $\mathbf{c'}$. Due to Lemma 3.2.1, $\mathbf{c'}.ad$ is still a valid prefix of a computation. Note that d cannot possibly be familiar with C (nor any other secret apart from his own) after $\mathbf{c'}$, because all his calls were removed. At the same time we know that $\mathbf{c}(\mathsf{root})_a = \mathbf{c'}(\mathsf{root})_a$.

Hence, there exists a longer prefix, $\mathbf{c'}.ad.\mathbf{d}.ae$, of this computation such that a finally learns C during ae. Therefore, $\mathbf{c}(\mathsf{root})_a$ is smaller than $\mathbf{c'}.ad.\mathbf{d}(\mathsf{root})_a$, because the latter includes at least one more secret (namely D); this is a contradiction with the assumption that a knew the most number of secrets before learning C in the $\mathbf{c}.ad$ prefix of a correct computation.

Due to Lemma 3.2.1 we may remove all calls involving agents x such that $\overline{ab}.\overline{bc}.\mathbf{c} \not\models F_a X$, as well as any calls with agents familiar with C (hence b and c), to get the prefix $\overline{ab}.\overline{bc}.\mathbf{c'}.ac$, where no such calls have been made in $\mathbf{c}$. □

Example 4.3.2 (An example of forming CC and NCC)**.** *Consider the agents a, b, c, d, e and call sequence $ab.bc.da.ac$. Here we would consider the agents a, b, c, d to belong to CC, and e to belong to NCC.*

Lemma 4.3.1 gives us a certain prefix which must be available for all correct propositional protocols. From here we consider different case scenarios in order to get our desired result. Before going to the case analysis we need one more important result.

From Lemma 4.2.2, if we are aiming to create a correct protocol with $2n - 3$ calls or fewer, then all agents must be involved in a call after at most $n - 1$ calls. This gives us a useful cut-off point when looking at how many calls we have remaining before all agents must have been involved in a call.

Lemma 4.3.3. *For a protocol on n agents to correctly terminate in m calls, there must be no prefix of $m - n - r + 4$ calls such that r agents have not been involved in a call.*

Proof. In this situation a total of $n+r-4$ calls remain. From Lemma 4.2.2, as every agent must be in a call after at most $m - n + 2$ calls, this is the same as saying there must be no prefix in which r agents have yet to have been in a call, with only $r - 2$ calls remaining until all agents must have been in a call.

Consider an agent not yet involved in a call. If this agent calls an agent already involved in a call then we have $r-1$ agents which have not yet been in a call and $r-3$ calls remaining until all must have been in a call. This leaves us in the same position, but with fewer agents yet to be involved in a call, and fewer calls. If this continues, we would reach a state with no calls remaining before all agents must have been in at least one call, and two agents not yet in a call. We can also make all calls currently available from all agents which have been involved in calls so far.

Again this will, at best, still leave us in the scenario where there are $r - 1$ agents left which have not yet been in a call and $r - 3$ calls remaining before all must have been involved in a call, but now with no calls directly from or to any agents which have already been involved in a call. If these were the only options, we could repeatedly apply them until we are left with two agents and zero calls, in which case we would have failed. Therefore, there must be a call between two agents not yet involved in any call.

Consider any pair of such agents. If, after they call each other, one (or both) of these agents could now call an agent already involved in a call then we are again in the scenario where we have $r - 2$ agents which have not yet been in a call and $r - 4$ calls remaining before all must have been involved in a call; the exact situation in which we started, but now with fewer agents which have not yet been in a call and fewer calls remaining. If these two agents were the last two yet to be involved in a call, then we were already done before this call took place, as we would have two agents left but zero calls. Therefore there must be two agents a and b, such that there is a call between a and b, which cannot be immediately followed by a call to any agent already in a call.

Assume neither a or b now wishes to initiate a call. As there trivially must be more than two agents in total, and this was the first call for both agents, neither a or b is currently an expert, meaning they must be involved in future calls. As neither wishes to initiate a call, this means they must be called, say by an agent c after a call sequence $\mathbf{c}$.

By relabelling, let c call a, i.e. $\mathbf{c}.ca$ is a prefix of a computation. In this situation, by

Lemma 3.2.3, c was not familiar with A, hence, by Lemma 3.2.1, there exists a call sequence $\mathbf{c'}.ac$ where $\mathbf{c'}$ is $\mathbf{c}$ where all calls that involve an agent familiar with A are removed. But this means either a is called by an agent which has already been involved in a call, or we can create the situation from Lemma 4.3.1.

Therefore, either a or b wishes to initiate a call with another agent c, which has yet to be involved in a call. Hence, every agent which is familiar with C is also familiar with A. Therefore, by Lemma 3.2.3, no agent with C will ever initiate a call with a. Therefore a must make the call to learn C.

From Lemma 4.3.1 it then follows that there must be a call sequence $\mathbf{c}$ such that after this sequence ac is available. This sequence must involve a, and hence can be formed in such a way as to contain at most p new agents for the p new calls involved. If all agents have now been involved in a call we are done due to Lemma 4.2.2, because we have used n calls and yet only two agents are familiar with C. But, if more agents are yet to be involved in a call, then after ac we have at most $r - p - 5$ calls remaining for at least $r - p - 3$ agents to be involved in a call. We remain in the exact situation as we started, but now with fewer agents which have not yet been involved in a call and fewer calls remaining. We may try to repeat this process, however as there is a finite number of agents and calls, we will eventually be left with no calls remaining and agents not yet involved in a call. □

Note that Lemma 4.3.3 already shows that every correct propositional gossip protocol has to use at least $2n-3$ calls in the worst case, because we begin with $n-2$ calls remaining for n agents yet to be involved in a call.

Lemma 4.3.4. *For a correct propositional gossip protocol with n agents, if there are x calls remaining then each secret must be familiar to at least $n - x$ agents. If only $n - x$ agents are familiar with a secret, say, A, then in every future call one of the agents involved gets to be familiar with A. Furthermore, if only $n - x$ agents are familiar with A, with x calls remaining and y agents are currently experts, then at least $n - x - y$ agents must be active.*

Proof. As each call can increase the number of agents which are familiar with a secret by at most one, if there are x calls remaining then each secret must be familiar to at least $n - x$ agents.

If we have x calls remaining and only $n - x$ agents are familiar with A, then in the remaining x calls, x agents must learn A. At most one agent may learn A in each call,

therefore every future call must have an agent learning A. In other words, every call must be between an agent that is not familiar with A and an agent that is familiar with A.

As every call must pass on A, after a call is made both agents involved are now familiar with A. All agents which are not experts must be involved in at least one more call to become an expert. This means we have at least $n-x-y$ non-expert agents that are familiar with A and need to be in at least one more call each. We claim that in this situation all such agents have to be a caller or a callee of an active call which would then imply that at least $n - x - y$ agents are active.

Notice that if fewer than $n - x - y$ agents are active, then not all of the $n - x - y$ non-expert agents that are familiar with A may be a caller or a callee of an active call. This is because no agent that is familiar with A may call another agent that is familiar with A, so at most one such agent may be involved in each of these calls.

Suppose, that there is a non-expert agent, denoted by b, that is familiar with A but is not a caller nor a callee in any active call. Let us execute in any order all active calls of agents that are not familiar with A. Note that all these calls are made to agents that are familiar with A and they can make at most one such a call, because they get to be familiar with A afterwards. Moreover, executing any such call does not activate nor deactivate any calls of other agents that are not familiar with A, because they cannot even observe that such a call was made. Hence, no new calls become active and so none of these calls can be to b, because we assumed he is not a callee in any active call.

Finally, after all these calls are made, agent b is familiar with A and is still not an expert and is still not active, because he was not involved in any call. However, all active agents are now familiar with A and therefore cannot call b. Moreover, all active agents in the future must be familiar with A, because an agent can only become active by being called. So agent b will never be called nor call anyone to become an expert; a contradiction. □

4.4 Different Call Scenarios

Let us fix a correct propositional gossip protocol, P, for $n > 4$ agents. We are going to split all its computations into different call scenarios based on specific conditions being satisfied by their prefix (corresponding to different lemmas in this section). In each of them we will show that at least $2n - 2$ calls is made by the protocol in the worst case. In all these call scenarios we assume that the call sequence starts with a prefix as in Lemma 4.3.1, i.e., the first two calls made are between agents a and b and then b and c, and the call ac occurs

after a number of calls in order to form CC.

Lemma 4.4.1. *If the call ac is made when a is familiar with all secrets apart from C, then there exists an extension of this prefix which formed CC, in which at least* $2n-2$ *calls are made.*

Proof. As after a and b have had their call, no calls in the sequence leading to ac may involve C, these can occur before b and c have had their call. Therefore, after at least $n-2$ calls we can reach the position where a is familiar with all secrets apart from C, and c has not yet been in any calls. From here, if we are to complete the protocol in $2n-3$ calls, we have just $n-1$ calls remaining and $n-1$ agents that are not familiar with C, hence in every call an agent must learn C. As ac is active let us perform this call.

Now a and c are both experts, and the only agents which are familiar with C. This means that every call must now involve an agent calling an agent which is familiar with C. However, assume in this state an agent d can call a. If we took da before ac, then we would have used $n-1$ calls, and therefore $2n-3$ would not be possible as only one agent is familiar with C from Lemma 4.2.2. Hence all calls must be to c. This means that if we go back to the situation before ac, every agent must be able to call c. However, if we go back to the original case now, where b and c have had their call, this means we have used $n-1$ calls, but the other $n-2$ agents can now initiate a call with c. This takes us to $2n-3$ calls yet b is not an expert. □

Lemma 4.4.2. *If the call ac is made when a is familiar with all secrets apart from C and D, then there exists an extension of this prefix which formed CC in which at least* $2n-2$ *calls are made.*

Proof. We may use Lemma 3.2.1 to remove all calls involving d. Hence, after ac this leaves just $n-2$ calls, yet there are still $n-1$ agents which are not familiar with D. □

Example 4.4.3 (An example of the case given in Lemma 4.4.1)**.** *Consider a correct propositional gossip protocol with the agents* a, b, c, d, e *and call sequence ab.bc.da.ea.ac. From here, Lemma 4.4.1 shows that there exists an extension of this prefix, in which at least* $2n-2$ *calls are made.*

We can extend Lemma 4.4.2 to get an even stronger result.

Lemma 4.4.4. *Assume that at least one agent, denoted by d, is not involved in any call when the call ac is made. Then for any extension of this prefix which formed CC such that*

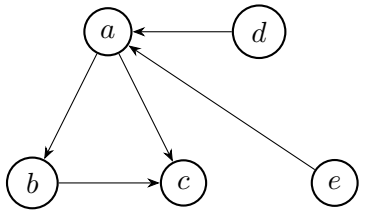

Figure 4.2: Directed communication graph for the call sequence *ab.bc.da.ea.ac* in example 4.4.3

all agents apart from one were involved in a call with an agent from CC, which is then extended, there exists a further extension in which at least $2n-2$ *calls are made.*

Proof. In this situation, CC has $n-1$ agents, and hence at least $n-1$ calls. This leaves just $n-2$ calls, yet there are still $n-1$ agents which are not familiar with the secret of the agent which has not yet been involved in a call. □

Lemma 4.4.5. *If the call ac is made when a is familiar with all secrets apart from C and two others, denoted by D and E, then there exists an extension of this prefix which formed CC in which at least* $2n-2$ *calls are made.*

Proof. Let us follow this until after *ac*, where by Lemma 3.2.1, neither d nor e have yet been involved in a call. If either d or e could be called, or could call, an agent in CC we would be done, as this would be $n-1$ calls and there should still be an agent not yet in a call. For the same reason no agent in CC can initiate a call in this state. Hence, there must now be a call between d and e. After this call (and in fact before), every future call must lead to an agent learning D and an agent learning E. This means that at least two agents must be active here by Lemma 4.3.4. Hence, after the call between d and e, both d and e are active, and are the only agents which are active.

Now, we are in a situation where D and E must be learned in all future calls, as these secrets are familiar to only two agents after $n-1$ calls, however, C is currently familiar to only three agents, meaning it must be learned in all but one of the future calls. This implies either d or e must now be able to initiate a call with an agent which is familiar with C.

Assume neither d nor e wishes to initiate a call with c in this state. As the call between d and e was not dependent on C, these calls could occur before any agent has been involved in a call with c, by Lemma 3.2.1. We could reach a state where b has not been involved in

a call with c, and ac is now available. This would mean after $n-3$ calls, c has yet to be involved in a call. If now neither d nor e wished to initiate a call with c, we could do both of these calls, meaning after $n-1$ calls c has not yet been involved in a call, and hence by Lemma 4.2.2 this is now impossible to be correct in $2n-3$ calls or fewer in the worst case. Hence after d and e have been in their call, at least one of these agents wishes to initiate a call with c.

Assume that dc and ec are available. Once the first of these calls are made, the second call involves no agent learning D or E.

Assume then that da and ec are available. Once these calls are made, all four of these agents would be experts. This would mean no agents are active so no further calls could occur, yet b is not an expert. Assume that db and ec are available. Once these calls are made all four of these agents would be familiar with A. This would mean no agent will ever initiate a call with a, and so a will never become an expert. This proves the statement for $n = 5$.

If $n > 5$, then we have an agent f, which is not a, b, c, d or e. Assume then that df and ec are available. As neither d nor f are familiar with C, after df all future calls must lead to an agent learning not only D and E but also C. As these calls were not dependent on C, these calls could occur before any agent has been involved in a call with c. We could reach this state where b has not been involved in a call with c, and ac is now available. After df (but not ec), $n-2$ calls have been made, and no agent has been involved in a call with c, hence the next call must be to c.

This means that if either d or f wished to initiate a call with any agent other than c then we would be done. So now if we consider this state but with the original prefix with CC completed first, all active agents now wish to initiate a call with c, at which point they would become experts and never initiate another call. Hence, neither a nor b will ever become an expert. □

Example 4.4.6 (An example of the case given in Lemma 4.4.5)**.** *Consider a correct propositional gossip protocol with the agents a, b, c, d, e and call sequence ab.bc.ac. From here, Lemma 4.4.5 shows that there exists an extension of this prefix, in which at least $2n-2$ calls are made.*

Lemma 4.4.7. *Assume that the call ac is made when at least three agents were not involved in any call. Assume further that we may extend this prefix such that for agents not yet in a call, i.e. NCC, a second cycle component is formed. This begins with $\overline{xy}$, followed by*

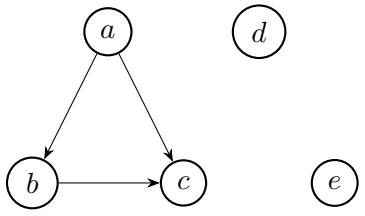

Figure 4.3: Directed communication graph for the call sequence *ab.bc.ac* in example 4.4.6

$\overline{yz}$ and then a call sequence **d** *involving all agents not yet involved in a call, before ending with xz. Then at least $2n-2$ calls can be made.*

Proof. In this scenario, exactly n calls have been used. This means every secret must be familiar to at least three agents, if we were to complete in $2n-3$ calls from Lemma 4.2.2. We find that both C and Z are familiar to exactly 3 agents, and hence must be learned in all future calls. Hence, the next call must be from and to one of a, b, c and x, y, z.

Furthermore, these 3 calls must all be available from Lemma 4.3.4. Assume there is no call between x, y, z and a. Every call must have an agent learn both C and Z, so any agent which is familiar with exactly one of these secrets must call an agent which is familiar with the other. Therefore, if there is no call between x, y, z and a, then a may not be in any future calls, as these are the only 3 agents which will be familiar with Z and not C, else another call has been made to an agent and C has not been learned. So, in this state these three calls must be available and no others, all calls must be between a, b, c and x, y, z, with each involved exactly once.

This means that either, at least two of these calls are initiated from agents in CC, or at least two of these calls are initiated from agents in NCC. As these are independent, we do not know which occurs first, and hence, due to relabelling, we can say that at least two of these calls are initiated by agents in CC. But assume NCC is still in the position where only two calls have been made, between x and y and then between y and z. Now let the two calls from a, b, c to x, y, z happen. In this position t agents have not yet been involved in a call after $n-t+1$ calls. Hence $t-2$ calls remain before all these t agents must have been involved in a call. The result follows from Lemma 4.3.3. □

Example 4.4.8 (An example of the case given in Lemma 4.4.7)**.** *Consider a correct propositional gossip protocol with the agents a, b, c, x, y, z and call sequence ab.bc.ac.xy.yz.xz. Now, Lemma 4.4.7 shows that at least $2n-2$ calls can be made.*

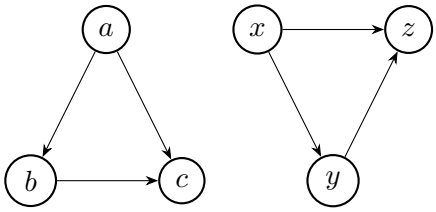

Figure 4.4: Directed communication graph for the call sequence *ab.bc.ac.xy.yz.xz* in example 4.4.8

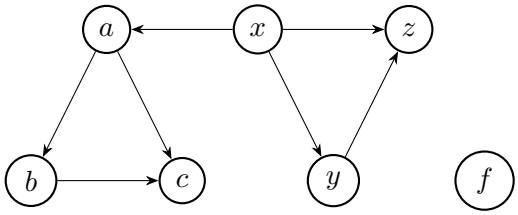

Figure 4.5: Directed communication graph for the call sequence *ab.bc.ac.xy.yz.xz.xa* in example 4.4.10

Lemma 4.4.9. *Assume that the call ac is made when at least three agents were not involved in any call. Assume further that we may extend this prefix such that for agents not yet in a call, i.e. NCC, a second cycle component is formed, beginning with* $\overline{xy}$*, followed by* $\overline{yz}$ *and then a call sequence* **d***, leaving at least one agent not yet involved in a call, before ending with xz. Assume also from this NCC connected cycle component (now referred to as* **NCC-CC***) an agent wishes to initiate a call with an agent in CC (or be called by an agent in CC). Then at least* $2n-2$ *calls can be made.*

Proof. Let there be t agents not yet involved in a call. After the call from an agent in NCC-CC to an agent in CC (or from CC to NCC-CC), $n-t+1$ calls have been made, yet t agents have yet to be involved in a call. Hence $t-2$ calls remain before all these t agents must have been involved in a call. The result follows from Lemma 4.3.3. □

Example 4.4.10 (An example of the case given in Lemma 4.4.9)**.** *Consider a correct propositional gossip protocol with the agents* a, b, c, x, y, z, f *and call sequence ab.bc.ac.xy.yz.xz.xa. Now, Lemma 4.4.9 shows that at least* $2n-2$ *calls can be made.*

Lemma 4.4.11. *Assume ac can occur after some call sequence to form CC, leaving at least 3 agents not involved in a call.*

Consider this prefix being extended by including all calls possible which can be initiated by the agents in CC, or directly to agents in CC. This forms a larger CC, leaving at least 3 agents not involved in a call.

Assume further that we may extend this new prefix such that for agents not yet in a call, i.e. the updated NCC, a second cycle component is formed, beginning with $\overline{xy}$, followed by $\overline{yz}$ and then a call sequence **d**, *leaving at least one agent not yet involved in a call, before ending with xz.*

Assume in this situation no agent from NCC wishes to initiate a call with CC until it is familiar with all secrets of agents in NCC. Then at least $2n-2$ calls can be made.

Proof. If by extending the prefix by making all calls to or from an agent in CC a second cycle is formed in CC then we are done by 4.3.3. Hence, the secret of the last agent to be called/call in this prefix from outside CC is familiar to just two agents, the agent itself and the agent in CC (or its extension) which it was in a call with. This would mean that $n-2$ further calls in which this secret is exchanged will be required. If now no agent from NCC called any agent from CC until it was an expert in NCC, then n calls have been used with only two agents being familiar with this secret, hence we are done.

For an agent to be familiar with all secrets of agents in NCC, assuming CC and NCC-CC have been formed, there must have been a total of at least n calls. This means every secret in this scenario must be familiar to at least three agents by Lemma 4.2.2. Before the final call in NCC, where two agents must become experts in NCC, no agent wishes to initiate a call with an agent in CC by assumption. If after this call, there were any further calls internally in NCC, then we would have used $n+1$ calls, and yet only three agents would be familiar with C, and hence we would be done.

Therefore, every future call must involve C. However, only these two agents which are experts in NCC may be active, and yet we have three agents which are familiar with C and are not experts. The result follows from Lemma 4.3.4. □

Lemma 4.4.12. *Consider a prefix such that ac is made when at least 3 agents are not involved in any call. Assume that there is no extension of this prefix such that agents in NCC can form a cycle without a call with an agent in CC. Then at least $2n-2$ calls can be made.*

Proof. As we know there must be a call available between agents in NCC. We are left with two cases. Either, after this call there is a possible call to another agent in NCC, forming

the situation where x must be able to call z at some future instance, or no such second call exists, and the largest connected component of agents in NCC without a call to CC is 2.

If only two agents remain not involved in a call after CC has formed with ac this is just Lemma 4.4.5.

Consider case 1 where there is a possible call to another agent in NCC, forming the situation where x must be able to call z at some future instance. As a call to CC is needed before xz occurs, this means that for x to learn the required secrets at least k calls are needed, with only a maximum of $k-1$ of these secrets coming from agents which had previously not yet been in any calls. If not all secrets of agents in NCC were needed before xz (as well as Z) then we are in the same position as in Lemma 4.4.9, so we are done. If all NCC secrets were needed before xz (apart from Z) then we are in a situation where $n+1$ calls have been made, and yet only two agents are familiar with Z, hence we are done by Lemma 4.2.2.

Consider case 2 where no such second call exists, and the largest connected component in NCC without a call to CC is two. We can consider these two agents to be the last two agents in NCC, with all other agents being connected to CC already in the graph of calls made. If this was not the case then by calling CC now would mean we have added two agents to CC in two calls, and would therefore need another call between two agents which have not yet been involved in a call, which is the same position we started in. This would be the scenario after n calls, and so now every call must involve an agent learning both of these secrets.

Let us call these agents x and y, with secrets X and Y. As xy was forced (if any other call was possible, then we would have used $n-1$ calls without one of x or y being in a call and therefore be done by Lemma 4.2.2), no other agents may be active, and therefore by Lemma 4.3.4 both x and y must still be active in this scenario. Also, by our original assumption, both of these calls must be to agents in the original CC. But now, if immediately after ac, we had xy and then both of these calls from x and y back to agents in CC, t agents have yet to be involved in a call, but $n-t+1$ calls have been made. Hence $t-2$ calls remain before all these t agents must have been involved in a call. The result follows from Lemma 4.3.3. □

Example 4.4.13 (An example of the case given in Lemma 4.4.12)**.** *Consider a correct propositional gossip protocol with the agents a, b, c, x, y, z, f and call sequence $ab.bc.ac.xy.yz.xa.xz$. Now, Lemma 4.4.12 shows that at least $2n-2$ calls can be made.*

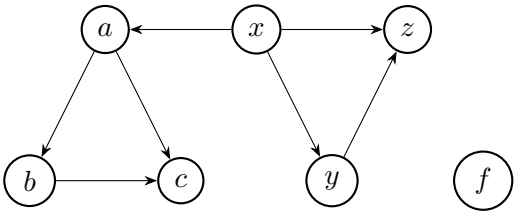

Figure 4.6: Directed communication graph for the call sequence $ab.bc.ac.xy.yz.xa.xz$ in example 4.4.13

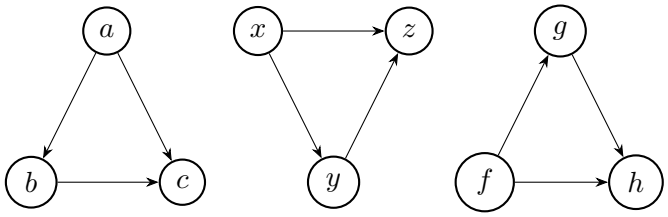

Figure 4.7: Directed communication graph for the call sequence $ab.bc.ac.xy.yz.xz.fg.gh.fh$ in example 4.4.15

Lemma 4.4.14. *Assume ac can occur after some call sequence to form CC, leaving at least 3 agents not involved in a call.*

Assume further that in NCC at least two further instances of cycle components can form, all disconnect from each other. Then at least $2n - 2$ *calls can be made.*

Proof. As eventually the protocol must connect all agents, these three (or more) cycles must join each other. However, two of these must be able to join to form a bigger connected component without the others being involved initially. Hence we could do this before any calls have been made in the other components at all. This takes us to the same position as in Lemma 4.4.9. □

Example 4.4.15 (An example of the case given in Lemma 4.4.14)**.** *Consider a correct propositional gossip protocol with the agents* $a, b, c, x, y, z, f, g, h$ *and call sequence* $ab.bc.ac.xy.yz.xz.fg.gh.fh$*. Now, Lemma 4.4.14 shows that at least* $2n - 2$ *calls can be made.*

Lemma 4.4.16. *Assume ac can occur after some call sequence to form CC, leaving at least 3 agents not involved in a call.*

Assume further that we may extend this prefix such that for agents not yet in a call, i.e. NCC, a second cycle component is formed.

Assume that this new prefix can be further extended such that we can reach a situation where these components remain disconnected, but possibly more calls have been made until we are left with these two connected cycle components, and finally two agents not yet in a call. Then at least $2n - 2$ *calls can be made.*

Proof. We will firstly show that no calls could be made involving these two components, and that in fact after these components are formed we are left with just two agents. There are a few stages to this. Firstly, this call between the final two agents must be the only call available at the time, so no other agent is active. Also, by Lemma 4.3.4 both these agents must be active after this call.

If, after the other two components had their ac call, either of our final two agents x and y could call an agent not yet in a call, this would mean we are again in a position where xz must be available, either with calls to other connected components, which fails by Lemma 4.4.9, or without a call to other connected components which fail by Lemma 4.4.14. So x and y must call agents which have been in calls by the time the ac calls have been made. Now consider the calls available to x and y. Both are to agents already in calls, and therefore, after these two calls we have made $t + 1$ calls for t agents. If any other agent remained not involved in a call, at this stage we are done by Lemma 4.3.3. Therefore, no other agents may exist.

The calls x and y make may not be to the same connected component. If they were, we are again in the situation where we have made $t + 1$ calls for t agents, and yet more agents exist, so we are done by Lemma 4.3.3.

Now consider where in the connected components the calls from x and y may go. All calls need to pass on X and Y, but also as only three agents are familiar with each of the respective C secrets, only one future call may lead to an agent not learning these. As x and y call opposite connected components, this must be the only call which doesn't involve it. This means we must call A_1, A_2, B_1, B_2, C_1 or C_2. As these calls were available independent of the two CCs, they may happen before a_1c_1 or a_2c_2, so must not be to either a_1 or a_2. As neither these calls, nor any calls in either CC are dependent on the bc calls, it could be before these happen also, and hence must be to c_1 and c_2.

Now, after xc_1 and yc_2, all calls must pass on X, Y, C_1 and C_2. Hence, by Lemma 4.3.4, all four of these agents must still be active (as there are certainly no experts). The

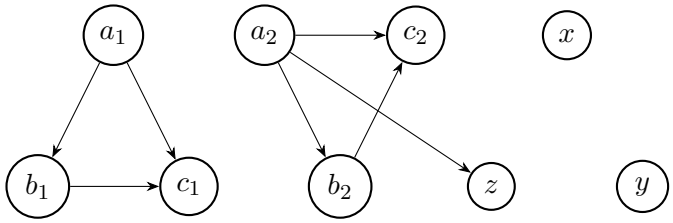

Figure 4.8: Directed communication graph for call sequence $a_1b_1.b_1c_1.a_1c_1.a_2b_2.b_2c_2.a_2c_2.a_2z$ in example 4.4.17

only possible solution is for x and c_1 to call a_2 and b_2 and for y and c_2 to call a_1 and b_1. But again, x and c_1 calling a_2 and b_2 are independent of C_2, and so could even happen immediately after a_2b_2, and lead us to failure by Lemma 4.3.3. □

Example 4.4.17 (An example of the case given in Lemma 4.4.16)**.** *Consider a correct propositional gossip protocol with the agents* $a_1, b_1, c_1, a_2, b_2, c_2, x, y, z$ *and call sequence* $a_1b_1.b_1c_1.a_1c_1.a_2b_2.b_2c_2.a_2c_2.a_2z$*. Now, Lemma 4.4.16 shows that at least* $2n-2$ *calls can be made.*

4.5 The Lower Bound

We are finally ready to prove Theorem 4.1.1, which establishes a $2n-2$ lower bound on the number of calls used by a correct propositional protocol in the worst case.

Proof. In Lemma 4.3.1 we have shown a necessary possible structure for all correct propositional protocols, where CC is formed. We then examined every possibility that can take place from this structure.

In Lemma 4.4.1 we showed that should CC contain all agents then no correct propositional protocol may exist with fewer than $2n-2$ calls. This meant at least one agent must not have been involved in the calls leading up to ac, and hence it must be possible that after ac at least one agent has not yet been involved in a call.

By Lemma 4.4.2, Lemma 4.4.4 and Lemma 4.4.5 we see that it must be possible for there to be at least three agents not yet in a call at this stage.

In Lemma 4.4.7 we showed that these remaining agents cannot form a complete cycle component using all remaining agents themselves. We also show in Lemma 4.4.9 that if an agent remains which has still not been involved in a call after this second cycle component

has formed, then these two cycle components cannot immediately call each other. In Lemma 4.4.11 we also showed that once the second cycle component is formed, waiting for an expert in NCC does not work either. We also showed that this is the case if we extend CC my making all possible calls first. In Lemma 4.4.12 we showed that there must be a connected cycle component in NCC which does not call CC before it is completed.

We have shown a connected cycle component must form in NCC, but not yet be able to call or be called by CC. In Lemma 4.4.14 we showed that we could not form three (or more) independent cycle components, so the remaining agents may not form another connected cycle component themselves. If there was just one agent left we would be done by Lemma 3.2.3. Finally in Lemma 4.4.16 we showed that once two cycle components are formed, if we have two additional agents which have only been involved in a call with each other then this will not give us a protocol with fewer than $2n - 2$ calls either.

Together, this shows that every correct propositional protocol for $n > 3$ agents generates a computation of length $\geq 2n - 2$. □

4.6 Conclusion

In this chapter we have been focused on proving Theorem 4.1.1, that every correct propositional protocol for $n > 3$ agents generates a computation of length $\geq 2n - 2$, solving Problem 7 as stated in [9]. This shows propositional gossip protocols cannot require as few calls as general epistemic gossip protocols, shown to be $2n - 3$ in [9]. In proving Theorem 4.1.1 we have also established results regarding the structures of correct propositional gossip protocols. These may be useful when trying to improve this lower bound further, or when trying to design correct protocols.

Chapter 5

Decision Problems for Propositional Protocols

5.1 Introduction

Having shown that no correct propositional gossip protocol exists whose undirected communication graph is not complete in Chapter 3, and investigated the number of calls required in a correct propositional gossip protocol in Chapter 4, we now investigate further the problem of correct propositional gossip protocols, if a given propositional gossip protocol is correct, with the intention of determining the complexity of the problem. Knowing the complexity of such a problem not only details its difficulty, and a sense of the time required to solve, but also increases the understanding of the problem. Knowing the problem's complexity gives insight to the feasibility of effectively using such protocols in specific applications.

5.2 The Complexity of Propositional Gossip Protocol Problems

We now move on to analysing the computation complexity of important decision problems for propositional gossip protocols such as checking termination, partial correctness and correctness. We say a protocol terminates if all computations are finite. We first establish a necessary and sufficient condition for a propositional protocol to terminate.

Lemma 5.2.1. *A propositional protocol generates an infinite computation if and only if a call is made without the caller gaining new information within the first n^2 calls.*

Proof. ($\Leftarrow$) Consider a propositional gossip protocol P. Let $\mathbf{c}.ab$ be a prefix of a computation of P such that $\mathbf{c}(\mathsf{root})_a = \mathbf{c}.ab(\mathsf{root})_a$. Hence there exists a rule $\psi \to ab$ for agent a such that $\mathbf{c} \models \psi$. But we clearly have then $\mathbf{c}.ab \models \psi$, because ψ only depends on the secrets agent a is familiar with and they do not change after ab is made. Therefore, call ab can be performed again after $\mathbf{c}.ab$. It is easy to see that $\mathbf{c}(\mathsf{root})_a = \mathbf{c}.(ab)^k(\mathsf{root})_a$ and that $\mathbf{c}.(ab)^k$ is a prefix of a computation of P for any k, so P may not terminate.

($\Rightarrow$) Consider any infinite computation $\mathbf{c}$ of P. Along this computation, the current gossip situation can change at most n^2 times, because each agent can be familiar with at most n secrets. So within the first n^2 calls of $\mathbf{c}$ there has to be a call after which the caller does not learn any new secrets. □

As we will see, the previous lemma suffices to establish that all the decision problems studied in this section are in coNP. To show them to be coNP-hard, we show three different but similar reductions from the well-known 3-SAT problem. This is the problem of Boolean satisfiability, where each clause is in conjunctive normal form, and each clause is limited to at most three literals, which was found to be an NP-complete problem, and used to show other problems are also NP-complete [21].

Theorem 5.2.2. *The problem of checking if a given propositional gossip protocol terminates is coNP-complete.*

Proof. First, we show the problem to be in coNP. Due to Lemma 5.2.1, to show *non-termination*, it suffices to guess a call sequence $\mathbf{c}$ of length n^2 and a rule $\psi \to ab$ of the protocol such that $\mathbf{c} \models \psi$ and $\mathbf{c}(\mathsf{root})_a = \mathbf{c}.ab(\mathsf{root})_a$. All of that is of polynomial size and can be checked in polynomial time.

To show the problem is coNP-hard we will create a polynomial time reduction from the 3-SAT problem, such that termination's NO instances match with 3-SAT's YES instances. We will call this special agent the final agent and denote him by f. The basic idea of this is to have an agent which will learn a certain set of secrets iff the original problem is satisfiable. Only when this agent learns all these secrets will he be able to make a call that can be repeated indefinitely resulting in a non-termination of the protocol.

For each Boolean variable, x, used in the 3-SAT formula, we will create three agents: variable agent (x), true agent ($x_\top$), and false agent ($x_\bot$). For every $i \leq m$, where m is

the number of clauses in the 3-SAT formula, we have agent c_i that corresponds to the i-th clause.

Now we define the rules of the protocol for each of the agents. For each variable agent, x, while this agent is only familiar with its own secret X, x may initiate a call with agent $x_\top$ or agent $x_\bot$ with the guard $(\neg X_\top \wedge \neg X_\bot)$ and the scheduler will decide which of these calls takes place. This essentially sets the value of the x variable in the 3-SAT formula to true or false, respectively.

Now any true agent, $x_\top$, and false agent, $x_\bot$, that was called by his corresponding variable agent can make a call to any clause agent, c, whose corresponding clause in the 3-SAT formula uses the literal x or $\neg x$, respectively. This is done with the guard $(X \wedge \neg C)$, to ensure this call cannot be repeated.

Once a clause agent receives a call, this means that the corresponding clause is satisfied by the current truth assignment. Once this happens, the clause agent calls f, with a guard $(X \wedge \neg F)$, to make a call to f that cannot be repeated.

Now, f will learn the secrets of all clause agents only if all clauses have been satisfied. Agent f can only make a call to a fixed variable agent x if he learns all of these clause secrets, with a guard $(C_1 \wedge ... \wedge C_n)$, and hence this guard will remain true, leading to the protocol not terminating.

The only agent that can repeat a call, and therefore lead to the protocol not terminating, is f, and this may only happen if all clauses are satisfied at the same time. Hence, this corresponds directly to YES instances of 3-SAT. If we had a NO instance, it would be impossible for f to learn all clause secrets, and therefore the protocol would terminate as each call is made at most once. It is easy to see that this whole construction can be done in polynomial time and size. □

Definition 5.2.3. *$a : A \wedge \neg B \rightarrow ab$ states that agent a will initiate a call with b if the guard $(A \wedge \neg B)$ is satisfied for agent a.*

In the following examples we shall use the 3-SAT instance $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ for examples of when the formula may be satisfied and $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ for examples of when the formula may not be satisfied. We shall introduce a basic skeleton structure for each of these reductions, which are then adapted dependent on the exact reduction.

Definition 5.2.4. *The formal definition of the basic skeleton for $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ can be seen below:*

$x : \neg X_\top \wedge \neg X_\bot \to xx_\top;$
$x : \neg X_\top \wedge \neg X_\bot \to xx_\bot;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\top;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\bot;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\top;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\bot;$
$x_\top : X \wedge \neg C_1 \to x_\top c_1;$
$x_\bot : X \wedge \neg C_2 \to x_\bot c_2;$
$y_\top : Y \wedge \neg C_2 \to y_\top c_2;$
$y_\bot : Y \wedge \neg C_1 \to y_\bot c_1;$
$z_\top : Z \wedge \neg C_2 \to z_\top c_2;$
$z_\bot : Z \wedge \neg C_3 \to z_\bot c_3;$
$c_1 : (X \vee Y) \wedge \neg F \to c_1 f;$
$c_2 : (X \vee Y \vee Z) \wedge \neg F \to c_2 f;$
$c_3 : Z \wedge \neg F \to c_3 f;$

Definition 5.2.5. *The formal definition of the basic skeleton for* $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ *can be seen below:*

$x : \neg X_\top \wedge \neg X_\bot \to xx_\top;$
$x : \neg X_\top \wedge \neg X_\bot \to xx_\bot;$
$x_\top : X \wedge \neg C_1 \to X_\top c_1;$
$x_\top : X \wedge \neg C_2 \to X_\top c_2;$
$x_\bot : X \wedge \neg C_3 \to X_\bot c_3;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\top;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\bot;$
$y_\top : Y \wedge \neg C_2 \to y_\top c_2;$
$y_\bot : Y \wedge \neg C_1 \to y_\bot c_1;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\top;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\bot;$
$z_\top : Z \wedge \neg C_4 \to z_\top c_4;$
$z_\bot : Z \wedge \neg C_2 \to z_\bot c_2;$
$c_1 : (X \vee Y) \wedge \neg F \to c_1 f;$

$c_2 : (X \vee Y \vee Z) \wedge \neg F \to c_2 f$;
$c_3 : X \wedge \neg F \to c_3 f$;
$c_4 : Z \wedge \neg F \to c_4 f$;

Example 5.2.6 (An example reduction for termination using 3-SAT instance: $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$)**.**
This example shows how the reduction will work when the formula can be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 3 clause agents (labelled c_1, c_2, c_3) and 1 final agent (labelled f).
This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ with the rule seen below:

$f : C_1 \wedge C_2 \wedge C_3 \to fx$

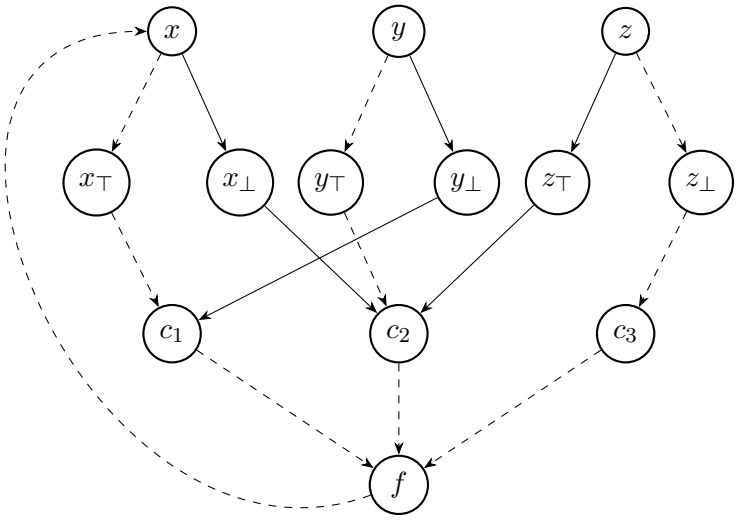

Figure 5.1: Directed communication graph for example 5.2.6. Calls leading to the protocol not terminating have been highlighted by dashed lines, showing that $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ may be satisfied by $(x \wedge y \wedge \neg z)$.

Example 5.2.7 (An example reduction for termination using 3-SAT instance: $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$)**.**
This example shows how the reduction will work when the formula cannot be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 4 clause agents (labelled c_1, c_2, c_3, c_4) and 1 final agent (labelled f).

This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ with the rule seen below:

$f : C_1 \wedge C_2 \wedge C_3 \wedge C_4 \rightarrow fx$

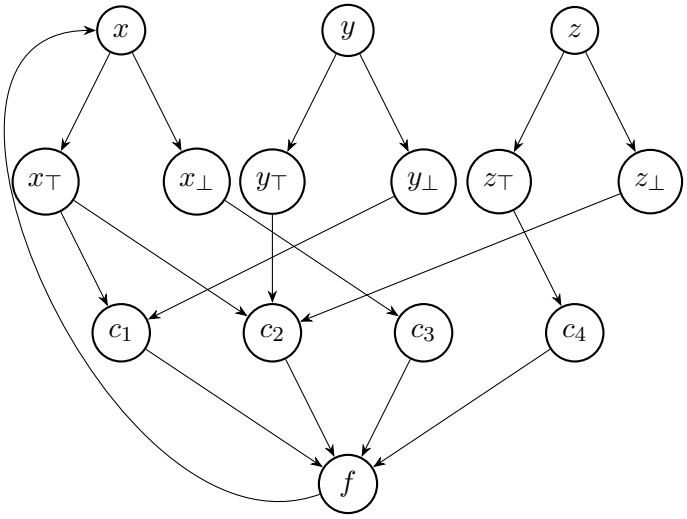

Figure 5.2: Directed communication graph for example 5.2.7. This protocol will always terminate, showing that the formula $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ is not satisfiable.

Using similar techniques we can show the problems of partial correctness and correctness are also coNP-complete.

Theorem 5.2.8. *The problem of checking if a given propositional gossip protocol is partially correct is coNP-complete.*

Proof. First, we show the problem to be in coNP. The protocol is partially correct if for

all leaves $\mathbf{c}$ of the computation tree of the protocol, and all agents a, we have $\mathbf{c} \models Exp_a$,

$$\mathbf{c} \models \bigwedge_{a \in \mathsf{G}, S \in \mathsf{Sec}} F_a S, \tag{5.1}$$

i.e., if each agent is an expert in the gossip situation $\mathbf{c}(\mathsf{root})$. This is true iff every possible protocol that terminates, terminates correctly, i.e. with every agent being an expert. Therefore, if we are told the protocol is not partially correct, and are given the sequence of calls, then the verifier can check in polynomial time if this protocol is possible, verify it terminates, and check there is at least one agent which is not an expert.

To show the problem is coNP-hard we start with the same method as with showing termination is coNP-hard in Theorem 5.2.2, by a reduction from 3-SAT. The set-up of the agents and protocol are exactly the same as in the termination problem, apart from the final agent. An extra agent that we call the repeat agent, denoted by r, is also added. Now, f will call r until he learns the secrets corresponding to all clauses. Unlike the termination case, where once f learns all clauses' secrets he repeatedly makes a call, now f is no longer able to call r once he is familiar with all clauses' secrets. This can be achieved by setting the guard of the fr call to $\neg(C_1 \wedge C_2 ... \wedge C_n)$. It is clear that construction of this protocol can be done in polynomial time and space.

Now, note that this protocol will terminate iff we have a YES instance of 3-SAT, because, otherwise, the guard of the fr call will always be true and hence f will keep on calling r. Once the protocol does terminate, not all agents can be experts: in particular, for any variable x, only one of the agents $x_\top$ and $x_\bot$ will ever be called or make a call, so only one of them can ever become an expert. Therefore, if this protocol terminates then it necessarily terminates in an incorrect gossip situation and so is not partially correct. On the other hand, if this protocol can never terminate then it is partially correct by default, because partial correctness only requires finite computations to reach the all-expert gossip situation. □

Example 5.2.9 (An example reduction for partial correctness using 3-SAT instance: $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$)**.**
This example shows how the reduction will work when the formula can be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 3 clause agents (labelled c_1, c_2, c_3) and 1 final agent (labelled f), as well as 1 repeat agent (labelled r).
This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ with the rule seen below:

$f : \neg(C_1 \wedge C_2 \wedge C_3) \rightarrow fr$

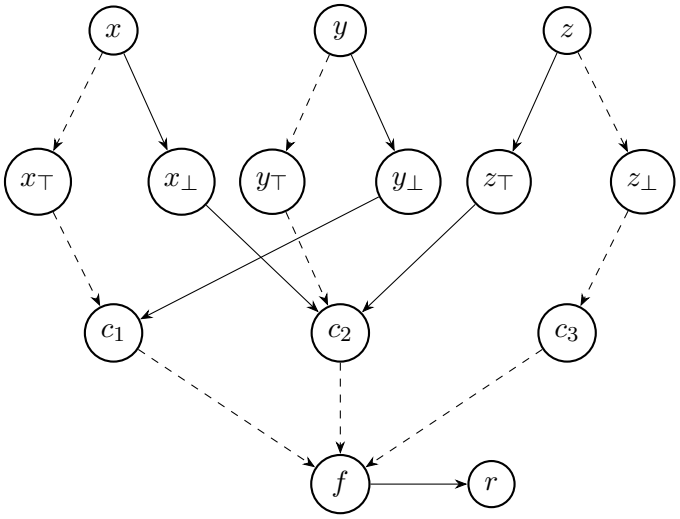

Figure 5.3: Directed communication graph for example 5.2.9. Calls leading to the protocol terminating incorrectly have been highlighted by dashed lines, showing that the protocol is not partially correct, and that $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ may be satisfied by $(x \wedge y \wedge \neg z)$.

Example 5.2.10 (An example reduction for partial correctness using 3-SAT instance: $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$)**.**
This example shows how the reduction will work when the formula cannot be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 4 clause agents (labelled c_1, c_2, c_3, c_4) and 1 final agent (labelled f), as well as 1 repeat agent (labelled r).
This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ with the rule seen below:

$f : \neg(C_1 \wedge C_2 \wedge C_3 \wedge C_4) \to fr$

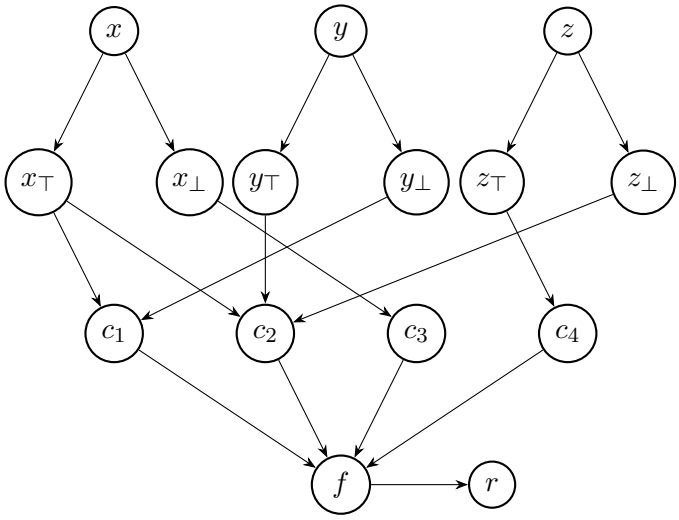

Figure 5.4: Directed communication graph for example 5.2.10. No computation of this protocol will terminate, and therefore the protocol is partially correct, showing that the formula $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ is not satisfiable.

Theorem 5.2.11. *The problem of checking if a given propositional gossip protocol is correct is coNP-complete.*

Proof. First we show the problem to be in coNP. The protocol is correct if it is partially correct and it terminates. This means every computation terminates correctly, i.e. with every agent being an expert. Therefore, the verifier can check in polynomial time if a protocol is not correct by guessing a polynomially long prefix of computation after which

a call can be repeated indefinitely (which leads to an infinite computation) or no call is possible but not all agents are experts, the same way it was done in Theorem 5.2.2 and Theorem 5.2.8, respectively.

To show the problem is coNP-hard we start with the same method as with showing termination was coNP-complete in Theorem 5.2.2, by a reduction from 3-SAT. However, there are some small differences. We will again have the special agent, which we call the final agent and denote by f, which will repeat a call once it is familiar with a certain set of secrets. However, additionally, there is a new agent l and every agent at every stage would wish to initiate a call with l, unless he is familiar with L already.

Any agent i which is familiar with L reverts to the Learn New Secrets protocol (see Example 2.1.3), i.e.

$$*[[]_{j\in\mathsf{G}}(F_iL \wedge \neg F_iJ) \to ij].$$

Note that any of these calls can be used at most once, because i will learn J after the ij call is made.

For each Boolean variable, x, used in the 3-SAT formula, we will create three agents: variable agent (x), true agent ($x_\top$), and false agent ($x_\bot$). For every $i \leq m$, where m is the number of clauses in the 3-SAT formula, we have agent c_i that corresponds to the i-th clause.

Now we define the rules of the protocol for each of the agents. For each variable agent, x, while this agent is only familiar with its own secret X, x may initiate a call with agent $x_\top$ or agent $x_\bot$ with the guard $(\neg X_\top \wedge \neg X_\bot)$ and the scheduler will decide which of these calls takes place. This essentially sets the value of the x variable in the 3-SAT formula to true or false, respectively.

Now any true agent, $x_\top$, and false agent, $x_\bot$, that was called by his corresponding variable agent can make a call to any clause agent, c, whose corresponding clause in the 3-SAT formula uses the literal x or $\neg x$, respectively. This is done with the guard $(X \wedge \neg C)$, to ensure this call cannot be repeated.

Once a clause agent receives a call and does not learn L, this means that the corresponding clause is satisfied by the current truth assignment. Once this happens, the clause agent calls f, with a guard $(X \wedge \neg F)$, to make a call to f that cannot be repeated.

Now, f will learn the secrets of all clause agents but not L only if all clauses have been satisfied. Apart from the option of calling l and reverting to Learn New Secrets, f only makes a call if it learns all of these clause secrets. Agent f can make a call to a fixed

variable agent x with guard $(C_1 \wedge ...C_n \wedge \neg L)$, which allows f to make infinitely many calls to x until he learns L. There is the possibility that x may have already learned L at this stage, but having a single computation which does not terminate is enough for the protocol to be incorrect, and there is certainly no requirement for x to be familiar with L in this case.

The only agent which may repeat a call, and therefore lead to the protocol not terminating is f, and this may only happen if all clauses are satisfied. If not all clauses are satisfied, the only calls remaining will be to l, or by agents that are familiar with L. If this occurs, then clearly the protocol will then continue until all agents are experts and then terminate correctly. Hence, the protocol is correct if and only if we have a NO instance of 3-SAT. It is easy to see that this whole construction can be done in polynomial time and space. □

Example 5.2.12 (An example reduction for correctness using 3-SAT instance: $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$)**.**
This example shows how the reduction will work when the formula can be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 3 clause agents (labelled c_1, c_2, c_3) and 1 final agent (labelled f), as well as 1 learn new secrets agent (labelled l).
This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ with the rules seen below:

$f : (C_1 \wedge C_2 \wedge C_3 \wedge \neg L) \to fx$
$i \in \mathsf{G} : \neg L \to il$
$i \in \mathsf{G} : L \wedge \neg J \to ij, j \in \mathsf{G}$
Not every computation of this protocol will terminate correctly, and therefore the protocol is correct, showing that the formula $(x \vee \neg y) \wedge (\neg x \vee y \vee z) \wedge (\neg z)$ may be satisfied by $(x \wedge y \wedge \neg z)$.

Example 5.2.13 (An example reduction for correctness using 3-SAT instance:
$(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$)**.**
This example shows how the reduction will work when the formula cannot be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 4 clause agents (labelled c_1, c_2, c_3, c_4) and 1 final agent (labelled f), as well as 1 learn new secrets agent (labelled l).
This protocol combines the basic skeleton for $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ with the rules seen below:
$f : (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge \neg L) \to fx$
$i \in \mathsf{G} : \neg L \to il$
$i \in \mathsf{G} : L \wedge \neg J \to ij, j \in \mathsf{G}$

Every computation of this protocol will terminate correctly, and therefore the protocol is correct, showing that the formula $(x \vee \neg y) \wedge (\ x \vee y \vee \neg z) \wedge \neg x \wedge z$ is not satisfiable.

5.3 Conclusion

In this chapter we have focused on the complexity of problems relating to correct propositional gossip protocols, showing that the problems of termination, partial correctness and correctness are all coNP-complete. In doing so we have also found a useful condition on termination which further improves our understanding of these protocols.

Chapter 6

Fair and Correct Communication Graphs

6.1 Introduction

Having investigated propositional gossip protocols under a demonic scheduler, we now evolve to look how changing this scheduler to include fairness may affect the correctness of the protocol. Recall from Chapter 2: an infinite computation is ***rule-fair*** (resp. ***agent-fair***) if all rules (resp. agents) that are active (the corresponding guards are true) after infinitely many prefixes of this computation are selected infinitely often. By definition a finite computation is rule-fair and agent-fair, as no computation may continue infinitely. We say that a gossip protocol P ***rule-fairly terminates*** (resp. ***agent-fairly terminates***) if all its rule-fair (resp. agent-fair) computations are finite. We do not define agent-fair nor rule-fair partial correctness, because they are equivalent to partial correctness. This is because partial correctness is a condition on finite computations only and every finite computation is agent-fair and rule-fair. A protocol which rule-fairly (agent-fairly) terminates and is partially correct is said to be ***correct under rule-fairness (agent-fairness)***. Note that, straight from the definition, any protocol which is correct under agent-fairness is also correct under rule-fairness, because rule-fairness can only narrow down further the set of permitted computations. A natural next step is to explore how the solutions given in Chapters 3 and 5 change when fairness is introduced. Previously, if a call could be repeated indefinitely, then the scheduler would take this option and lead to non-termination and therefore failure of the protocol, however this is not always natural.

When we again imagine the problem as three work colleagues working together as in Chapter 1, one of these colleagues would not simply repeat a call indefinitely if they could also call their other colleague. This may not always lead to all colleagues learning all the information, but surely would be worth trying! By introducing fairness, we stop these situations by ensuring other possible calls may take place.

Clearly, introducing fairness will not decrease the number of calls required in the worst case for a correct protocol, however investigating how fairness affects termination is of interest.

We may observe that even simple results from Chapter 3 no longer hold. As we will see shortly, when fairness is introduced correct protocols exist when the undirected communication graph is a star, which was shown not to be the case in Theorem 3.2.4 for the case without fairness. Results regarding termination found in Lemma 3.2.3 also no longer hold. An agent calling another agent while being familiar with that agent's secret may now still lead to a correct protocol.

6.2 Fair and Correct Communication Graphs

We begin by showing that the scheduler's type has a huge impact on whether a gossip protocol works correctly. In particular we show that rule-fairness and agent-fairness are not equivalent.

Proposition 6.2.1. *There exists a propositional gossip protocol which is correct under agent-fairness but is not a correct propositional gossip protocol.*

Proof. Let us have three agents, a, b and c. The protocol only consists of a rule $\neg F_a C \to ab$ for agent a, and $F_b A \wedge \neg F_b C \to bc$ for agent b. Clearly this is not a correct propositional gossip protocol as it may never terminate, e.g., ab call can be repeated indefinitely. However, this propositional gossip protocol is correct under agent-fairness. Note that the first call has to be ab. After that call, both ab and bc calls are active. Due to the agent-fairness of the scheduler used, eventually the bc call must take place, ensuring that both b and c become experts. After that, the only active call is ab, and once this occurs no calls are active and so the protocol terminates. Moreover, all agents are experts at that point. □

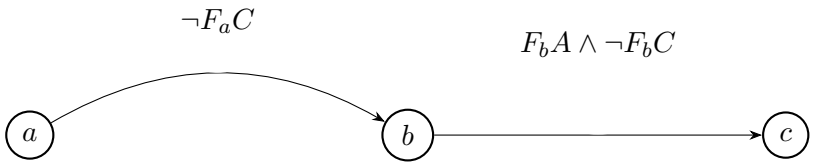

Figure 6.1: Directed communication graph for the protocol proposed in Proposition 6.2.1. Each guard has been included on the diagram, above the edge representing the corresponding call.

As mentioned, any protocol which is correct under agent-fairness is also correct under rule-fairness. We now show that the opposite does not hold.

Proposition 6.2.2. *There exists a propositional gossip protocol which is correct under rule-fairness but is not correct under agent-fairness.*

Proof. Let us have three agents, a, b and c. The protocol only consists of a rule $\neg F_a C \to ab$ for agent a, and rules $F_b A \wedge \neg F_b C \to bc$, $F_b A \wedge \neg F_b C \to ba$ for agent b. This is not a propositional gossip protocol which is correct under agent-fairness. Its computation has to start with ab, at which point calls ab, ba and bc are active. Note that a computation repeatedly alternating between making calls ab and ba after that is agent-fair, because a call of each active agent (a and b only) is picked infinitely many times. The protocol will not terminate in this case, so it is not correct under agent-fairness. However, this propositional gossip protocol is correct under rule-fairness, because under a rule-fair scheduler the active call bc has to be picked eventually. Once bc is made both b and c will become experts. This leaves ab as the only active call, and once this occurs, the protocol terminates in a gossip situation where all agents are experts. □

With this knowledge that the type of the scheduler used can change if a given protocol is correct, we revisit the Problem 5 of [9] to determine what the communication graph should be for there to exist a propositional gossip protocol which is correct under agent-fairness or rule-fairness. We start with the simplest case of a path (linear) graph where the n agents are labeled $x_1, \ldots, x_n$ and, for any $i < n$, agent x_i can only call x_{i+1}. We then proceed with increasingly broader graph classes, which, on the downside, require increasingly more complex protocols to be correct under agent-fairness or under rule-fairness.

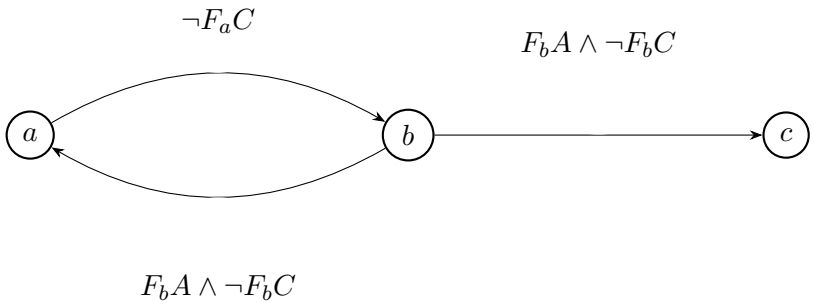

Figure 6.2: Directed communication graph for the protocol proposed in Proposition 6.2.2. Each guard has been included on the diagram, on the edge representing the corresponding call.

Theorem 6.2.3. *For any path graph G (a connected graph with a sequence of edges joining a sequence of distinct vertices), there exists a propositional gossip protocol which is correct under agent-fairness with G as its communication graph.*

Proof. The protocol consists of rules $\neg Exp_{x_1} \to x_1x_2$ for agent x_1, $F_{x_2}X_1 \wedge \neg Exp_{x_2} \to x_2x_3$ for agent x_2, ..., $F_{x_{n-1}}X_{n-2} \wedge \neg Exp_{x_{n-1}} \to x_{n-1}x_n$ for agent x_{n-1}. In other words, an agents wants to call the next agent if he is familiar with the secret of the previous agent and he is not yet an expert.

Note that initially only agent x_1 is active and he will call x_2, after which both of these agents become active. Due to agent-fairness, agent x_2 will eventually make his call after which agent x_3 becomes active and so on. Note that, for every $i < n$, when the call $x_{i-1}x_i$ becomes active, x_{i-1} is already familiar with all secrets $X_1, \ldots, X_{i-1}$. This means that when eventually the call $x_{n-1}x_n$ takes place, both of the agents involved become the first two experts. Then eventually, the call $x_{n-2}x_{n-1}$ will take place due to agent-fairness, and x_{n-2} will become the third expert and so on until x_1 becomes an expert when the protocol terminates. □

As pointed out earlier, Theorem 6.2.3 implies that the same result holds for rule-fairness. We can also get that every graph with a Hamiltonian path can be a communication graph of a protocol which is correct under agent-fairness / correct under rule-fairness by ignoring the rest of the edges and constructing the protocol as in Theorem 6.2.3. But

instead we are going to generalize this construction to show this result for any tree, which is any connected acyclic undirected graph. The gossip protocol will be set up in a way to ensure there is a unique directed path from every agent to a special agent $\bar{x}$, who is guaranteed to be the first agent to become an expert. Then all the secrets collected in $\bar{x}$ will eventually filter back up the directed paths to all the other agents.

Theorem 6.2.4. *For any tree T, there exists a propositional gossip protocol which is correct under agent-fairness with T as its undirected communication graph.*

Proof. Let us pick any agent $\bar{x}$ in this tree T. We now turn this undirected tree into a directed one by setting the direction of all edges in this tree towards $\bar{x}$. First, the distance of an agent x from $\bar{x}$ is the length of the shortest path from x to $\bar{x}$ traversing just edges of T. Now, formally, the direction of an edge connecting two agents is from the agent at a larger distance from $\bar{x}$ towards the agent at a smaller distance to $\bar{x}$. This is well-defined, because we now show that no two neighboring agents in T can have the same distance to $\bar{x}$. If they had then these two paths of equal length, from each of them to $\bar{x}$, cannot completely overlap. So these paths would intersect at $\bar{x}$ or another agent. The two parts of these paths until the intersection together with the edge connecting these two agents would form a cycle, which is not possible as T was supposed to be acyclic. Similarly, we can argue that each node apart from $\bar{x}$ in such formed tree has a unique outgoing edge ($\bar{x}$ has none). This is because if an agent x has two successors in such a directed tree then there would be a path from each of them to $\bar{x}$ that intersect at some point. These paths to the intersection point together with the edges from x would form a cycle; a contradiction.

Let $pred(x)$ denote the set of all agents who have an outgoing edge to x in such a defined directed tree and $succ(x)$ denote the unique agent who has an incoming edge from x.

Now, the protocol will only consist of a single rule for each agent $x \in \mathsf{G} \setminus \{\bar{x}\}$: $\neg Exp_x \wedge \bigwedge_{y \in pred(x)} F_x Y \to x \circ succ(x)$. In other words, an agent wants to call its unique direct successor agent if he is familiar with all the secrets of his direct predecessors and he is not yet an expert. Agent $\bar{x}$ has no rules.

The rest of the proof proceeds similarly to Theorem 6.2.3. First, note that when the call $x \circ succ(x)$ becomes active then x has to already be familiar with all secrets of all his (indirect) predecessors in this directed tree. This is because it can only become active after all direct predecessors of x call him and by inductive assumption all of them have to be familiar with all secrets of their (indirect) predecessors at that point. Second, due

to agent-fairness, every call that is active will eventually be made, because there is only at most one such call per agent. Initially, only agents that do not have predecessors are active and all of them will make their calls. A call does not stop being active until the agent making it becomes an expert. Due to the protocol setup this cannot happen before $\bar{x}$ becomes an expert, but that only takes place once all the calls that can be made were made (otherwise some secrets would not reach $\bar{x}$).

Note that once $\bar{x}$ becomes an expert (and the agent calling him), the number of active agents is $n - 2$. Now, only when a new agent becomes an expert (by calling an agent who is already an expert), he stops being active and will never become active again. This shows that the protocol is partially correct. Moreover, as long as there is at least one active call, one of them will have an expert as its callee. This guarantees the protocol to agent-fairly terminate, because eventually such a call is made and the number of active agents decreases. Together this shows the protocol to be correct under agent-fairness. □

We are now ready to prove the result for arbitrary undirected graphs. This result is a corollary of Theorem 6.2.4.

Corollary 6.2.5. *For any undirected connected graph G, there exists a propositional gossip protocol which is correct under both agent-fairness and rule-fairness with G as its undirected communication graph.*

Proof. Any such graph G has a spanning tree T, which we pick arbitrarily. We use the same protocol as defined in Theorem 6.2.4 when applied to T, as a protocol which is correct under agent-fairness, is also correct under rule-fairness. □

It is trivial to see that for any undirected graph which is not connected there cannot be a correct protocol with this graph as its communication graph, because no agent can ever become an expert as secrets cannot be shared across different components. This together with Corollary 6.2.5 resolves the case of undirected communication graphs, but naturally leads on to the question of what happens in the directed case.

Theorem 6.2.6. *For any strongly connected directed graph D, there exists a propositional gossip protocol which is correct under rule-fairness with D as its directed communication graph.*

Proof. The protocol that we are going to use is: $\neg Exp_x \to xy$ for all agents x and agents y which are direct successors of x in D. In other words, an agent keeps on calling every

agent he can until he is an expert. Clearly, this protocol can only terminate once every agent is an expert, so it is partially correct.

Now suppose this protocol does not rule-fairly terminate and so there exists a rule-fair computation such that agent a does not learn secret B. As the scheduler is rule-fair, every call which is active will eventually be made. As a is not an expert, and there is a path from a to b, a must be trying to make a call to an agent whose distance to b in D is smaller than that of a. If this agent, say c, is already familiar with B then a would get to be familiar with B eventually due to rule-fairness. If c is not yet familiar with B, then c must be trying to make a call to an agent who is closer to b, and so on. As the number of calls in this path that ends at b or at another agent already being familiar with B is finite, and all these calls are active, eventually all agents along this path will learn B; a contradiction. □

Now, we are going to generalize even further the result of Corollary 6.2.5, by showing a construction of a protocol which is correct under agent-fairness for an arbitrary strongly connected directed graph as its directed communication graph (note that any undirected connected graph can be viewed as a strongly connected directed graph).

Theorem 6.2.7. *For any strongly connected directed graph D, there exists a propositional gossip protocol which is correct under agent-fairness with D as its directed communication graph.*

Proof. Let us number all the agents $x_1, \ldots, x_n$. Intuitively, the constructed protocol will proceed in stages, but agents can progress through stages at a different pace. At stage i an agent tries to learn the secret X_i. Once he achieves that he moves on to stage $i+1$ until eventually he reaches stage $n+1$ when he becomes an expert and stops being active. Note that an agent can skip a stage if he is already familiar with the desired secret. It is clear the guard of the rule for an agent a at stage i can be written down as a propositional formula: $\bigwedge_{j<i} F_a X_j \wedge \neg F_a X_i$.

At the beginning of the protocol, all agents start at stage 1, apart from agent x_1 which can already move on to stage 2. Every agent at stage i tries to make just one call: to any of his direct successors in D whose distance to x_i is the shortest possible in D (breaking ties arbitrarily). Note that this is well-defined, because D is strongly connected.

We now show that at any point of the computation an agent, a, who is at the smallest stage number, i, can reach an agent that is familiar with X_i by following a path created

by the calls active at that time. The proof is by induction on the distance of a to x_i. If this distance is 1 then it is obviously true because he can call x_i directly. Otherwise, a either wants to call an agent still at stage i, which by the inductive assumption can reach an agent which is familiar with X_i, or an agent at a higher stage, in which case that agent is already familiar with X_i.

As the scheduler is agent-fair and each agent just tries to make a single call, along the just proven finite path to X_i, all agents will eventually succeed to learn X_i and move to stage $i+1$. And this process will continue until all agents are experts. □

Note, however, it is not necessary for the communication graph to be strongly connected. Consider three agents a, b and c, and a directed graph with edges ab and cb only; it is not strongly connected, as there is no directed path between a and c. And yet, a propositional protocol which is correct under agent-fairness exists with rules: $\neg Exp_a \to ab$ for agent a and $\neg Exp_c \to cb$ for agent c. This protocol repeats these two calls until the agent making them is an expert. On the other hand, we cannot even broaden the class of directed graphs to weakly connected ones (i.e., connected when the edges' directions are ignored).

Proposition 6.2.8. *There exists a weakly connected directed graph D, such that no propositional gossip protocol which is correct under either rule-fairness or agent-fairness exists with D as its directed communication graph.*

Proof. We prove this with a counterexample. Consider three agents a, b and c, and a weakly connected directed graph with edges ba and bc only. W.l.o.g., the computation starts with the call ba followed by bc at a later point. But at that point, both b and c are experts, and hence cannot initiate a call, because otherwise such a call could be repeated forever and the protocol would not terminate. But this means that no further calls may be made, and yet a is still not an expert. □

6.3 Conclusion

In this chapter we have explored the effects of adding fairness to propositional gossip protocols. We found that changing the scheduler has great impact on these protocols, with rule-fairness also having different results to agent-fairness. We continued further, to show that for any connected undirected graph, there exists a propositional gossip protocol which

is correct agent-fairness and under rule-fairness with this graph as its communication graph in Corollary 6.2.5. This is in stark contrast with the general case, where the undirected communication graph had to be complete for a correct propositional gossip protocol to exist as shown in Theorem 3.2.9 in Chapter 3. Further, we investigated the question for directed graphs, finding that for any strongly connected directed graph there exists a propositional gossip protocol which is correct under rule-fairness with this graph as its communication graph in Theorem 6.2.6, and extending this to the agent-fair case in Theorem 6.2.7. Finally, in Proposition 6.2.8, we found that a directed graph being weakly connected is not enough to guarantee the existence of even a propositional gossip protocol which is correct under rule-fairness with this graph as its communication graph.

Chapter 7

The Complexity of Propositional Gossip Protocol Problems Under Fair Schedulers

7.1 Introduction

We now look at the computational complexity of checking if a given propositional gossip protocol terminates and if it is correct under agent-fair or rule-fair schedulers. With no fairness restrictions on the scheduler used, these problems were shown to be coNP-complete in Chapter 5. We show exactly the same computational complexity but with modified proofs to account for the fairness constraints.

7.2 Complexity of Checking Fair Correctness

Theorem 7.2.1. *Checking if a given propositional gossip protocol agent-fairly or rule-fairly terminates is coNP-complete.*

Proof. Due to monotonicity of the gossip situations along any computation (as agents never forget secrets), the gossip situation can change at most n^2 times along any computation. This is because each of the n agents can be familiar with at most n different secrets, and with each change at least one more secret is learned. Skipping a call that does not change the current gossip situation will give us another valid computation. This is because

the truth value of all propositional formulas used as guards would stay the same. The protocol does not agent-fairly (rule-fairly) terminate iff there is an infinite agent-fair (resp. rule-fair) computation. In such a computation the gossip situation stabilizes after at most n^2 changes. Then under agent-fairness, each active agent can then make at least one call that does not change the gossip situation and under rule-fairness all active calls are like that (with at least one agent being active). It is easy to see that we can now guess this polynomial-size prefix of this infinite computation and which calls should be repeated forever, and verify this guess in polynomial time. This shows that non-termination is in NP and so termination is in coNP.

We show the problem is coNP-hard with the same reduction as in Theorem 5.2.2. □

Theorem 7.2.2. *Checking if a given propositional gossip protocol is correct under agent-fairness or rule-fairness is coNP-complete.*

Proof. We show this problem to be in coNP similarly as in Theorem 7.2.1 by guessing and checking a polynomial sized finite computation where not all agents end up as experts or a polynomial sized prefix of an infinite computation.

As for coNP-hardness, the set-up is as in Theorem 5.2.2 and Theorem 7.2.1 with some minor changes. There are three additional agents v, w and l. If an agent $a \in \mathsf{G} \setminus \{v, w\}$ is not familiar with L then a can call l. If an agent $a \in \mathsf{G} \setminus \{v, w\}$ at any point learns L then a starts to follow the LNS protocol (see Example 2.1.3) for the set of agents $\mathsf{G} \setminus \{v, w\}$. If an agent $a \in \mathsf{G} \setminus \{v, w\}$ is familiar with all secrets apart from V and W, he can call v, and if a is familiar with all secrets apart from W then he can call W.

If v is familiar with all clause secrets as well as F, but not W, v can call w, and once v is also familiar with W, he can call f until he becomes an expert. Additionally, if f is familiar with all clause secrets but not L, V nor W, he can call v. It is clear that this whole construction can be done in polynomial time and size and all such defined guards are propositional formulas.

Assume that the formula is not satisfiable. Then f cannot learn all clause secrets without learning L, and hence will eventually call (or be called by) an agent which is familiar with L, and hence revert to LNS that excludes v and w. Therefore, v can only be called by an agent which is familiar with all secrets apart from V and W. At this stage, agents may begin to call w if they are familiar with all secrets apart from W, thus making both agents experts. Every agent which is familiar with W is an expert, and so any agent

which calls such an agent will also be an expert. We can see that at termination every agent will be an expert.

Assume that the formula is satisfiable. Thus it is possible to be in a situation where f is familiar with all clause secrets but not L, V nor W, so can call v. After this call, v may call w, and then repeatedly call f, until v becomes an expert (which he is not yet as he is not familiar with L). Now, after the first instance of the vf call, every agent which is familiar with V is also familiar with W. This means that an agent is either familiar with both V and W, or neither V nor W. As w can never make a call, this means w must be called by an agent which is familiar with L in order to learn this secret. Yet, for an agent to call w, he must first learn V. Now, every agent which is familiar with V is also familiar with W, and so no agent that is familiar with L will ever call w, hence w will never become an expert. □

Example 7.2.3 (An example reduction for fair correctness using 3-SAT instance: $(X \vee \neg Y) \wedge (\neg X \vee Y \vee Z) \wedge \neg Z$)**.**
This example shows how the reduction will work. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 3 clause agents (labelled c_1, c_2, c_3) and 1 final agent (labelled f), as well as 1 learn new secrets agent (labelled l) and 2 further additional agents (labelled v, w).
The formal definition of this protocol can be seen below.

$x : \neg X_\top \wedge \neg X_\bot \to xx_\top;$
$x : \neg X_\top \wedge \neg X_\bot \to xx_\bot;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\top;$
$y : \neg Y_\top \wedge \neg Y_\bot \to yy_\bot;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\top;$
$z : \neg Z_\top \wedge \neg Z_\bot \to zz_\bot;$
$x_\top : X \wedge \neg C_1 \to x_\top c_1;$
$x_\bot : X \wedge \neg C_2 \to x_\bot c_2;$
$y_\top : Y \wedge \neg C_2 \to y_\top c_2;$
$y_\bot : Y \wedge \neg C_1 \to y_\bot c_1;$
$z_\top : Z \wedge \neg C_2 \to z_\top c_2;$
$z_\bot : Z \wedge \neg C_3 \to z_\bot c_3;$
$c_1 : (X \vee Y) \wedge \neg F \to c_1 f;$

$c_2 : (X \vee Y \vee Z) \wedge \neg F \to c_2 f$;
$c_3 : Z \wedge \neg F \to c_3 f$;
$f : (C_1 \wedge C_2 \wedge C_3 \wedge \neg L \wedge \neg V \wedge \neg W) \to fv$
$v : (C_1 \wedge C_2 \wedge C_3 \wedge F \wedge \neg W) \to vw$
$v : (W \wedge \neg Exp_v) \to vf$
$i \in \mathsf{G} \setminus \{v, w\} : \neg L \to il$
$i \in \mathsf{G} \setminus \{v, w\} : \mathsf{Sec} \setminus \{V, W\} \to iv$
$i \in \mathsf{G} \setminus \{v, w\} : \mathsf{Sec} \setminus \{W\} \to iw$
$i \in \mathsf{G} \setminus \{v, w\} : L \wedge \neg J \to ij, j \in \mathsf{G} \setminus \{v, w\}$

Example 7.2.4 (An example reduction for fair correctness using 3-SAT instance: $(X \vee \neg Y) \wedge (\ X \vee Y \vee \neg Z) \wedge \neg X \wedge Z$)**.**
This example shows how the reduction will work when the formula cannot be satisfied. This will require 3 variable agents (labelled x, y, z), 3 true agents (labelled $x_\top, y_\top, z_\top$), 3 false agents (labelled $x_\bot, y_\bot, z_\bot$), 4 clause agents (labelled c_1, c_2, c_3, c_4) and 1 final agent (labelled f), as well as 1 learn new secrets agent (labelled l) and 2 further additional agents (labelled v, w).
The formal definition of this protocol can be seen below.

$x : \neg X_\top \wedge \neg X_\bot \to x x_\top$;
$x : \neg X_\top \wedge \neg X_\bot \to x x_\bot$;
$x_\top : X \wedge \neg C_1 \to X_\top c_1$;
$x_\top : X \wedge \neg C_2 \to X_\top c_2$;
$x_\bot : X \wedge \neg C_3 \to X_\bot c_3$;
$y : \neg Y_\top \wedge \neg Y_\bot \to y y_\top$;
$y : \neg Y_\top \wedge \neg Y_\bot \to y y_\bot$;
$y_\top : Y \wedge \neg C_2 \to y_\top c_2$;
$y_\bot : Y \wedge \neg C_1 \to y_\bot c_1$;
$z : \neg Z_\top \wedge \neg Z_\bot \to z z_\top$;
$z : \neg Z_\top \wedge \neg Z_\bot \to z z_\bot$;
$z_\top : Z \wedge \neg C_4 \to z_\top c_4$;
$z_\bot : Z \wedge \neg C_2 \to z_\bot c_2$;
$c_1 : (X \vee Y) \wedge \neg F \to c_1 f$;
$c_2 : (X \vee Y \vee Z) \wedge \neg F \to c_2 f$;

$c_3 : X \wedge \neg F \rightarrow c_3 f$;
$c_4 : Z \wedge \neg F \rightarrow c_4 f$;
$f : (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge \neg L \wedge \neg V \wedge \neg W) \rightarrow fv$
$v : (C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge F \wedge \neg W) \rightarrow vw$
$v : (W \wedge \neg Exp_v) \rightarrow vf$
$i \in \mathsf{G} \setminus \{v, w\} : \neg L \rightarrow il$
$i \in \mathsf{G} \setminus \{v, w\} : \mathsf{Sec} \setminus \{V, W\} \rightarrow iv$
$i \in \mathsf{G} \setminus \{v, w\} : \mathsf{Sec} \setminus \{W\} \rightarrow iw$
$i \in \mathsf{G} \setminus \{v, w\} : L \wedge \neg J \rightarrow ij, j \in \mathsf{G} \setminus \{v, w\}$

7.3 Conclusion

In this chapter we have determined the complexity of the propositional gossip protocol problem under fair schedulers. These problems were found to be coNP-complete, as in Chapter 7 where these problems were found to be coNP-complete when not considering fairness. Although there are many differences brought about by changing the scheduler used, there still remains many similarities, with the complexity being one of these. As seen, the proofs also remain similar to the cases where fairness is not used in Chapter 7, and in some cases are the same, yet their differences need to be noted, especially when dealing with correctness.

Chapter 8

Simulation

8.1 Introduction

We move on to investigating the simulation problem, in particular, its complexity. Given a tree T, any removal of branches from T yields a ***subtree*** of T. Consider two gossip protocols P and P', we say that P ***can simulate*** P' if the computational tree of P' is equal to some subtree of P. Furthermore, if both P can simulate P' and P' can simulate P then we say they P and P' are ***bisimilar***. Note that these definitions are different to those given for programs in [28].

Example 8.1.1 (An example of simulation)**.** *The protocol defined by the calls:*
$x : (X \wedge \neg Y) \rightarrow xy$;
$y : (X \wedge \neg Z) \rightarrow yz$;
can simulate the protocol defined by the call:
$x : (X \wedge \neg Y) \rightarrow xy$;

The problem of simulation is an interesting and useful one. Given two protocols is it possible for one to simulate the other? Having the knowledge that a protocol P can simulate a protocol P' gives us the ability potentially to extract useful information without having to carry out as much work. For example if we know that P terminates, then we also know that P' terminates. This is also true of rule/agent fair termination. Furthermore, we have useful information in terms of potentially having a correct protocol with fewer calls required. For example, if we know P' to be a correct protocol, then P cannot be correct whilst requiring fewer calls.

Observe that if LNS simulates a propositional protocol P, then P must terminate. This is because no computation of LNS will involve a call being made without the caller learning new information, the proof then follows from Lemma 5.2.1. Conversely, if LNS cannot simulate a protocol, then there is a computation of P where a call is made to an agent, where that agent knew the secret of the agent it is calling. With some schedulers this may lead to a correct computation, but if we have a demonic scheduler without any fairness constraints, using Lemma 3.2.3 it is known that this will lead to a path which does not terminate.

If a protocol P' simulates LNS, this does not give us information on whether P' is correct, or even if it does or does not terminate. However, is does give us the information that should P' be correct, it cannot be correct on fewer than $n^2 - n$ calls, as this is the number of calls required for LNS to be guaranteed to terminate correctly.

Example 8.1.2 (An example of simulation - comparing LNS to LXS on three agents)**.** *We shall label these three agents a, b, c and index them such that a has the lowest index and c has the largest.*
Let us recall that for the LXS (Learn Next Secret) protocol the following program is used by agent i:

$$*[[]_{\{j \in \mathsf{G} | j > i\}} \neg F_i J \wedge \bigwedge_{\{k \in \mathsf{G} | k < j\}} F_i K \rightarrow ij]$$

Hence, the only two computations of this protocol are:
ab.ac.bc ; ab.bc.ac.

In comparison LNS has 24 computations: ; ab.ac.bc ; ab.ca.bc ; ab.bc.ac ; ab.cb.ac ; ba.ac.bc ; ba.ca.bc ; ba.bc.ac ; ba.cb.ac ; ac.ab.cb ; ac.ba.cb ; ac.bc.ab ; ac.cb.ab ; ca.ab.cb ; ca.ba.cb ; ca.bc.ab ; ca.cb.ab ; bc.ab.ca ; bc.ba.ca ; bc.ac.ba ; bc.ca.ba ; cb.ab.ca ; cb.ba.ca ; cb.ac.ba ; cb.ca.ba.

Observe that LNS can simulate LXS, this in itself shows that LXS must terminate. As LXS cannot simulate LNS, they are not bisimilar.

Note that simulation does not require protocols to lead to an all expert situation, or even terminate.

Example 8.1.3 (An example of simulation without termination)**.** *The protocol defined by the calls:*
$x : X \rightarrow xy$;
$y : X \rightarrow yz$;

can simulate the protocol defined by the call:
$x : X \to xy$;

In Example 8.1.3 we observe that there are indeed no requirements on termination. This fortifies the fact that simulation can be a powerful tool if one of the protocols in question has already been studied. Further, the scheduler also has no effect on simulation, due to the definition being on computation trees, and hence all possible call sequences. This therefore leads us on to looking at the complexity of the problem.

8.2 Complexity of the Simulation Problem for Propositional Gossip Protocols

We shall begin by looking at the complexity of the simulation problem for propositional gossip protocols. As mentioned, having the knowledge that a protocol P can simulate a protocol P' can provide us with useful information regarding the protocols without having to do any additional analysis on the protocol itself. Knowing how efficiently we can solve this problem will give us insight into whether this is a feasible option in general.

Theorem 8.2.1. *LNS can simulate a protocol with the same number of agents iff the protocol terminates.*

Proof. ($\Leftarrow$) Consider a terminating propositional gossip protocol P. Initially every call is available so LNS can simulate this first call. Let us assume LNS can simulate the prefix of a computation of P, $\mathbf{c}$, and let us consider an extension of this $\mathbf{c}.ab$, such that ab is the first call in the computation which cannot be simulated by LNS. By defintion of the LNS protocol, this call must be made where $\mathbf{c}.ab \models F_a B$, else this call would also be available to LNS. Yet, Lemma 3.2.3 shows that for a propositional gossip protocol to terminate, there must be no call such that an agent calls an agent whose secret it is already familiar with, a contradiction.

($\Rightarrow$) As the LNS protocol always terminates, it may not simulate any protocol which does not terminate. The longest computation of LNS is no more than n^2 calls, $(n^2 - n)$ calls in fact, and hence LNS cannot simulate a protocol P which does not terminate, and hence must have an infinite computation. □

Corollary 8.2.2. *The simulation problem for propositional gossip protocols is coNP-hard.*

Proof. We achieve this result with a polynomial time reduction to the termination problem of propositional gossip protocols, which is shown to be coNP-complete in 5.2.2, using LNS due to the fact that LNS can simulate a protocol iff the protocol terminates, as shown in Theorem 8.2.1. □

Theorem 8.2.3. *The simulation problem for propositional gossip protocols is coNP-complete.*

Proof. In Corollary 8.2.2 we showed that the simulation problem is coNP-hard, it remains to show that the problem is in coNP. As explained in Theorem 7.2.1, the gossip situation can change at most n^2 times along any computation and skipping a call that does not change the current gossip situation will give us another valid computation. Should the protocol P not be able to simulate the protocol P', we may give the rooted call sequence $\mathbf{c}$, of length n^2 or less, which is possible in P' but not in P. It is possible for this call sequence to be of length n^2 or less, as we may prune a longer call sequence, leaving just productive calls where the gossip situation changes to leave a rooted call sequence of length $n^2 - n$ or less, as the gossip situation may change at most $n^2 - n$ times. After reaching this gossip situation, the final call in this call sequence, which is possible in P' but not in P, must be available, and hence the length of $\mathbf{c}$ is n^2 or less. This may then therefore be checked in polynomial time. □

Theorem 8.2.4. *The bisimilarity problem for propositional gossip protocols is coNP-hard.*

Proof. To show the problem is coNP-hard we will create a polynomial time reduction from the 3-SAT problem, such that termination's NO instances match with 3-SAT's YES instances. This follows in much the same way as the proof of Theorem 5.2.2 in Chapter 5. We construct two protocols in the same way as described in Theorem 5.2.2, however, in the first of these two protocols we remove the option for f to make a call if it learns all clause secrets. Hence, for the first protocol, for each Boolean variable, x, used in the 3-SAT formula, we will create three agents: variable agent (x), true agent ($x_\top$), and false agent ($x_\bot$). For every $i \leq n$, where n is the number of clauses in the 3-SAT formula, we have agent c_i that corresponds to the i-th clause. Now we define the rules of the protocol for each of the agents. For each variable agent, x, while this agent is only familiar with its own secret X, x wants to call agent $x_\top$ or agent $x_\bot$ and the scheduler will decide which of these calls takes place. This essentially sets the value of the x variable in the 3-SAT formula to true or false respectively.

Now any true agent, $x_\top$, and false agent, $x_\bot$, that was called by his corresponding variable agent can make a call to any clause agent, c, whose corresponding clause in the 3-SAT formula uses the literal x or $\neg x$, respectively. This is done with the guard $(X \wedge \neg C)$, to ensure this call cannot be repeated.

Once a clause agent receives a call, this means the corresponding clause is satisfied by the current truth assignment. Once this happens, the clause agent calls f, with a guard $(X \wedge \neg F)$, to make a call to f that cannot be repeated.

Now, f will learn the secrets of all clause agents only if all clauses have been satisfied. However in this first protocol, agent f will never initiate a call. For the second protocol, for each Boolean variable, x, used in the 3-SAT formula, we will create three agents: variable agent (x), true agent $(x_\top)$, and false agent $(x_\bot)$. For every $i \leq n$, where n is the number of clauses in the 3-SAT formula, we have agent c_i that corresponds to the i-th clause. Now we define the rules of the protocol for each of the agents. For each variable agent, x, while this agent is only familiar with its own secret X, x wants to call agent $x_\top$ or agent $x_\bot$ and the scheduler will decide which of these calls takes place. This essentially sets the value of the x variable in the 3-SAT formula to true or false respectively.

Now any true agent, $x_\top$, and false agent, $x_\bot$, that was called by his corresponding variable agent can make a call to any clause agent, c, whose corresponding clause in the 3-SAT formula uses the literal x or $\neg x$, respectively. This is done with the guard $(X \wedge \neg C)$, to ensure this call cannot be repeated.

Once a clause agent receives a call, this means the corresponding clause is satisfied by the current truth assignment. Once this happens, the clause agent calls f, with a guard $(X \wedge \neg F)$, to make a call to f that cannot be repeated.

Now, f will learn the secrets of all clause agents only if all clauses have been satisfied. Agent f can only make a call to a fixed variable agent x if he learns all of these clause secrets, with a guard $(C_1 \wedge ... \wedge C_n)$, and he can repeat this call indefinitely. Now, we see that the only way in which these protocols would differ would be if f learns all clause secrets, which is only possible if we have a YES instance of 3-SAT. Hence, the same call sequences are available and the protocols are bisimilar iff we have a NO instance of 3-SAT. □

Theorem 8.2.5. *The bisimilarity problem for propositional gossip protocols is coNP-complete.*

Proof. In Theorem 8.2.4 we showed that the bisimilarity problem is coNP-hard, it remains to show that the problem is in coNP. This can be done in a similar way as in Theorem

8.2.3. Should the protocols P and P' not be bisimilar, this means one of these protocols may not simulate the other. Therefore, let us assume that protocol P does not simulate a second protocol P', i.e., there exists a rooted call sequence $\mathbf{c}$ which is possible in P' but not in P. Should the protocol P not be able to simulate the protocol P', we may give the rooted call sequence $\mathbf{d}$, of length n^2 or less, which is possible in P' but not in P. From here, the proof follows from Theorem 8.2.3. □

8.3 Conclusion

In this chapter we have investigated the problem of simulation for propositional gossip protocols. Simulation is an interesting problem, as it can give us huge amounts of information about protocols if one is known to simulate another which we already have information about. In Theorem 8.2.1 we have shown the result that LNS may simulate a propositional gossip protocol if and only if the protocol terminates. Not only is this an interesting result, but it also gave us the basis for moving on to questions about the complexity of the simulation problem. We found the simulation problem to be coNP-complete, which was also the case with the problem of finding if two protocols are bisimilar.

Chapter 9

Conclusion

9.1 Summary of Results

In this Thesis we have investigated propositional gossip protocols. The aim was to answer the following questions:

Question 1: What is the required communication graph for a correct propositional gossip protocol?

Question 1 was answered in Chapter 3. We investigated the different possible call structures that are required for a correct propositional gossip protocol. This led us to the main result of Chapter 3 in Theorem 3.2.9, that no correct propositional gossip protocol exists whose undirected communication graph is not complete.

Question 2: Is there a lower bound on the number of calls required in a correct propositional gossip protocol?

Question 2 was investigated in Chapter 4, building on results found in Chapter 3. The knowledge that the communication graph must be complete was a starting point for our solution. The focus was on proving Theorem 4.1.1, that every correct propositional protocol on $n > 3$ agents uses at least $2n - 2$ calls in the worst case, solving Problem 7 as stated in [9]. This result shows that although propositional gossip protocols are simpler to apply, and require less memory than general case gossip protocols, they also cannot be as efficient

as epistemic protocols at spreading information to ensure each agent ends up as an expert in a terminating state. All results in Chapter 4 were with this aim in mind, however other useful results were also found that are applicable elsewhere when investigating problems on the minimal number of calls required on gossip protocols in general.

Question 3: What is the complexity of checking if a propositional gossip protocol is correct?

Chapter 5 focused on Question 3. With the knowledge that the communication graph must be complete we had a useful starting point in determining the complexity of checking whether a propositional gossip protocol is correct. We again gathered some useful results, such as Lemma 5.2.1 showing that a propositional protocol will not terminate if and only if a call is made without the caller gaining new information within the first $|\mathsf{G}|^2$ calls. With results such as this we began by showing that the termination problem was coNP-complete in 5.2.2, before showing the main result that the problem of checking if a given propositional gossip protocol P is correct is coNP-complete in Theorem 5.2.11.

Question 4: How does the introduction of fair schedulers change the results of Questions 1 and 3?

Question 4 was investigated in Chapters 6 and 7. We started by trying to find the required communication graph for a correct propositional gossip protocol under fair schedulers. We started by showing that the type of scheduler does indeed have an effect, with agent-fair having different correct protocols compared to rule-fair. From here we again built up results, culminating in Theorem 6.2.4, where it was shown that for any tree, there exists a propositional gossip protocol which is correct under agent-fairness with this tree as its communication graph, leading to Corollary 6.2.5, showing for any undirected connected graph there exists a propositional gossip protocol which is correct under both agent-fairness and rule-fairness with this graph as its communication graph. This led us to explore the scenario for directed graphs, in Theorems 6.2.6 and 6.2.7 showing that the same result is true for strongly connected directed graphs. Whilst it is possible to have a propositional gossip protocol which is correct under agent-fairness with a weakly connected directed graph as its directed communication graph, in Proposition 6.2.8 it was shown that this is not always the case.

In Chapter 7 we moved on to the second part of Question 4, where we looked at how fair schedulers affected the complexity of the correctness problem. Again we started by focusing on the termination problem in Theorem 7.2.1, showing it to be coNP-complete, with a similar but modified proof to that which showed the original problem to be coNP-complete in Theorem 5.2.2. From here we showed the correctness problem again to be coNP-complete in Theorem 7.2.2.

Question 5: What is the complexity of checking if one propositional gossip protocol may simulate another?

We answered Question 5 in Chapter 8. We found that the simulation problem is coNP-complete in Theorem 8.2.3, and that the bisimilar problem is coNP-complete in Theorem 8.2.5.

9.2 Future Prospects

An immediate extension to Question 1 would be to revisit this problem on directed graphs. We found that the undirected communication graph must be complete, however if we consider the directed graph this certainly does not have to be complete. Another extension would be to investigate increasing the bound found for Question 2 further. Here we have shown that every correct propositional protocol on $n > 3$ agents uses at least $2n - 2$ calls in the worst case, however, it may be possible to increase this further on $n > 4$ agents, or show that $2n - 2$ calls is always the worst case. It may be interesting to investigate different variations of correctness. Our definition of partially correct allows protocols to be deemed as partially correct simply if they never terminate. Investigating protocols which are eventually correct may be an interesting problem, similar to that of fairness, but allowing calls to remain available once the all expert situation is reached and the protocol to still be seen as correct. In our general scenario this change would have little practical impact, as if an expert may make a call, then this call will remain active and therefore may be repeated, and hence must lead to the all expert situation immediately. Yet, as the call sequence *ab.bc.ca* is possible, there are some differences. If a call is made by an expert, and this call then leads to all agents being experts, then this protocol may be seen as being correct. The main difference would be seen when using fair schedulers as, for example, this

would allow a single agent to make all calls, and still be seen as correct. This is another way of viewing the problem, as we may deem that every agent being an expert is sufficient for our purpose, regardless of if calls continue after this stage.

Bibliography

[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14*, pages 113–126. ACM, 2014.

[2] Krzysztof R. Apt, Davide Grossi, and Wiebe van der Hoek. Epistemic protocols for distributed gossiping. In *Proceedings of the 15th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2015)*, volume 215 of *EPTCS*, pages 51–66, 2016.

[3] Krzysztof R. Apt, Davide Grossi, and Wiebe van der Hoek. When are two gossips the same? In Gilles Barthe, Geoff Sutcliffe, and Margus Veanes, editors, *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 36–55. EasyChair, 2018.

[4] Krzysztof R. Apt, Eryk Kopczyński, and Dominik Wojtczak. On the computational complexity of gossip protocols. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 765–771, 2017.

[5] Krzysztof R. Apt and Dominik Wojtczak. On decidability of a logic of gossips. In *Proceedings of the 15th European Conference, JELIA 2016*, volume 10021 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 2016.

[6] Krzysztof R. Apt and Dominik Wojtczak. Common knowledge in a logic of gossips. In *Proc. of the 16th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2017)*, volume 251 of *EPTCS*, pages 10–27, 2017.

[7] Krzysztof R. Apt and Dominik Wojtczak. Decidability of fair termination of gossip protocols. In *Proc. of the 21st International Conference on Logic for Programming,*

Artificial Intelligence and Reasoning (LPAR 21), volume 1 of *Kalpa Publications in Computing*, pages 73–85, 2017.

[8] Krzysztof R. Apt and Dominik Wojtczak. Verification of distributed epistemic gossip protocols. *J. Artif. Intell. Res. (JAIR)*, 62:101–132, 2018.

[9] Krzysztof R. Apt and Dominik Wojtczak. Open problems in a logic of gossips. In *Proceedings Seventeenth Conference on Theoretical Aspects of Rationality and Knowledge (TARK 2019)*, volume 297 of *EPTCS*, pages 1–18, 2019.

[10] Maduka Attamah, Hans Van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. Knowledge and gossip. In *Proceedings of ECAI'14*, pages 21–26. IOS Press, 2014.

[11] Richard T Bumby. A problem with telephones. *SIAM Journal on Algebraic Discrete Methods*, 2(1):13–18, 1981.

[12] Martin C. Cooper, Andreas Herzig, Faustine Maffre, Frédéric Maris, and Pierre Régnier. Simple Epistemic Planning: Generalised Gossiping. In *Proceedings of ECAI 2016*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1563–1564. IOS Press, 2016.

[13] Martin C. Cooper, Andreas Herzig, Frédéric Maris, and Julien Vianey. Temporal epistemic gossip problems. In *European Conference on Multi-Agent Systems*, pages 1–14. Springer, 2018.

[14] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[15] Malvin Gattinger and Jana Wagemaker. Towards an analysis of dynamic gossip in NetKAT. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 280–297. Springer, 2018.

[16] Frank Harary and Allen J. Schwenk. The communication problem on graphs and digraphs. *Journal of the Franklin Institute*, 297(6):491–495, 1974.

[17] Sandra M. Hedetniemi, Stephen T. Hedetniemi, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[18] Andreas Herzig and Faustine Maffre. How to share knowledge by gossiping. In *Proc of the 13th European Conference on Multi-Agent Systems (EUMAS 2015), Revised Selected Papers*, volume 9571, pages 249–263. Springer, 2015.

[19] Andreas Herzig and Faustine Maffre. How to share knowledge by gossiping. *AI Communications*, 30(1):1–17, 2017.

[20] Juraj Hromkovič, Ralf Klasing, Andrzej Pelc, Peter Ruzicka, and Walter Unger. *Dissemination of Information in Communication Networks - Broadcasting, Gossiping, Leader Election, and Fault-Tolerance*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2005.

[21] Richard M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[22] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proc. of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 482–491. IEEE, 2003.

[23] Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *Operating Systems Review*, 41(5):2–7, 2007.

[24] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Providing high availability using lazy replication. *ACM Transactions on Computer Systems (TOCS)*, 10(4):360–391, 1992.

[25] Joseph Livesey and Dominik Wojtczak. Minimal number of calls in propositional protocols. In *International Conference on Reachability Problems*, pages 132–148. Springer, 2021.

[26] Joseph Livesey and Dominik Wojtczak. Propositional gossip protocols. In *Proc. of 23rd International Symposium on Fundamentals of Computation Theory (FCT 2021)*, 2021.

[27] Joseph Livesey and Dominik Wojtczak. Propositional gossip protocols under fair schedulers. In *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022, (to appear)*. Springer, 2022.

[28] Robin Milner. An algebraic definition of simulation between programs. In *International Joint Conference on Artificial Intelligence*, 1971.

[29] Robert Tijdeman. On a telephone problem. *Nieuw Archief voor Wiskunde*, 3(XIX):188–192, 1971.

[30] Hans van Ditmarsch and Malvin Gattinger. The limits to gossip: Second-order shared knowledge of all secrets is unsatisfiable. In *Logic, Language, Information, and Computation*, pages 237–249. Springer International Publishing, 2022.

[31] Hans van Ditmarsch, Davide Grossi, Andreas Herzig, Wiebe van der Hoek, and Louwe B. Kuijer. Parameters for epistemic gossip problems. In *Proceedings of the 12th Conference on Logic and the Foundations of Game and Decision Theory (LOFT 2016)*, 2016.

[32] Hans van Ditmarsch and Ioannis Kokkinis. The expected duration of sequential gossiping. In *Multi-Agent Systems and Agreement Technologies*, pages 131–146. Springer, 2017.

[33] Hans van Ditmarsch, Ioannis Kokkinis, and Anders Stockmarr. Reachability and expectation in gossiping. In Bo An, Ana Bazzan, João Leite, Serena Villata, and Leendert van der Torre, editors, *PRIMA 2017: Principles and Practice of Multi-Agent Systems*, pages 93–109, Cham, 2017. Springer International Publishing.

[34] Hans van Ditmarsch, Wiebe van Der Hoek, and Louwe B Kuijer. The logic of gossiping. *Artificial Intelligence*, 286:103306, 2020.

[35] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. Epistemic protocols for dynamic gossip. *J. of Applied Logic*, 20(C):1–31, 2017.

[36] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. Dynamic gossip. *Bull. Iran. Math. Soc.*, pages 1–28, 2018.