Bachelor thesis
Computing Science

Radboud University

Distributed Knowledge Proofs in the Coq Proof Assistant

Author:
Michiel Philipse
s1016359

Supervisor:
dr. Engelbert Hubbers
hubbers@cs.ru.nl

Second assessor:
dr. Freek Wiedijk
freek@cs.ru.nl

July 27, 2021

Abstract

Multi-agent epistemic modal logics, such as $\mathbf{S5^n}$, have a wide variety of use cases, including artificial intelligence, proofs of cryptographic protocols, database theory, and economics. In any logic, the main goal is usually to prove that a statement holds, using for instance natural deduction. However, constructing natural deduction proofs by hand can be quite tedious. In this thesis, we present an $\mathbf{S5^n}$ Coq implementation, based on an **S5** implementation by Benzmüller and Woltzenlogel Paleo [2]. We adapt the natural deduction proof rules as presented in Huth and Ryan [12] to the Gentzen-style to better suit Coq, and we extend these rules with proof rules for distributed knowledge. We also provide proof tactics with our Coq implementation matching these proof rules. We conclude with multiple examples showcasing how our Coq implementation can be used to prove statements with distributed knowledge.

Contents

Chapter 1

Introduction

Epistemic modal logic is all about reasoning about knowledge. Within epistemic modal logic we are often interested in the logic of multi-agent systems: systems where multiple knowledge agents, often thought of as people, have different knowledge about their situation.

One particular type of knowledge, introduced by Halpern and Moses [9], is *distributed knowledge*. The distributed knowledge of a group of agents is the knowledge that these agents could know if they were to combine all the knowledge of every agent in that group. This is useful for determining what a group of agents could potentially know if they would all share their knowledge with each other in a mathematically perfect manner.

Distributed knowledge and multi-agent epistemic modal logic in general have a wide variety of applications, ranging from game theory within economics [1, 5], to program verification [11] and proofs of cryptograhic protocols [8, 21], to database theory [13, 24], distributed systems [4, 6, 9] and AI in general [17, 10].

In this thesis, we will have a look at how the multi-agent epistemic modal logic $\mathbf{S5^n}$ can be implemented in Coq, with a special focus on distributed knowledge. The main research question that we will be answering is: "How can multi-agent epistemic modal logic with distributed knowledge be implemented in Coq?" To answer this question, we will have to answer the following subquestions:

- How can existing proof rules be adapted for Gentzen-style proofs?
- What proof rules apply to distributed knowledge?
- How can these proof rules be implemented in Coq?

The Coq proof assistant [20] is a formal proof management system that can help us with performing natural deduction proofs. Other Coq implementations of epistemic modal logic do exist, but these usually focus on *common knowledge*, rather than *distributed knowledge*.

We present an $\mathbf{S5^n}$ Coq implementation that contains all four knowledge operators, as defined in Huth and Ryan [12], which includes *knowledge* operator $\mathcal{K}_i$, *shared knowledge* operator $\mathcal{E}_G$, *common knowledge* operator $\mathcal{C}_G$, and *distributed knowledge* operator $\mathcal{D}_G$. We also provide proof tactics for both the Gentzen-style deduction rules for the knowledge operators, as well as for the normal natural deduction rules for the predicate operators, based on ProofWeb [14].

We will start by having a closer look at some of the theoretical aspects of epistemic modal logic and multi-agent systems. After that, we will discuss how existing deduction rules for multi-agent epistemic modal logic, as seen in Huth and Ryan [12], can be adapted for Gentzen-style natural deduction proofs. Then we will define natural deduction rules for distributed knowledge since these are often left out in the literature. And finally, we will discuss our $\mathbf{S5^n}$ Coq implementation, which is based on an **S5** implementation by Benzmüller and Woltzenlogel Paleo [2], and we will show some examples of how it can be used to prove statements about multi-agent systems with natural deduction.

In Chapter 2 we will go over the details of epistemic modal logic and multi-agent systems. In Chapter 3 we will have a closer look at Gentzen-style proofs and the natural deduction rules that apply to distributed knowledge, and in Chapter 4 we will discuss how we implemented these proof rules in Coq. In Chapter 5 we show some example proofs using our Coq implementation. Finally, in Chapter 6 we will talk a bit more about related works, with different approaches to distributed knowledge, and other existing Coq implementations of modal logics.

A summary of the natural deduction rules can be found in Appendix A, and our Coq implementation can be found in Appendix B, as well as online at `https://gitlab.science.ru.nl/mphilipse/coq-multi-agent-systems`.

Chapter 2

Preliminaries

2.1 Epistemic modal logic (S5)

Before we get into distributed logic and multi-agent systems, we first have to take a closer look at modal logic, or more specifically *epistemic* modal logic. All the definitions in this chapter are based on Chapter 5 of Huth and Ryan [12], unless otherwise specified.

Modal logic is an extension of propositional logic. It adds the necessity ($\Box$) and the possibility ($\Diamond$) operators to the well-known propositional operators ($\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$). Here $\Box p$ means that the statement p is *necessarily* true, and $\Diamond p$ means that the statement p is *possibly* true. What that means exactly, depends on the type of modal logic that you are talking about.

Epistemic modal logic is a type of modal logic in which we reason about the knowledge of agents. You can think of an agent as just a person who knows certain things. If we think of ourselves as an agent, then if $\Box p$ holds for us, it means that we know that p holds, and on the other hand if $\Diamond p$ holds for us, it means that p does not contradict our knowledge, so p *could* hold according to us.

2.1.1 Kripke models

In order to mathematically reason about modality, we need semantics to describe what something like $\Diamond p$ actually means. So how do we mathematically reason about the necessity and possibility of propositions?

For this, we use so-called *Kripke semantics*, also known as possible world semantics, to reason about what "worlds" are *possible*. However, before we can get into the details of Kripke semantics, we first have to take a look at *Kripke models*. These Kripke models are used to specify how worlds relate to each other, and this will allow us to reason about the necessity and possibility of propositions.

Definition 2.1. A Kripke model $\mathcal{M}$ is a tuple (W, R, L), consisting of:

- A set W of worlds,
- An accessibility relation $R \subseteq W \times W$ over the worlds,
- A labeling function $L : W \to \mathcal{P}(atoms)$, that maps each world to a set of atomic propositions.

The accessibility relation R links worlds together. For example, $x\,R\,y$ means that world y is accessible from world x, meaning that it is possible to transition from world x to world y. Note that $x\,R\,y$ is just an alternative notation for $(x, y) \in R$. Within *epistemic* modal logic, a world y being accessible from a world x, means that when we are in world x, we cannot know whether or not we actually are in world x or if we are in world y. Or in other words, if x is the "real" world, y could also be the "real" world according to our knowledge.

The labeling function L is used to specify which atomic propositions (i.e. named variables, so propositions without operators) hold in each world. For instance, $p \in L(x)$ says that atomic proposition p holds in world x.

Example 2.1. Say we have two atomic propositions p and q, then we could construct a Kripke model such as shown in Figure 2.1.

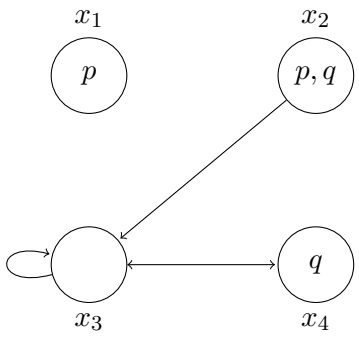

Figure 2.1: An example of a Kripke model

In this model, we have four worlds, $W = \{x_1, x_2, x_3, x_4\}$, which all have been labeled with a set of atomic propositions that hold in those worlds. For example, in world x_1 atomic proposition p holds, but q does not hold, because the labeling function maps x_1 to the singleton set containing only p, so $L(x_1) = \{p\}$. World x_3 may seem to not have a label, but this means that here non of the atomic propositions hold, so $L(x_3) = \emptyset$.

Note that, while we have one world for every combination of atomic propositions, this is not required. You could have multiple worlds labeled with the same set of atomic propositions, and you could have combinations of atomic propositions that do not occur in any world.

The accessibility relation R is depicted in Figure 2.1 with arrows. As you can see in world x_3, it is possible that a world is accessible from itself, as shown with the arrow that loops from x_3 to x_3, thus describing that $x_3 \, R \, x_3$. In this example model, the arrows represent $R = \{(x_2, x_3), (x_3, x_3), (x_3, x_4), (x_4, x_3)\}$. We will later see in Section 2.1.3 that, because of the fact that we are reasoning about *epistemic* modal logic, there are some properties that the accessibility relation should have in order to really make sense.

2.1.2 Kripke semantics

Definition 2.2. Kripke semantics define a satisfaction relation $x \Vdash \varphi$ over a Kripke model $\mathcal{M} = (W, R, L)$, which is used to evaluate whether or not a proposition φ holds in a world $x \in W$. For any atomic proposition a, $x \Vdash a$ holds if and only if $a \in L(x)$. For all the operators that we have in modal logic, the relation is defined inductively for any proposition φ and ψ:

$$
\begin{aligned}
x \Vdash a &\text{ iff } a \in L(x) \\
x \Vdash \neg\varphi &\text{ iff } x \nVdash \varphi \\
x \Vdash \varphi \wedge \psi &\text{ iff } x \Vdash \varphi \text{ and } x \Vdash \psi \\
x \Vdash \varphi \vee \psi &\text{ iff } x \Vdash \varphi \text{ or } x \Vdash \psi \\
x \Vdash \varphi \rightarrow \psi &\text{ iff } x \nVdash \varphi \text{ or } x \Vdash \psi \\
x \Vdash \varphi \leftrightarrow \psi &\text{ iff } (x \Vdash \varphi \text{ if and only if } x \Vdash \psi) \\
x \Vdash \Box\varphi &\text{ iff for each } y \in W \text{ with } x \, R \, y, \text{ we have } y \Vdash \varphi \\
x \Vdash \Diamond\varphi &\text{ iff there exists a } y \in W \text{ such that } x \, R \, y \text{ and } y \Vdash \varphi
\end{aligned}
$$

This means that we now know how to evaluate a statement like $\Diamond\varphi$ for some world in a model. By applying the definitions above, we know that $\Diamond\varphi$ only holds in a world x, if there is another world y, which is accessible from x, in which φ holds. For example, $\Diamond q$ holds for x_3 in the model from Example 2.1, because x_4 is accessible from x_3, since $(x_3, x_4) \in R$, and $x_4 \Vdash q$ holds, because $q \in L(x_4)$.

Often, we do not care as much about which world is the "real" world. Instead, it can be more useful to check if the Kripke model as a whole satisfies a formula.

Definition 2.3. A Kripke model $\mathcal{M} = (W, R, L)$ satisfies a formula φ, if every world $x \in W$ satisfies φ.

In an even more general sense, we often do not even care about the specifics of a Kripke model. Instead, we would often just like to discuss whether a formula is valid in general.

Definition 2.4. A formula φ is *valid*, if it is satisfied by every possible Kripke model.

2.1.3 Axioms for epistemic modal logic

In Example 2.1 we have shown an example of a Kripke model. However, if we think about what the accessibility relation R represents in epistemic modal logic, then this model does not make much sense. To make sure that R makes sense with our logic, we have axioms that specify extra requirements for R, depending on the type of modal logic we are using.

	Axiom name	Property of R	Formula Scheme
K	distributivity	-	$\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$
D	seriality	$\forall x \in W,\, \exists y \in W, x\, R\, y$	$\Box\varphi \to \Diamond\varphi$
T	reflexivity	$\forall x \in W, x\, R\, x$	$\Box\varphi \to \varphi$
4	transitivity	$\forall x, y, z \in W, x\, R\, y \wedge y\, R\, z \to x\, R\, z$	$\Box\varphi \to \Box\Box\varphi$
5	Euclidean	$\forall x, y, z \in W, x\, R\, y \wedge x\, R\, z \to y\, R\, z$	$\Diamond\varphi \to \Box\Diamond\varphi$
B	symmetry	$\forall x, y \in W, x\, R\, y \to y\, R\, x$	$\varphi \to \Box\Diamond\varphi$

Table 2.1: Summary of the different axioms and their formula schemes

Every axiom has a corresponding formula scheme, that should be valid if this axiom is part of the axiomatic system we are using. The relation between the properties of R and the formula schemes is based on *correspondence theory.*

In epistemic modal logic, it turns out that all the above axioms should hold, which gives us an axiomatic system called **S5**. While we will skip over some of the details of correspondence theory, we will discuss why these properties of R and these formula schemes make sense in epistemic modal logic.

Distributivity (K)

In all normal modal logics the *distributivity* axiom (K) holds. In fact, we can prove that its formula scheme is valid, based on Definition 2.2 of the satisfaction relation. If we assume that for some arbitrary x it holds that $x \Vdash \Box(\varphi \to \psi) \wedge \Box\varphi$, then we can prove that $x \Vdash \Box\psi$ must hold:

$$
\begin{aligned}
x \Vdash \Box(\varphi \to \psi) \wedge \Box\varphi \quad &\iff \quad x \Vdash \Box(\varphi \to \psi) \text{ and } x \Vdash \Box\varphi \\
&\iff \quad \forall y \in W, xRy \to (y \Vdash \varphi \to \psi) \text{ and} \\
&\qquad\quad\ \ \forall y \in W, xRy \to (y \Vdash \varphi) \\
&\implies \quad \forall y \in W, xRy \to (y \Vdash \psi) \\
&\iff \quad x \Vdash \Box\psi
\end{aligned}
$$

Seriality (D)

Epistemic modal logic reasons about idealized knowledge agents that cannot have contradicting knowledge. So if an agent knows something ($\Box\varphi$), then it must be that it does not contradict their knowledge ($\Diamond\varphi$). So, in epistemic modal logic, we expect the *seriality* axiom (D) to hold.

But unlike axiom K, this axiom is not enforced by the Kripke semantics. Instead, if we want to make sure that this axiom holds, we have to make sure that the accessibility relation R has a certain property. For seriality, this means that for every $x \in W$, there must exist a $y \in W$ such that $x\,R\,y$, which means that there should always be a world that could be the actual world according to our knowledge.

The example model from Example 2.1 does not yet satisfy the seriality axiom, because there is no world that is accessible from x_1. If we add the relation (x_1, x_1) to R, then it does satisfy the seriality axiom, as shown in Figure 2.2.

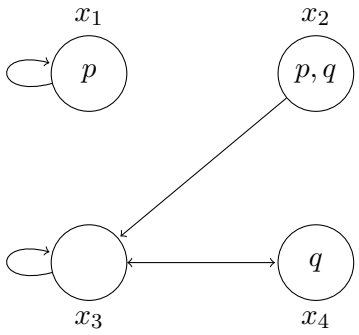

Figure 2.2: A Kripke model that satisfies axioms K and D

Reflexivity (T)

Again, because idealized knowledge agents cannot have contradicting knowledge, they can only know true things. This means that every world should be accessible from itself, since idealized knowledge should not contradict the "real" world. This is what the *reflexivity* axiom (T) describes.

Note that this reflexivity axiom implies the seriality axiom (D), since if for every world $x \in W$ it is the case that $x\,R\,x$, then it automatically follows that there exists a $y \in W$ for every world $x \in W$ such that $x\,R\,y$.

To make our example model also satisfy the reflexivity axiom, we need to also add reflexive relations to worlds x_2 and x_4, as shown in Figure 2.3.

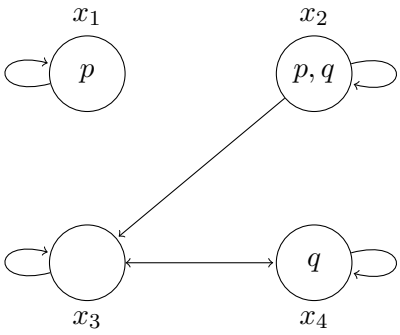

Figure 2.3: A Kripke model that satisfies axioms K, D and T

Transitivity (4)

If y could be the real world according to our knowledge at x (so $x\,R\,y$), and z could be the real world according to our knowledge at y (so $y\,R\,z$), then it would make sense that z could be the real world according to our knowledge at x (so $x\,R\,z$). This also means that if we know something ($\Box\varphi$), then we know that we know that thing ($\Box\Box\varphi$). Thus the *transitivity* axiom (4) should also hold in epistemic modal logic.

This means, that in our example model, because we have $x_2\,R\,x_3$ and $x_3\,R\,x_4$, we must make it such that $x_2\,R\,x_4$, as shown in Figure 2.4. The axiomatic system that combines the axioms K, D, T, and 4 is called **S4**, so we say that Figure 2.4 represents an **S4** Kripke model.

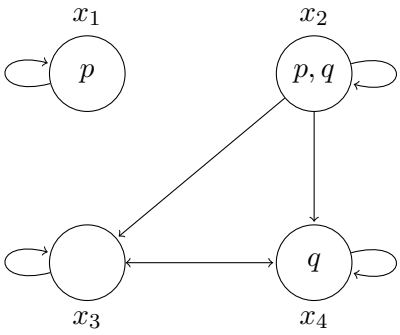

Figure 2.4: An **S4** Kripke model

Symmetry & Euclidean (B & 5)

The *symmetry* axiom (B) says that if y could be the real world according to our knowledge at x (i.e. $x\,R\,y$), then x could be the real world according to our knowledge at y (i.e. $y\,R\,x$). This also means that if φ holds, then we know that φ does not contradict our knowledge (i.e. $\Box\Diamond\varphi$), which means that for all accessible worlds, there exists a world where φ holds.

The *Euclidean* axiom (5) says that if y and z both could be the real world according to our knowledge at x (i.e. $x\,R\,y$ and $x\,R\,z$), then z could be the real world according to our knowledge at y (i.e. $y\,R\,z$). This also means that if φ does not contradict our knowledge (i.e. $\Diamond\varphi$), then we know that φ does not contradict our knowledge (i.e. $\Box\Diamond\varphi$).

These last two axioms are quite similar. In fact, axioms T and 5 imply axiom B. If you assume that $x\,R\,y$, then we know because of reflexivity (T) that $x\,R\,x$, and because of the Euclidean axiom (5), it must be that $y\,R\,x$, because $x\,R\,y \land x\,R\,x \to y\,R\,x$, thus proving symmetry (B).

Also, axioms 4 and B imply axiom 5. If you assume that $x\,R\,y$ and $x\,R\,z$, then we know because of symmetry (B) that $y\,R\,x$, and because of the transitivity axiom (4), it must be that $y\,R\,z$, because $y\,R\,x \land x\,R\,z \to y\,R\,z$, thus proving the Euclidean axiom (5).

And thus, as pointed out by Fact 5.16 from Huth and Ryan [12], a relation is reflexive (T), transitive (4), and Euclidean (5), if and only if it is reflexive (T), transitive (4), and symmetric (B). And with all these axioms, we get an axiomatic system called **S5** (where the axioms K, D, T, 4, 5, and B all hold), and our accessibility relation R becomes an *equivalence relation.*

To turn our example model from Figure 2.4 into a proper **S5** Kripke model, we need to add the relations (x_3, x_2) and (x_4, x_2), as shown in Figure 2.5. Because the accessibility relation R is now an equivalence relation, you can see that our worlds have been split up into separate, all-connected groups.

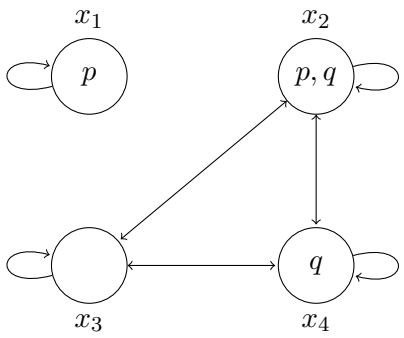

Figure 2.5: An **S5** Kripke model

2.2 Multi-agent systems ($\mathbf{S5^n}$)

In the previous section, we only reasoned about a singular knowledge agent. To reason about groups of knowledge agents who all have potentially different knowledge, we have to look at multi-agent systems.

2.2.1 Kripke models for multi-agent systems

To reason about the knowledge of multiple agents, we need a separate accessibility relation R_i for every agent i in the set of agents $\mathcal{A}$. This means that a Kripke model $\mathcal{M}$ for multi-agent systems consists of a tuple $(W, (R_i)_{i\in\mathcal{A}}, L)$.

Each one of these separate accessibility relations should satisfy all the axioms from **S5**. The axiomatic system where we have n relations for n knowledge agents is called $\mathbf{S5^n}$.

Definition 2.5. An $\mathbf{S5^n}$ Kripke model $\mathcal{M}$ for a set $\mathcal{A}$ of n agents, is a tuple $(W, (R_i)_{i\in\mathcal{A}}, L)$ consisting of:

- A set W of worlds,
- For each i in $\mathcal{A}$, an accessibility relation $R_i \subseteq W \times W$ over the worlds,
- A labeling function $L : W \to \mathcal{P}(atoms)$, that maps each world to a set of atomic propositions.

Now that we have separate accessibility relations for every agent $i \in \mathcal{A}$, the normal necessity operator is no longer sufficient. For multi-agent system we usually use four different kinds of knowledge operators:

- Knowledge operator $\mathcal{K}_i$ for every agent $i \in \mathcal{A}$
- Shared knowledge operator $\mathcal{E}_G$ for a group of agents $G \subseteq \mathcal{A}$
- Common knowledge operator $\mathcal{C}_G$ for a group of agents $G \subseteq \mathcal{A}$
- Distributed knowledge operator $\mathcal{D}_G$ for a group of agents $G \subseteq \mathcal{A}$

With knowledge operator $\mathcal{K}_i$ you can specify that a specific agent i knows something. For instance, the statement $\mathcal{K}_1\varphi$ means that agent 1 knows that φ holds. This is similar to the $\Box$-operator that we have seen before, since $\Box\varphi$ means that we know that φ holds, while $\mathcal{K}_1\varphi$ means that agent 1 knows that φ holds.

Note that, unlike the necessity operator ($\Box$), the possibility operator ($\Diamond$) has no proper alternative for multi-agent systems. While this might seem like a problem, it turns out that this is not a problem at all, since a statement like $\Diamond\varphi$ can be rewritten as $\neg\Box\neg\varphi$. This is equivalent, because saying that something is possible, is the same as saying something is not always impossible.

The shared knowledge operator $\mathcal{E}_G$ can be used to say that every agent in a group G knows that φ holds, which is written as $\mathcal{E}_G\varphi$. This operator can be seen as an abbreviation, since it can be defined in terms of the knowledge operator $\mathcal{K}_i$:

$$\mathcal{E}_G\varphi := \bigwedge_{i \in G} \mathcal{K}_i\varphi$$

The common knowledge operator $\mathcal{C}_G$ is a bit more complex. It is an extension of $\mathcal{E}_G$, where $\mathcal{C}_G\varphi$ means that every agent in G knows that φ holds, but also every agent knows that every agent knows that φ holds, which in turn is also known by every agent, and so on recursively. This operator can be rewritten in terms of $\mathcal{E}_G$, however, since it consists of an infinite conjunction it can technically not be seen as an abbreviation (because infinite conjunctions are not part of the "language" of epistemic modal logic):

$$\mathcal{C}_G\varphi := \bigwedge_{i=1}^{\infty} \mathcal{E}_G^i\varphi, \text{ where } \mathcal{E}_G^n\varphi = \begin{cases} \varphi & \text{if } n = 0 \\ \mathcal{E}_G\mathcal{E}_G^{n-1} & \text{if } n \geq 0 \end{cases}$$

For instance $\mathcal{C}_G\varphi$ specifies that $\mathcal{E}_G\varphi \wedge \mathcal{E}_G(\mathcal{E}_G\varphi) \wedge \mathcal{E}_G(\mathcal{E}_G(\mathcal{E}_G\varphi)) \wedge \ldots$ and so on.

Lastly, we have the distributed knowledge operator $\mathcal{D}_G$, which is the main focus of our thesis. It describes the distributed knowledge of the agents in a group G, which is all the knowledge that can be deduced from the combined knowledge of every agent in G. $\mathcal{D}_G$ cannot easily be expressed in terms of $\mathcal{K}_i$, since it combines knowledge in a special way, where something can be distributed knowledge, even though non of the agents have that particular knowledge themselves, as illustrated in Example 2.2.

Example 2.2. As an example of distributed knowledge, say that agent 1 knows that p holds, and agent 2 knows that $p \to q$ holds, then we know it is part of the distributed knowledge of agent 1 and agent 2 that q holds [9]. Or, written mathematically: $\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}}q$ holds.

2.2.2 Kripke semantics for multi-agent systems

To prove statements such as $\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}}q$, we must first extend the Kripke semantics to properly define what the knowledge operators mean.

Definition 2.6. The satisfaction relation $\Vdash$ is defined as follows for the knowledge operators:

$x \Vdash \mathcal{K}_i\varphi$ iff for each $y \in W$, $x\, R_i\, y$ implies that $y \Vdash \varphi$

$x \Vdash \mathcal{E}_G\varphi$ iff for each $i \in G$, it is the case that $x \Vdash \mathcal{K}_i\varphi$

$x \Vdash \mathcal{C}_G\varphi$ iff for each $k \geq 1$, it is the case that $x \Vdash \mathcal{E}_G^k\varphi$

$x \Vdash \mathcal{D}_G\varphi$ iff for each $y \in W$, we have $y \Vdash \varphi$, whenever $x\, R_i\, y$ for all $i \in G$

Hakli and Negri [7] provide a slightly different definition for the satisfaction relation for $\mathcal{D}_G$:

$$x \Vdash \mathcal{D}_G\varphi \text{ iff for each } y \in W \text{ such that } (x, y) \in \bigcap_{i\in G} R_i, \\ \text{it is the case that } y \Vdash \varphi$$

As you can see, the Kripke semantic definition of $\mathcal{K}_i$ is similar to that of $\Box$, but with the addition of an agent-specific accessibility relation, and the Kripke semantic definitions of $\mathcal{E}_G$ and $\mathcal{C}_G$ correspond to the previously given mathematical definitions.

The definitions for $\mathcal{D}_G$ are a bit more complex. While the definition from Hakli and Negri looks quite different, it describes the same thing as the definition from Huth and Ryan. The intersection from Hakli and Negri is described by Huth and Ryan as "whenever for each $i \in G$ we have $x\, R_i\, y$".

The satisfaction definition for $\mathcal{D}_G$ is somewhat similar to that of $\mathcal{K}_i$, but it uses an intersection of the accessibility relation of all agents in group G. The intersection combines the knowledge of the agents, by only leaving the worlds accessible that do not contradict anyone's knowledge.

If we try to come up with a Kripke model for Example 2.2, where we say that $\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}} q$, we could get something like shown in Figure 2.6. You can see that every arrow has been annotated with either R_1, R_2 or both, to show which relation R_i the arrow is part of.

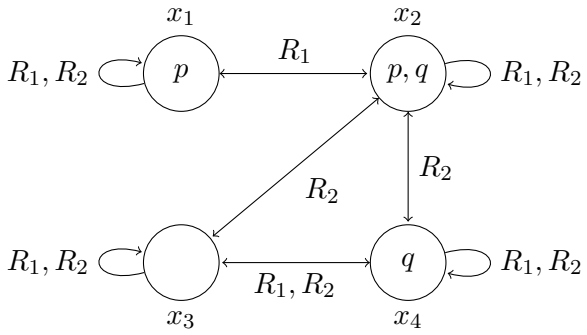

Figure 2.6: An example of an $\mathbf{S5^2}$ Kripke model

Because the accessibility relation R_i is an equivalence relation, it can often be convenient to leave out some of the details while drawing a Kripke model. Since R_i is reflexive, we know that every world is accessible to itself, so we can leave out the looping arrows for every world. And since R_i is symmetric, we know that every arrow must go in both directions, so we can just draw them as lines. This means that we can get a much cleaner-looking Kripke model, as shown in Figure 2.7, which still represents the same model as Figure 2.6.

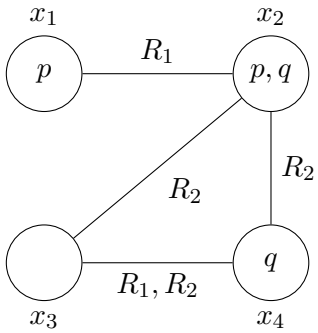

Figure 2.7: An example of a simplified $\mathbf{S5^2}$ Kripke model

You can see that for agent 1, we have chosen to differentiate the worlds in which p holds (x_1 and x_2), and the worlds in which p does not hold (x_3 and x_4). Similarly, for agent 2 we have separated the worlds in which $p \to q$ holds (x_2, x_3 and x_4) and the world where $p \to q$ does not hold (x_1).

Based on the assumption that $\mathcal{K}_1 p$ holds, we know that we must either be in x_1 or x_2, since only from those worlds holds p in every accessible world for agent 1. And based on the assumption that $\mathcal{K}_2(p \to q)$ holds, we know that we must be in either x_2, x_3 or x_4, since only from those worlds $p \to q$ holds in every accessible world for agent 2.

This means, that it follows from the assumption that $\mathcal{K}_1 p$ and $\mathcal{K}_2(p \to q)$ hold, that we must be in x_2. The only world that is accessible from x_2 for both agent 1 and agent 2, is x_2 itself. And in x_2 the atomic proposition q holds, so it follows that $\mathcal{D}_{\{1,2\}} q$ holds whenever $\mathcal{K}_1 p$ and $\mathcal{K}_2(p \to q)$ hold in our model.

However, the Kripke model that we have shown here, is just one of infinitely many possible Kripke models. So instead of showing that a statement like $\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}} q$ holds for one specific model, we would like to prove that it is valid in general. In the next chapter, we will show how we can use proof rules and natural deduction to show that a formula must hold for every world in every possible $\mathbf{S5^n}$ Kripke model.

Chapter 3

Knowledge Proofs

In propositional logic, we can prove a formula like $p \wedge (p \to q) \to q$ using *natural deduction*. We use derivation rules, such as the ones listed in Appendix A.1, to prove a formula based on certain premises. We write for instance $\varphi_1, \varphi_2 \vdash \psi$ to say that we want to prove ψ based on the premise that φ_1 and φ_2 hold. To prove that a formula holds in general, we have to show that for instance $\vdash p \wedge (p \to q) \to q$ holds, without any premises.

The proof of a sequent like $\vdash p \wedge (p \to q) \to q$ can be shown visually as a proof tree, as shown in Figure 3.1. Every horizontal line in such a proof tree represents a deduction rule being applied. At the top of any branch in a proof tree, you will often see the "hyp"-rule, which says that what we are trying to prove is one of the premises on the left-hand side.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \wedge (p \to q) \vdash p \wedge (p \to q)}\,\text{hyp}}{p \wedge (p \to q) \vdash p \to q}\,\wedge e2 \quad \dfrac{\dfrac{}{p \wedge (p \to q) \vdash p \wedge (p \to q)}\,\text{hyp}}{p \wedge (p \to q) \vdash p}\,\wedge e1}{p \wedge (p \to q) \vdash q}\,\to e}{\vdash p \wedge (p \to q) \to q}\,\to i$$

Figure 3.1: A proof of $\vdash p \wedge (p \to q) \to q$

Similarly, we would like to prove formulas about multi-agent systems using natural deduction. For instance, we would like to prove the formula $\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}} q$ from Example 2.2. For this, we need new derivation rules for the knowledge operators.

3.1 Natural deduction rules for S5

Before we have a look at the proof rules of the knowledge operators in $\mathbf{S5^n}$, we shall first have a look at the proof rules of **S5**.

In the modal logic **S5**, we have the necessity operator ($\Box$) and the possibility operator ($\Diamond$). However, remember that the possibility operator can be defined in terms of the necessity operator, e.g. $\Diamond p \iff \neg\Box\neg p$. So, to prove things in **S5** with natural deduction, it is sufficient to just have proof rules for $\Box$.

Definition 3.1. Huth and Ryan [12] define a $\Box$-introduction rule and a $\Box$-elimination rule as follows:

$$\dfrac{\begin{array}{|c|}\hline \vdots \\ \varphi \\ \hline\end{array}}{\Box\varphi}\,\Box i \qquad \dfrac{\Box\varphi \qquad \begin{array}{|c|}\hline \vdots \\ \varphi \\ \vdots \\ \hline\end{array}}{}\,\Box e$$

Huth and Ryan use these rules in *Fitch notation*. We can rewrite them with full sequents to better fit in *Gentzen-style* proof trees, as Hubbers has done in the (unpublished) course material of the course "Logic and Applications":

$$\dfrac{\Gamma, \boxed{\Box \vdash \varphi}}{\Gamma \vdash \Box\varphi}\,\Box i \qquad \dfrac{\Gamma \vdash \Box\varphi}{\Gamma, \boxed{\Box\ \Gamma \vdash \varphi}}\,\Box e$$

In these rules, you see dashed boxes labeled by $\Box$. These dashed boxes mean that their contents apply to an arbitrary accessible world. This is somewhat similar to the $\forall i$ and $\forall e$ rules, where you are also reasoning about an arbitrary instance of a certain type.

The context Γ in these rules normally contains the premises that we already have assumed to hold. However, with the addition of these "dashed boxes", the context Γ can now also contain the various nested dashed-boxed that we might be in.

Definition 3.2. We define the modal context Γ as follows for **S5**:

$$\Gamma \quad ::= \quad \cdot \mid \Gamma,\ \varphi \mid \Gamma,\ \boxed{\Box}$$

To make it easier to define this grammar, we only indicate the beginning of a dashed box with $\boxed{\Box}$. These dashed boxes always reach to end of the sequent, so it is sufficient to formally only indicate where they start, even though we prefer to to draw the full box.

Note that we usually do not write the initial comma at the start of a (non-empty) context or just after the beginning of a dashed box.

When you are trying to prove something within in a dashed box, you can only use the assumptions from inside of that dashed box in the context. Assumptions outside (i.e. before the start) of the current dashed box are related to a different world and thus cannot be used.

The $\overline{\Gamma}$ in the $\Box e$ rule represents a normal context *without* dashed boxes inside of it. This is because when you eliminate a dashed box, that dashed box is allowed to have assumptions in it, but the dashed box that you are eliminating should be the innermost dashed box, as you can not eliminate multiple boxes at once.

Definition 3.3. We define the normal context $\overline{\Gamma}$ as follows:

$$\overline{\Gamma} \quad ::= \quad \cdot \mid \overline{\Gamma},\ \varphi$$

Using the rules $\Box i$ and $\Box e$, we can now prove a formula like $\Box p \wedge \Box(p \to q) \to \Box q$, as shown in Figure 3.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Box p \wedge \Box(p \to q) \vdash \Box p \wedge \Box(p \to q)}\,\text{hyp}}{\Box p \wedge \Box(p \to q) \vdash \Box(p \to q)}\,\wedge e2}{\Box p \wedge \Box(p \to q),\ \boxed{\Box \vdash p \to q}}\,\Box e \qquad \dfrac{\dfrac{\dfrac{}{\Box p \wedge \Box(p \to q) \vdash \Box p \wedge \Box(p \to q)}\,\text{hyp}}{\Box p \wedge \Box(p \to q) \vdash \Box p}\,\wedge e1}{\Box p \wedge \Box(p \to q),\ \boxed{\Box \vdash p}}\,\Box e}{\Box p \wedge \Box(p \to q),\ \boxed{\Box \vdash q}}\,\to e}{\Box p \wedge \Box(p \to q) \vdash \Box q}\,\Box i}{\vdash \Box p \wedge \Box(p \to q) \to \Box q}\,\to i$$

Figure 3.2: A proof of $\Box p \wedge \Box(p \to q) \to \Box q$

Based on the axioms that hold in **S5**, which we discussed in Section 2.1.3, we can define some more proof rules. While the axioms K, D, B, T, 4, and 5 all hold in **S5**, it is sufficient to have rules only for T, 4, and 5. These three axioms imply that the accessibility relation R is an equivalence relation, and thus they imply symmetry (B) and seriality (D).

Definition 3.4. The natural deduction rules based on **S5**'s axioms are defined as follows:

$$\frac{\Gamma \vdash \Box\varphi}{\Gamma \vdash \varphi}\,\mathrm{T} \qquad \frac{\Gamma \vdash \Box\varphi}{\Gamma \vdash \Box\Box\varphi}\,4 \qquad \frac{\Gamma \vdash \neg\Box\varphi}{\Gamma \vdash \Box\neg\Box\varphi}\,5$$

Again, these rules are based on Huth and Ryan [12], but have been rewritten in Gentzen-style by Hubbers.

3.2 Natural deduction rules for $\mathbf{S5^n}$

Now that we have seen the proof rules for **S5**, let us now see how we can extend these to $\mathbf{S5^n}$. What proof rules can we define for the knowledge operators $\mathcal{K}_i$, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$?

3.2.1 Natural deduction rules for $\mathcal{K}_i$

The knowledge operator $\mathcal{K}_i$ is very similar to the necessity operator from **S5**, with the only difference being that $\mathcal{K}_i$ has an argument for which agent i knows something. Because of this, the rules that we have seen for **S5** can easily be adapted to fit the $\mathcal{K}_i$ operator.

Definition 3.5. The deduction rules for the knowledge operator $\mathcal{K}_i$ are defined similarly to those of the necessity operator $\Box$:

$$\frac{\Gamma, \boxed{\mathcal{K}_i \vdash \varphi}}{\Gamma \vdash \mathcal{K}_i\varphi}\,\mathcal{K}i \qquad \frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma, \boxed{\mathcal{K}_i\ \Gamma \vdash \varphi}}\,\mathcal{K}e$$

$$\frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma \vdash \varphi}\,\mathcal{K}T \qquad \frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma \vdash \mathcal{K}_i\mathcal{K}_i\varphi}\,\mathcal{K}4 \qquad \frac{\Gamma \vdash \neg\mathcal{K}_i\varphi}{\Gamma \vdash \mathcal{K}_i\neg\mathcal{K}_i\varphi}\,\mathcal{K}5$$

Note that the dashed boxes here are labeled with $\mathcal{K}_i$. This is because these dashed boxes now say that we are reasoning about an arbitrary world that is accessible to agent i.

Using these rules, we can construct a proof similar to Figure 3.2 for the formula $\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \to \mathcal{K}_1 q$, as shown in Figure 3.3.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \vdash \mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q)}\,\text{hyp}}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \vdash \mathcal{K}_1(p \to q)}\,\wedge e2}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q), \boxed{\mathcal{K}_1 \vdash p \to q}}\,\mathcal{K}e \qquad \dfrac{\dfrac{\dfrac{}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \vdash \mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q)}\,\text{hyp}}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \vdash \mathcal{K}_1 p}\,\wedge e1}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q), \boxed{\mathcal{K}_1 \vdash p}}\,\mathcal{K}e}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q), \boxed{\mathcal{K}_1 \vdash q}}\,\to e}{\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \vdash \mathcal{K}_1 q}\,\mathcal{K}i}{\vdash \mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \to \mathcal{K}_1 q}\,\to i$$

Figure 3.3: A proof of $\mathcal{K}_1 p \wedge \mathcal{K}_1(p \to q) \to \mathcal{K}_1 q$

3.2.2 Introduction and elimination rules for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$

As Huth and Ryan [12] point out, the knowledge operators $\mathcal{E}_G$, $\mathcal{C}_G$ and $\mathcal{D}_G$ are also "box-like" connectives. Just like $x \Vdash \mathcal{K}_i\varphi$, which holds if and only if for each world $y \in W$ with $x\, R_i\, y$ it is the case that $y \Vdash \varphi$, the $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ operators can be defined in a similar manner.

Definition 3.6. We can define special accessibility relations $R_{\mathcal{E}_G}$, $R_{\mathcal{C}_G}$, and $R_{\mathcal{D}_G}$ in terms of R_i:

$$\begin{aligned} x\, R_{\mathcal{E}_G}\, y &\quad \text{iff} \quad x\, R_i\, y \text{ for some } i \in G \\ x\, R_{\mathcal{C}_G}\, y &\quad \text{iff} \quad x\, R^k_{\mathcal{E}_G}\, y \text{ for some } k \geq 1 \\ x\, R_{\mathcal{D}_G}\, y &\quad \text{iff} \quad x\, R_i\, y \text{ for all } i \in G \end{aligned}$$

Note that our definition of $R_{\mathcal{C}_G}$is a bit different from the one by Huth and Ryan, as we say that $x\, R^k_{\mathcal{E}_G}\, y$ must hold for *some* $k \geq 1$, while they say that $x\, R^k_{\mathcal{E}_G}\, y$ must hold for *each* $k \geq 1$. This is most likely a mistake by Huth and Ryan. In Lemma `RCC` in Appendix B.1, which will be further discussed in Section 4.2.2, we prove that our definition makes sense.

Definition 3.7. Based on these relations we can define alternative definitions of the knowledge operators $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$:

$$\begin{aligned} x \Vdash \mathcal{E}_G\varphi &\quad \text{iff} \quad \text{for each } y \in W \text{ such that } x\, R_{\mathcal{E}_G}\, y, \text{ it is the case that } y \Vdash \varphi \\ x \Vdash \mathcal{C}_G\varphi &\quad \text{iff} \quad \text{for each } y \in W \text{ such that } x\, R_{\mathcal{C}_G}\, y, \text{ it is the case that } y \Vdash \varphi \\ x \Vdash \mathcal{D}_G\varphi &\quad \text{iff} \quad \text{for each } y \in W \text{ such that } x\, R_{\mathcal{D}_G}\, y, \text{ it is the case that } y \Vdash \varphi \end{aligned}$$

These alternative definitions are similar to the original definition of $\mathcal{K}_i$ in Definition 2.6. Because of this, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ are all what Huth and Ryan call "box-like" connectives, which means that we can define introduction and elimination rules similar to $\mathcal{K}i$ and $\mathcal{K}e$ for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, but with appropriately labeled dashed boxes for each of the operators.

Definition 3.8. The introduction and elimination rules for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ are defined as follows:

$$\frac{\Gamma,\ \boxed{\mathcal{E}_G\ \vdash \varphi}}{\Gamma \vdash \mathcal{E}_G\varphi}\,\mathcal{E}i \qquad \frac{\Gamma,\ \boxed{\mathcal{C}_G\ \vdash \varphi}}{\Gamma \vdash \mathcal{C}_G\varphi}\,\mathcal{C}i \qquad \frac{\Gamma,\ \boxed{\mathcal{D}_G\ \vdash \varphi}}{\Gamma \vdash \mathcal{D}_G\varphi}\,\mathcal{D}i$$

$$\frac{\Gamma \vdash \mathcal{E}_G\varphi}{\Gamma,\ \boxed{\mathcal{E}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{E}e \qquad \frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma,\ \boxed{\mathcal{C}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{C}e \qquad \frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma,\ \boxed{\mathcal{D}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{D}e$$

Here the dashed boxes still indicate that we are reasoning about an arbitrary accessible world, but how exactly the arbitrary world is accessible depends on the operator. For example, in a $\mathcal{D}_G$-labeled dashed box, we reason about an arbitrary world that is accessible through the $R_{\mathcal{D}_G}$ relation.

Note that, while Huth and Ryan [12] do mention the $\mathcal{E}i$, $\mathcal{E}e$, $\mathcal{C}i$, and $\mathcal{C}e$ rules, they do not mention the $\mathcal{D}i$ and $\mathcal{D}e$ rules. This is because they chose to only provide the rules for $\mathcal{K}_i$, $\mathcal{E}_G$, and $\mathcal{C}_G$. However, the $\mathcal{D}i$ and $\mathcal{D}e$ rules are valid since $\mathcal{D}_G$ is a "box-like" connective just like the other knowledge operators.

Definition 3.9. Since we now have four different kinds of dashed boxes, we define the modal context Γ for $\mathbf{S5^n}$ as follows:

$$\Gamma \quad ::= \quad \cdot \mid \Gamma, \varphi \mid \Gamma, \boxed{\mathcal{K}_i} \mid \Gamma, \boxed{\mathcal{E}_G} \mid \Gamma, \boxed{\mathcal{C}_G} \mid \Gamma, \boxed{\mathcal{D}_G}$$

Again, we use partial boxes to indicate the beginning of the different kinds of dashed boxes. For $\mathbf{S5^n}$, the normal context $\overline{\Gamma}$ remains the same as Definition 3.3 for **S5**.

3.2.3 Axioms for $\mathcal{E}_\mathbf{G}$, $\mathcal{C}_\mathbf{G}$, and $\mathcal{D}_\mathbf{G}$

We have seen that for $\mathcal{K}_i$, just like as for $\Box$, the axioms T, 4, and 5 all hold because R_i is an equivalence relation. Because of this, we have the rules $\mathcal{K}T$, $\mathcal{K}4$, and $\mathcal{K}5$ as shown in Definition 3.5. Now let us see if we can define similar rules for the other operators.

Axioms for $\mathcal{E}_\mathbf{G}$

As Huth and Ryan point out, axioms 4 and 5 do not hold for the operator $\mathcal{E}_G$ [12]. This becomes obvious if we look at the definition of $R_{\mathcal{E}_G}$.

From Definition 3.6 we know that $x \, R_{\mathcal{E}_G} \, y$ holds if and only if $x \, R_i \, y$ for some $i \in G$. So it can be that for $G = \{1, 2\}$ we have $x \, R_1 \, y$ and $y \, R_2 \, z$, which means that we have $x \, R_{\mathcal{E}_{\{1,2\}}} \, y$ and $y \, R_{\mathcal{E}_{\{1,2\}}} \, z$, but we do not necessarily have $x \, R_{\mathcal{E}_{\{1,2\}}} \, z$, which means that $R_{\mathcal{E}_G}$ is not necessarily transitive (axiom 4), as shown in Figure 3.4. Similarly, the $R_{\mathcal{E}_{\{1,2\}}}$ in Figure 3.4 is not Euclidean (axiom 5) either, since we have $y \, R_{\mathcal{E}_{\{1,2\}}} \, x$ and $y \, R_{\mathcal{E}_{\{1,2\}}} \, z$, but we do not have $x \, R_{\mathcal{E}_{\{1,2\}}} \, z$.

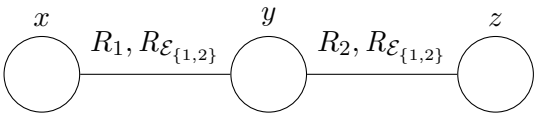

Figure 3.4: An example where $R_{\mathcal{E}_G}$ is not transitive and not Euclidean

This $R_{\mathcal{E}_{\{1,2\}}}$ is reflexive (axiom T), although the reflexive relations are not shown explicitly in Figure 3.4. However, the reflexivity of $R_{\mathcal{E}_G}$ does not always hold. In particular, if $G = \emptyset$, then there is no $i \in G$ for which $x \, R_i \, y$ holds for any worlds x and y. This means that if $G = \emptyset$, then $x \, R_{\mathcal{E}_G} \, x$ does not hold which means $R_{\mathcal{E}_G}$ is not reflexive.

Axioms for $\mathcal{C}_G$

For the $\mathcal{C}_G$ operator, axioms 4 and 5 do hold. This is because $R_{\mathcal{C}_G}$ is defined in terms of $R^k_{\mathcal{E}_G}$ for some $k \geq 1$, which means that $R_{\mathcal{C}_G}$ is the transitive closure of $R_{\mathcal{E}_G}$. As shown in Figure 3.5, you can see that $R_{\mathcal{C}_{\{1,2\}}}$ expands upon $R_{\mathcal{E}_{\{1,2\}}}$ with the relations (x, z) and (z, x), making the relation transitive. This is because (x, z) and (z, x) are in $R^2_{\mathcal{E}_{\{1,2\}}}$, since (x, y), (y, z), (z, y), and (y, x) are in $R_{\mathcal{E}_{\{1,2\}}}$.

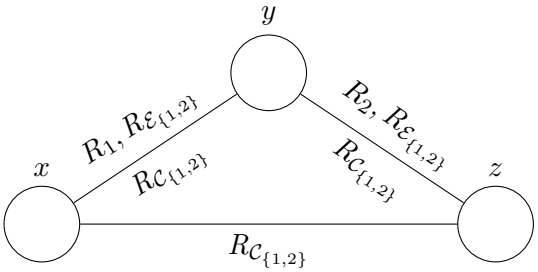

Figure 3.5: $R_{\mathcal{C}_G}$ is transitive and Euclidean

The relation $R_{\mathcal{C}_G}$ is also symmetric (axiom B) since it inherits this property from $R_{\mathcal{E}_G}$, which in turn inherits it from R_i. As we explained in Section 2.1.3, any relation that is both symmetric (axiom B) and transitive (axiom 4) is also Euclidean (axiom 5), so the relation $R_{\mathcal{C}_G}$ is also Euclidean. However, similar to $R_{\mathcal{E}_G}$, $R_{\mathcal{C}_G}$ is not reflexive when $G = \emptyset$, because $R_{\mathcal{C}_G}$ cannot be reflexive if $R_{\mathcal{E}_G}$ is not reflexive.

Definition 3.10. Because $R_{\mathcal{C}_G}$ is transitive (axiom 4) and Euclidean (axiom 5), we can have the following proof rules for $\mathcal{C}_G$:

$$\frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{C}_G\mathcal{C}_G\varphi}\,\mathcal{C}4 \qquad \frac{\Gamma \vdash \neg\mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{C}_G\neg\mathcal{C}_G\varphi}\,\mathcal{C}5$$

Axioms for $\mathcal{D}_G$

Huth and Ryan also point out that, unlike $R_{\mathcal{E}_G}$ and $R_{\mathcal{C}_G}$, $R_{\mathcal{D}_G}$ is a proper equivalence relation [12]. This is because the transitive, Euclidean, and reflexive properties of R_i carry over to $R_{\mathcal{D}_G}$, since $R_{\mathcal{D}_G}$ is the intersection of multiple R_i's.

Unlike for $R_{\mathcal{E}_G}$ and $R_{\mathcal{C}_G}$, which are not reflexive when $G = \emptyset$, $R_{\mathcal{D}_G}$ is always reflexive, even when $G = \emptyset$. This is because $x\, R_{\mathcal{E}_G}\, y$ holds if and only if there exists some agent $i \in G$ for which $x\, R_i\, y$ holds, which means that if there are no agents in the group G, then $x\, R_{\mathcal{E}_G}\, y$ (and by extension also $x\, R_{\mathcal{C}_G}\, y$) automatically does not hold. On the other hand, $x\, R_{\mathcal{D}_G}\, y$ holds if and only if for all agents $i \in G$ we have that $x\, R_i\, y$ holds, which means that if there are no agents in the group G, then $x\, R_{\mathcal{D}_G}\, y$ automatically holds, so in particular $x\, R_{\mathcal{D}_G}\, x$ also holds for any world x.

Definition 3.11. Because $R_{\mathcal{D}_G}$ is reflexive (axiom T), transitive (axiom 4), and Euclidean (axiom 5), we can have the following proof rules for $\mathcal{D}_G$:

$$\frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma \vdash \varphi}\,\mathcal{D}T \qquad \frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma \vdash \mathcal{D}_G\mathcal{D}_G\varphi}\,\mathcal{D}4 \qquad \frac{\Gamma \vdash \neg\mathcal{D}_G\varphi}{\Gamma \vdash \mathcal{D}_G\neg\mathcal{D}_G\varphi}\,\mathcal{D}5$$

These rules are not mentioned by Huth and Ryan, but given the properties of the relation $R_{\mathcal{D}_G}$, they do make sense. In Section 4.2.3 we will prove that these rules are indeed valid, using our Coq implementation.

3.2.4 Proof rules relating different knowledge operators

We have seen that we can have introduction and elimination rules for the different knowledge operators. We have also seen that we can have a variety of proof rules related to the axioms that hold for certain knowledge operators. In this section, we will see that we can also have proof rules that relate certain knowledge operators to other knowledge operators.

Shared knowledge $\mathcal{E}_\mathbf{G}$ related to $\mathcal{K}_\mathbf{i}$

Based on Definition 2.6, we know that $x \Vdash \mathcal{E}_G p$ holds if and only if $x \Vdash \mathcal{K}_i p$ for every $i \in G$. Because of this, we can prove that $\mathcal{E}_G\varphi$ holds by proving that $\mathcal{K}_i\varphi$ holds for every $i \in G$, which gives us the proof rule $\mathcal{KE}$. We can also prove that $\mathcal{K}_i\varphi$ holds by proving that $\mathcal{E}_G\varphi$ holds, as long as agent i is in group G, which gives us the proof rule $\mathcal{EK}$.

Definition 3.12. We have the following natural deduction rules relating shared knowledge $\mathcal{E}_G$ to $\mathcal{K}_i$:

$$\frac{\Gamma \vdash \mathcal{K}_i\varphi \text{ for each } i \in G}{\Gamma \vdash \mathcal{E}_G\varphi}\,\mathcal{KE} \qquad \frac{\Gamma \vdash \mathcal{E}_G\varphi \qquad i \in G}{\Gamma \vdash \mathcal{K}_i\varphi}\,\mathcal{EK}$$

Definition 3.13. To prove that an agent i is in a group G, we have the proof rule $\in G$, which says that $i \in G$ holds if i appears in the set G:

$$\frac{}{\Gamma \vdash i \in \{\ldots, i, \ldots\}}\,\in G$$

Common knowledge $\mathcal{C}_\mathbf{G}$ related to $\mathcal{E}_\mathbf{G}$ and $\mathcal{K}_\mathbf{i}$

Definition 2.6 also tells us that $x \Vdash \mathcal{C}_G p$ holds if and only if $x \Vdash \mathcal{E}_G^k p$ for every $k \geq 1$. Because of this, we can prove that a sequence of one or more repeated $\mathcal{E}_G$'s holds based on $\mathcal{C}_G$, which gives us the proof rule $\mathcal{CE}$. With the $\mathcal{CE}$ rule and repeated applications of the $\mathcal{K}i$, $\mathcal{EK}$, and $\mathcal{K}e$ rules, we can also define a proof rule for any sequence of $\mathcal{K}_{i_j}$'s, as long as every agent i_j is in group G, which gives us the proof rule $\mathcal{CK}$.

Definition 3.14. Natural deduction rules relating $\mathcal{C}_G$ with $\mathcal{E}_G$ and $\mathcal{K}_i$:

$$\frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{E}_G \ldots \mathcal{E}_G\varphi}\,\mathcal{CE} \qquad \frac{\Gamma \vdash \mathcal{C}_G\varphi \qquad i_j \in G}{\Gamma \vdash \mathcal{K}_{i_1} \ldots \mathcal{K}_{i_k}\varphi}\,\mathcal{CK}$$

In Figure 3.6 you can see why the $\mathcal{CK}$ rule works. We can use repeated applications of $\mathcal{K}i$, $\mathcal{EK}$, and $\mathcal{K}e$ to replace all $\mathcal{K}_i$'s with $\mathcal{E}_G$'s, as long as every agent i_j is in group G. Finally, once all $\mathcal{K}_i$'s have been replaced, we can use the $\mathcal{CE}$ rule to prove the sequence of $\mathcal{E}_G$'s based on a single $\mathcal{C}_G$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash \mathcal{C}_G\varphi}{\vdash \mathcal{E}_G \ldots \mathcal{E}_G\varphi}\,\mathcal{CE} \qquad i_j \in G}{\vdots}\,(\mathcal{EK}, \mathcal{K}e)^*}{\mathcal{K}_{i_1} \ \ldots \ \mathcal{K}_{i_{k-1}} \vdash \mathcal{E}_G\varphi}\,\mathcal{K}e \qquad i_k \in G}{\mathcal{K}_{i_1} \ \ldots \ \mathcal{K}_{i_{k-1}} \vdash \mathcal{K}_{i_k}\varphi}\,\mathcal{EK}}{\vdots}\,(\mathcal{K}i)^*}{\mathcal{K}_{i_1} \vdash \mathcal{K}_{i_2} \ldots \mathcal{K}_{i_k}\varphi}\,\mathcal{K}i}{\vdash \mathcal{K}_{i_1} \ldots \mathcal{K}_{i_k}\varphi}\,\mathcal{K}i$$

Figure 3.6: The $\mathcal{CK}$ rule described using the $\mathcal{K}i$, $\mathcal{EK}$, $\mathcal{K}e$, and $\mathcal{CE}$ rules

Distributed knowledge $\mathcal{D}_\mathbf{G}$ related to $\mathcal{K}_\mathbf{i}$, $\mathcal{E}_\mathbf{G}$, and $\mathcal{C}_\mathbf{G}$

Inspired by the rules from Huth and Ryan relating $\mathcal{E}_G$ and $\mathcal{C}_G$ to other knowledge operators, we can also come up with some rules for relating the distributed knowledge operator $\mathcal{D}_G$ to other knowledge operators.

For the distributed knowledge operator $\mathcal{D}_G$, it makes sense that when one agent i in a group G knows that φ holds, then φ is part of the distributed knowledge of group G.

This is because for $x \Vdash \mathcal{D}_G\varphi$ to hold, as we have seen in Definition 2.6, it must be that for each $y \in W$ we have $y \Vdash \varphi$ whenever we have $x \, R_i \, y$ for all agents $i \in G$. If we know that $x \Vdash \mathcal{K}_i\varphi$ holds, we know that for each $y \in W$ we have $y \Vdash \varphi$ whenever we have $x \, R_i \, y$. So if for a particular agent $j \in G$ it holds that $\mathcal{K}_j\varphi$, then it must be that $\mathcal{D}_G\varphi$ holds, because for each $y \in W$ such that $x \, R_j \, y$ it holds that $y \Vdash \varphi$, which means that $y \Vdash \varphi$ must also hold for each $y \in W$ where $x \, R_i \, y$ holds for all agents $i \in G$.

Definition 3.15. We define the following natural deduction rule for proving $\mathcal{D}_G$ based on $\mathcal{K}_i$:

$$\dfrac{\Gamma \vdash \mathcal{K}_i\varphi \qquad i \in G}{\Gamma \vdash \mathcal{D}_G\varphi}\,\mathcal{KD}$$

This property of $\mathcal{D}_G$ is similar to axiom A9 by Halpern and Moses [10], which is also discussed by Lambert [15], and axiom A11 by van der Hoek [11]. However, these do not consider the distributed knowledge of a group G, instead they consider the distributed knowledge of all agents. Hakli and Negri [7] have also shown that this property of $\mathcal{D}_G$ holds for their logic $\mathbf{G3KE_D}$ as well.

Similarly, we could also define proof rules $\mathcal{ED}$ and $\mathcal{CD}$. However, it turns out that these proof rules are not necessary, since we can already construct proofs for $\mathcal{D}_G$ based on $\mathcal{E}_G$ and $\mathcal{C}_G$ using the proof rules we already have, as shown in Figure 3.7 and Figure 3.8 respectively.

$$\dfrac{\dfrac{\vdash \mathcal{E}_G\varphi \qquad i \in G}{\vdash \mathcal{K}_i\varphi}\,\mathcal{EK} \qquad i \in G}{\vdash \mathcal{D}_G\varphi}\,\mathcal{KD}$$

Figure 3.7: Proving $\mathcal{D}_G$ based on $\mathcal{E}_G$ (as long as there exists some $i \in G$)

$$\dfrac{\dfrac{\dfrac{\vdash \mathcal{C}_G\varphi}{\vdash \mathcal{E}_G\varphi}\,\mathcal{CE} \qquad i \in G}{\vdash \mathcal{K}_i\varphi}\,\mathcal{EK} \qquad i \in G}{\vdash \mathcal{D}_G\varphi}\,\mathcal{KD}$$

Figure 3.8: Proving $\mathcal{D}_G$ based on $\mathcal{C}_G$ (as long as there exists some $i \in G$)

Notice that the $\mathcal{ED}$ and $\mathcal{CD}$ rules would only work if we know that there exists some $i \in G$. This is again because an empty group G would cause problems. When $G = \emptyset$, $\mathcal{E}_G\varphi$, and by extension $\mathcal{C}_G\varphi$, will always holds, because $\mathcal{E}_G\varphi$ asks if $\mathcal{K}_i\varphi$ holds for every $i \in G$, which it does if there are no agents $i \in G$.

However, when $G = \emptyset$, $\mathcal{D}_G\varphi$ will most likely not hold. This is because $x \Vdash \mathcal{D}_G\varphi$ asks if $y \Vdash \varphi$ holds whenever we have $x\,R_i\,y$ for all agents $i \in G$. Because $G = \emptyset$, for every x and y $x\,R_i\,y$ holds for all agents $i \in G$, because there are no agents $i \in G$. Because of this, when $G = \emptyset$, $\mathcal{D}_G\varphi$ asks if every world satisfies φ, which is most likely not the case in most Kripke models.

Because of the problems when $G = \emptyset$, and the fact that if $G \neq \emptyset$ then the proof can be done in a couple of steps with other proof rules, we have chosen not to include the $\mathcal{ED}$ and $\mathcal{CD}$ rules in the rest of our thesis.

In Appendix A.2 we have summarized all of the proof rules for $\mathbf{S5^n}$ that we have discussed in this chapter. In the next chapter, we will show how we can implement these rules in the proof assistant Coq to verify the validity of these proof rules, and to help us with the construction of knowledge proofs.

Chapter 4

Knowledge Proofs in Coq

In this chapter, we will explain how we have implemented the multi-agent epistemic modal logic $\mathbf{S5^n}$ in the proof assistant Coq, based on Benzmüller and Woltzenlogel Paleo [2].

Our implementation is a *shallow embedding*, which means that logical formulas are written in the logic of the proof assistant, as opposed to a *deep embedding*, where logical formulas are represented with custom-defined datatypes [23, 19]. Because of this, our implementation is very expressive and can easily be extended with more operators.

We will start with discussing our implementation of the knowledge operators $\mathcal{K}_i$, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, and after that we will discuss how we have implemented the proof rules in Coq.

4.1 Knowledge operators in Coq

Our $\mathbf{S5^n}$ implementation is based on an **S5** implementation by Benzmüller and Woltzenlogel Paleo [2]. They have implemented the necessity ($\Box$) and possibility ($\Diamond$) operators in Coq. Their implementation works by lifting propositions to a type $W \to Prop$, where W is the type for worlds, which means that the truth value of a proposition depends on the world that you are in.

```
Parameter W : Type. (* Type for worlds *)
Definition P := W -> Prop. (* Type of lifted propositions *)
```

This means that you can describe `x` $\Vdash$ `p` for an `x:W` and a `p:P` as `p x`, which is world `x` applied to lifted proposition `p`. So `p x` returns true if the proposition `p` holds in world `x`.

Note that, while we do closely follow the ideas from Benzmüller and Woltzenlogel Paleo, we have renamed some of their variable names to be more consistent with Huth and Ryan [12].

Because they use these lifted propositions, they also define alternative versions of the predicate logic operators in order for them to work with these lifted propositions. For example, they have defined an `m/\` operator that works the same as the normal `/\` operator, but for lifted propositions:

```
Definition mand (p q:P)(w:W) := (p w) /\ (q w).
Notation "p m/\ q" := (mand p q) (at level 79, right associativity).
```

Similarly, definitions for `m~`, `m\/`, `m->`, `m<->`, `mforall`, and `mexists` can be found in the full implementation as seen in Appendix B.1.

4.1.1 $\mathcal{K}_\mathbf{i}$ in Coq

To turn Benzmüller and Woltzenlogel Paleo's **S5** implementation into $\mathbf{S5^n}$, we start by turning the their necessity operator ($\Box$) into the knowledge operator $\mathcal{K}_i$.

From Definition 2.2 we know that $x \Vdash \Box\varphi$ holds if and only if for each $y \in W$ such that $x\,R\,y$, it is the case that $y \Vdash \varphi$. Benzmüller and Woltzenlogel Paleo have implemented this as follows:

```
Parameter R: W -> W -> Prop. (* Accessibility relation *)
Definition box (p:P) := fun x => forall y, (R x y) -> (p y).
```

The accessibility relation R is defined as a function $W \to W \to Prop$, meaning that `R x y` holds whenever world `y` is accessible from world `x`.

In $\mathbf{S5^n}$, we want to have a separate accessibility relation for every agent. So we add the agent as an argument to the accessibility relation:

```
Parameter A: Type. (* Type for agents *)
Parameter R: A -> W -> W -> Prop. (* Accessibility relation *)
```

With this updated accessibility relation, `R i x y` now means that world `y` is accessible from world `x` according to agent `i`. If we now update the definition of the necessity operator (`box`) by adding an argument for an agent `i`, we get the knowledge operator $\mathcal{K}_i$:

```
Definition K (i:A) (p:P) := fun x => forall y, (R i x y) -> (p y).
```

You can see that this specification matches the original definition for $\mathcal{K}_i$ in Definition 2.6, since `K i p x` holds if and only if for all worlds `y` we have that `R i x y` implies `p y`.

4.1.2 Groups of agents in Coq

To implement the other knowledge operators in Coq, we first need to think about how we can represent a group of agents in Coq. We decided to use a minimalistic approach, instead of using for instance Coq's `List` module.

In our implementation, a group of agents is represented as a function $A \to Prop$, which maps agents to true or false depending on whether or not they are in the group.

```
Definition G := A -> Prop. (* Groups of agents *)
```

This means that `g i` holds if and only if agent `i` is in group `g`. We chose this representation because sets can be seen as constants in our $\mathbf{S5^n}$ implementation, and representing sets of agents as functions $A \to Prop$ makes proving certain properties a bit easier.

One downside of this representation of groups is that defining specific groups can get a bit messy. That is why we have created a special notation that allows you to easily define a group containing a specific set of agents:

```
Notation "[ ]" := (fun i:A => False) (format "[ ]").
Notation "[ i1 ]" := (fun i:A => i=i1).
Notation "[ i1 , i2 , .. , iN ]" := (fun i:A =>
    (i=i1 \/ (i=i2 \/ .. (i=iN \/ False) ..))
).
```

This notation is based on Coq's `ListNotations`, but instead of returning a list structure, it returns a function that returns true only for agents in the group. For an empty set, it will always return false. For a set containing a single agent, it will return true only if the agent is that agent in the set. For a set containing multiple agents, it will return true only if the agent is equal to one of the agents in the set. Note that the `\/ False` in the notation above is necessary for Coq to find the recursive pattern of the notation.

To make it even easier to prove that $i \in G$, we have made the following proof tactic, which represents the $\in G$ rule from Definition 3.13:

```
Ltac minG :=
  match goal with
  | |- _ \/ _ => (left; reflexivity) || (right; minG)
  | |- _ = _  => reflexivity
  end ||
  fail "(could not find equal element in disjunction)".
```

This proof tactic recursively looks for the part of the disjunction where there is an equality that holds. Note that this proof tactic only works for groups that are of the same form as the ones you get by using our list notation.

Often we might want to say that something holds for all agents in a group, or that there exists an agent in the group for which something holds. For this we have also created special notation:

```
Notation "'gforall'  i : g , p" := (forall i : A, g i -> p)
  (at level 200, i ident, right associativity) : type_scope.
Notation "'gexists'  i : g , p" := (exists i : A, g i /\ p)
  (at level 200, i ident, right associativity) : type_scope.
```

In the next section where we will define the knowledge operators $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, you will see how this notation makes it easy to quantify over agents in a group.

4.1.3 $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ in Coq

Now that we know how we can represent groups of agents, we can start to define the knowledge operators $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ according to their definitions in Definition 2.6.

Definition 2.6 tells us that $x \Vdash \mathcal{E}_G\varphi$ holds if and only if for each agent $i \in G$ it is the case that $x \Vdash \mathcal{K}_i\varphi$. Using our `gforall` notation, this can be implemented as follows:

```
Definition E (g:G) (p:P) := fun x => gforall i : g, K i p x.
```

We also know that $x \Vdash \mathcal{C}_G\varphi$ holds if and only if for each $k \geq 1$ it is the case that $x \Vdash \mathcal{E}_G^k\varphi$. So let us first define $\mathcal{E}_G^k$:

```
Fixpoint E' (n:nat) (g:G) (p:P) :=
  match n with
  | 0 => E g p
  | S n' => E' n' g (E g p)
end.
```

As you can see, we have defined $\mathcal{E}_G^k$ as a fixed point construction, where for instance `E' 0 g p` is equal to `E g p`, and `E' 1 g p` is equal to `E' 0 g (E g p)` which is equal to `E g (E g p)`.

Using this definition of `E'`, we can define the common knowledge operator $\mathcal{C}_G$ as follows:

```
Definition C (g:G) (p:P) := fun x => forall n : nat, E' n g p x.
```

It is important to note that because we are using natural numbers, we start at $n = 0$, while in Definition 2.6 we use $k \geq 1$. You can see that our definition of `E' 0 g p` describes what would normally be $\mathcal{E}_G^1 p$, and `E' 1 g p` describes what would be $\mathcal{E}_G^2 p$, and so on. We decided to use natural numbers for our implementation since they are easier to work with within Coq.

And last but not least, we have the distributed knowledge operator $\mathcal{D}_G$. We know that $x \Vdash \mathcal{D}_G\varphi$ holds if and only if for each $y \in W$ such that for all $i \in G$ it is the case that $x\, R_i\, y$, we have $y \Vdash \varphi$, which we can specify in Coq as follows:

```
Definition D (g:G) (p:P) := fun x => forall y, (gforall i : g, R i x y) -> p y.
```

4.1.4 Modal validity in Coq

Benzmüller and Woltzenlogel Paleo do not explicitly define Kripke models. Instead, their embedding works by reasoning about arbitrary worlds. This is because the main goal is to prove the validity of formulas, meaning that we want to show that a formula must always hold in any arbitrary world of any Kripke model.

From Definition 2.4 we know that a formula φ is valid if it is satisfied by every possible Kripke model, and from Definition 2.3 we know that a Kripke model satisfies a formula φ if every world satisfies that formula. So in order to show that a formula is valid, we must show that it holds for every arbitrary world `w`:

```
Definition Valid (p:P) := forall w : W, p w.
```

If we combine this validity definition with our previously defined knowledge operators, we can for instance prove that Example 2.2 is valid by proving that the following theorem holds:

```
Theorem example2_2: Valid(
    K i1 p m/\  K i2 (p m-> q)
  m->
    D [i1, i2] q
).
```

4.2 Deduction rules for $S5^n$ in Coq

To prove the previously shown `Theorem example2_2`, we first need to implement the deduction rules from Section 3.2 in Coq.

However, we also need natural deduction rules for the normal propositional operators, such as the ones shown in Appendix A.1. For this, we will be using a modified version of the proof rules provided by ProofWeb [14]. These rules, modified for the modal operators by Benzmüller and Woltzenlogel Paleo, can be found in Appendix B.2.

4.2.1 Deduction rules for $\mathcal{K}_\mathbf{i}$ in Coq

Introduction and elimination rules for $\mathcal{K}_\mathbf{i}$

Benzmüller and Woltzenlogel Paleo [2] have provided in their **S5** implementation the proof tactics `box_i` and `box_e`, based on the deduction rules $\Box i$ and $\Box e$ from Definition 3.1.

```
Ltac box_i := let w := fresh "w" in let R := fresh "R"
              in (intro w at top; intro R at top).
Ltac box_elim H w1 H1 := match type of H with
        ((box ?p) ?w) => cut (p w1);
                         [intros H1 | (apply (H w1); try assumption)]
end.
Ltac box_e H H1:= match goal with | [ |- (_ ?w) ] => box_elim H w H1 end.
```

Based on these proof tactics, we would like to define tactics for the rules $\mathcal{K}i$ and $\mathcal{K}e$ from Definition 3.5. For the $\mathcal{K}i$ rule, the tactic is pretty much the same as for $\Box i$:

```
Ltac mK_i := let w := fresh "w" in let R := fresh "R0"
             in (intro w at top; intro R).
```

However, the `box_e` tactic is a bit more complicated, since it uses an auxiliary tactic `box_elim`. Instead of modifying this complicated proof tactic, we have chosen to implement the tactic for the $\mathcal{K}e$ rule (and most of the other rules that we will see later on as well) using lemmas.

```
Lemma Ke_lemma:
forall (i:A) (x:W) (p:P) (y:W), (K i p x) -> (R i x y) -> p y.
Proof.
intros i x p y H.
generalize y.
exact H.
Qed.
```

This lemma says that if $x \Vdash \mathcal{K}_i p$ holds for any agent i, any world x, and any proposition p, then $y \Vdash p$ holds if $x \, R_i \, y$ holds. This means that we can use this lemma to get out of a $\mathcal{K}_i$-labeled dashed box, since we can show that $y \Vdash p$ holds for some arbitrary accessible world y, by showing that $\mathcal{K}_i p$ holds.

```
Ltac mK_e i w := match goal with
  | |- _ ?w0 => apply (Ke_lemma i w); [> clear dependent w0 | try assumption]
end.
```

The proof tactic `mK_e` applies the `Ke_lemma` to the goal and generates two subgoals: `K i p x` and `R i x y`. On this second subgoal we call `try assumption`, to have Coq prove this subgoal automatically if `R i x y` is present in the current context. On the first subgoal we call `clear dependent w0`, to remove all the assumptions in the context that depend on `w0`. The assumptions that get removed are the ones that would be in the normal context $\overline{\Gamma}$ within the dashed box that gets eliminated in the definition of $\mathcal{K}e$ in Definition 3.5.

Axiom rules for $\mathcal{K}_\mathbf{i}$

Benzmüller and Woltzenlogel Paleo [2] have also provided axioms for the reflexivity (axiom T), transitivity (axiom 4), and symmetry (axiom B) of accessibility relation R, as well as a theorem that proves that R must also be Euclidean (axiom 5) based on these other axioms:

```
Axiom Rreflexivity: forall i x, R i x x.
Axiom Rtransitivity: forall i x y z, (R i x y) -> (R i y z) -> (R i x z).
Axiom Rsymmetry: forall i x y, (R i x y) -> (R i y x).
Theorem Reuclidean: forall i x y z, (R i x y) -> (R i x z) -> (R i y z).
Proof.
intros.
cut (R i y x).
```

```
+ intro.
  apply Rtransitivity with (y := x).
  exact H1.
  exact H0.
+ apply Rsymmetry.
  exact H.
Qed.
```

These axioms can be used to prove lemmas for the $\mathcal{K}T$, $\mathcal{K}4$, and $\mathcal{K}5$ rules. The proofs for these lemmas can be seen in the full implementation in Appendix B.1, but these are the lemmas that we have proven to hold:

```
Lemma KT_lemma: Valid (mforall i, mforall p, (K i p) m-> p).
Lemma K4_lemma: Valid (mforall i, mforall p, (K i p) m-> (K i (K i p))).
Lemma K5_lemma: Valid (mforall i, mforall p, m~ (K i p) m-> (K i (m~ (K i p)))).
```

The proof tactics for the $\mathcal{K}T$, $\mathcal{K}4$, and $\mathcal{K}5$ rules consist of straightforward applications of the lemmas above:

```
Ltac mKT := apply KT_lemma.
Ltac mK4 := apply K4_lemma.
Ltac mK5 := apply K5_lemma.
```

4.2.2 Relations $R_{\mathcal{E}_G}$, $R_{\mathcal{C}_G}$, and $R_{\mathcal{D}_G}$ in Coq

In order to define introduction and elimination rules for the operators $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, we need to be able to specify that a world is $R_{\mathcal{E}_G}$, $R_{\mathcal{C}_G}$, or $R_{\mathcal{D}_G}$-accessible. The relations $R_{\mathcal{E}_G}$, $R_{\mathcal{C}_G}$, and $R_{\mathcal{D}_G}$are defined according to their definitions in Definition 3.6:

```
Definition RE (g:G) (x y:W) := gexists i : g, R i x y.
Fixpoint RE' (n:nat) (g:G) (x y:W) :=
  match n with
  | 0 => RE g x y
  | S n' => exists z:W, RE' n' g x z /\ RE g z y
end.
Definition RC (g:G) (x y:W) := exists n : nat, RE' n g x y.
Definition RD (g:G) (x y:W) := gforall i : g, R i x y.
```

Similar to the definition of `E'` and `C` in Section 4.1.3, there is again a small difference in this definition of `RE'` and `RC` as opposed to Definition 3.6, since we use a natural number `n` instead of a $k \geq 1$.

We can now also prove lemmas to show that the alternative definitions of $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, as seen in Definition 3.7, are equivalent to the original definitions of $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$:

```
Lemma REE: forall g p x, (E g p x) <-> (forall y, (RE g x y) -> (p y)).
Lemma RE'E': forall n g x p, (E' n g p x) <-> (forall y, (RE' n g x y) -> (p y)).
Lemma RCC: forall g x p, (C g p x) <-> (forall y, (RC g x y) -> (p y)).
Lemma RDD: forall g p x, (D g p x) <-> (forall y, (RD g x y) -> (p y)).
```

The lemmas will be used in the proofs of the lemmas of the deduction rules. Again, the proofs for these lemmas can be seen in the full implementation in Appendix B.1. The proof of `Lemma RE'E'` helps us prove `Lemma RCC` a little bit easier. It was while proving `Lemma RCC`, that we discovered the error in Huth and Ryan [12] for the relation $R_{\mathcal{C}_G}$, as noted by Definition 3.6.

4.2.3 Deduction rules for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ in Coq

Introduction and elimination Rules for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$

For the introduction and elimination rules of $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ we will define lemmas, just like we did for $\mathcal{K}e$. The lemmas for these different operators are all really similar, the only difference being which relation they use. The proofs of these lemmas are also really similar, since we did most of the hard work already in proving lemmas `REE`, `RCC`, and `RDD`.

Here are the lemmas for the introduction rules $\mathcal{E}i$, $\mathcal{C}i$, and $\mathcal{D}i$:

```
Lemma Ei_lemma:
forall (g:G) (x:W) (p:P), (forall y, (RE g x y) -> p y) -> (E g p x).

Lemma Ci_lemma:
forall (g:G) (x:W) (p:P), (forall y, (RC g x y) -> p y) -> (C g p x).

Lemma Di_lemma:
forall (g:G) (x:W) (p:P), (forall y, (RD g x y) -> p y) -> (D g p x).
```

The proofs of these lemmas all consist of an introduction of the variables `g`, `x`, and `p`, an introduction of an assumption, an application of either `REE`, `RCC`, or `RDD`, and an exact application of the main assumption.

And here are the lemmas for the elimination rules $\mathcal{E}e$, $\mathcal{C}e$, and $\mathcal{D}e$:

```
Lemma Ee_lemma:
forall (g:G) (x:W) (p:P) (y:W), (E g p x) -> (RE g x y) -> p y.

Lemma Ce_lemma:
forall (g:G) (x:W) (p:P) (y:W), (C g p x) -> (RC g x y) -> p y.

Lemma De_lemma:
forall (g:G) (x:W) (p:P) (y:W), (D g p x) -> (RD g x y) -> p y.
```

Here, the proofs consist of an introduction of the variables `g`, `x`, `p`, and `y`, an introduction of an assumption, a generalization on `y`, an application of either `REE`, `RCC`, or `RDD`, and an exact application of the main assumption.

The proof tactics are defined similar to `mK_i` and `mK_e`, but with applications of the correct lemmas:

```
Ltac mE_i := let w := fresh "w" in let R := fresh "RE0"
             in (apply Ei_lemma; intro w at top; intro R).

Ltac mE_e g w := match goal with
  | |- _ ?w0 => apply (Ee_lemma g w); [> clear dependent w0 | try assumption]
end.
```

```
Ltac mC_i := let w := fresh "w" in let R := fresh "RC0"
             in (apply Ci_lemma; intro w at top; intro R).

Ltac mC_e g w := match goal with
  | |- _ ?w0 => apply (Ce_lemma g w); [> clear dependent w0 | try assumption]
end.

Ltac mD_i := let w := fresh "w" in let R := fresh "RD0"
             in (apply Di_lemma; intro w at top; intro R).

Ltac mD_e g w := match goal with
  | |- _ ?w0 => apply (De_lemma g w); [> clear dependent w0 | try assumption]
end.
```

Rules for relating $\mathcal{K}_\mathbf{i}$, $\mathcal{E}_\mathbf{G}$, $\mathcal{C}_\mathbf{G}$, and $\mathcal{D}_\mathbf{G}$

To implement the proof rules $\mathcal{KE}$, $\mathcal{EK}$, $\mathcal{KD}$, and $\mathcal{CE}$ we have proven the appropriate lemmas and defined a proof tactic that applies those lemmas:

```
Lemma KE_lemma: forall (g:G) (p:P) (x:W), (gforall i:g, K i p x) -> E g p x.
Lemma EK_lemma: forall (g:G) (p:P) (x:W), E g p x -> gforall i:g, K i p x.
Lemma KD_lemma: forall (i:A) (g:G) (p:P) (x:W), g i -> K i p x -> D g p x.
Lemma CE_lemma: forall (g:G) (p:P) (x:W) (n:nat), C g p x -> E' n g p x.
```

The $\mathcal{KE}$ rule requires you to prove that all agents in a group know something. The proof tactic `mKE` will make sure that these turn into separate goals by applying `repeat destruct` on the assumption which describes the group. Because a group is represented as a disjunction of each agent in that group, as described in Section 4.1.2, the `repeat destruct` will create a separate subgoal for each agent in the group.

```
Ltac mKE := let i := fresh "i" in let H := fresh "H"
            in apply KE_lemma; intros i H; repeat destruct H.
```

The tactics for the $\mathcal{EK}$ and $\mathcal{KD}$ rules apply the corresponding lemmas, but also try to automatically solve the subgoals of the form $i \in G$ using our `minG` tactic.

```
Ltac mEK g := apply EK_lemma with g; [>  | try minG].
Ltac mKD i := apply KD_lemma with i; [> try minG | ].
```

The lemma for the $\mathcal{CE}$ rule proves that `E' n g p x` is implied by `C g p x` for any natural number `n`. However, we normally do not use the `E'` operator in our definitions and proofs. Instead, we would like to be able to prove a sequence of $\mathcal{E}_G$'s (e.g. $\mathcal{E}_G(\mathcal{E}_G(\varphi))$) based on $\mathcal{C}_G$.

The proof tactic `mCE` recursively turns a goal consisting of a sequence of $\mathcal{E}_G$'s into a goal consisting of a single $\mathcal{E}'_G$, and then it applies the lemma from above:

```
Ltac mCE := match goal with
  | |- E' ?n ?g' (E ?g' ?p') _ => unfold E'; fold (E' (S n) g' p'); mCE
  | |- E' ?n ?g' ?p' _  => apply CE_lemma
  | |- E ?g' (E ?g' ?p') _ => fold (E' 1 g' p'); mCE
  | |- E ?g' ?p' _ => fold (E' 0 g' p'); apply CE_lemma
end.
```

Finally we have the proof rule $\mathcal{CK}$. Unlike the other proof rules, we did not use a lemmas for this one. Instead, we constructed a proof tactic that follows the structure from Figure 3.6, where we use the rules $\mathcal{K}i$, $\mathcal{EK}$, $\mathcal{K}e$, and $\mathcal{CE}$ to get the functionality of the $\mathcal{CK}$ rule.

We accomplished this by first defining a proof tactic `mCK'`, which turns a goal consisting of a sequence of $\mathcal{K}_i$'s into a single goal with $\mathcal{E}_G$ for some group G, and then we applied our `mCE` rule to turn goal with $\mathcal{E}_G$ into a goal with $\mathcal{C}_G$:

```
Ltac mCK' g:= match goal with
  | |- K ?i' (K _ _) ?w' => mK_i; mCK' g; only 1 : mK_e i' w'; only 1 : mEK g
  | |- K ?i' ?p' _ => mEK g
end.
Ltac mCK g := mCK' g; only 1 : mCE.
```

Since the `mEK` rule automatically applies the `minG` tactic, all the subgoals of $i_j \in G$ for the $\mathcal{CK}$ rules can be solved automatically as well. However, if they are not solved automatically, the `only 1` notation will focus only on the main goal, while keeping the other subgoals.

Axiom rules for $\mathcal{C}_G$ and $\mathcal{D}_G$

Finally, we have also proven lemmas for the rules $\mathcal{C}4$, $\mathcal{C}5$, $\mathcal{D}T$, $\mathcal{D}4$, and $\mathcal{D}5$. The axiom rules for $\mathcal{D}_G$ were not that hard to prove using the axioms for `R`. However, the axiom rules for $\mathcal{C}_G$ were a bit more difficult.

In order to prove the lemmas for $\mathcal{C}4$ and $\mathcal{C}5$, we first proved that the relation `RC` is transitive (`Lemma RCtransitivity`) and Euclidean (`Lemma RCEuclidean`). But to prove that this is indeed the case, we also had to prove that the relation `RE'` is transitive (`Lemma RE'transitivity`) and Euclidean (`Lemma RE'Euclidean`).

To prove that `RE'` is Euclidean, we had to prove two more auxiliary lemmas. We have proven `Lemma RE'symmetry`, which shows that relation `RE'` is symmetric, and we have proven `Lemma RE'reverse`, whichs shows the definition of `RE'` also works in reverse. So while `RE' (S n) g x y` is defined as `exists z:W, RE' n g x z /\ RE g z y`, it would be equivalent to use `exists z:W, RE g x z /\ RE' n g z y`.

The full implementation, including all the lemmas and proofs, can be found in Appendix B.1.

Chapter 5

Example Proofs

Now that we have seen which proof rules apply to the different knowledge operators, and now we know how these can be implemented in Coq, we shall have a look at a couple of example proofs. We will show some of these proofs as Gentzen-style proof trees, and we will show the corresponding Coq proofs using the proof tactics from Section 4.2.

5.1 Proofs of Logical Omniscience

As Huth and Ryan [12] point out, the operators $\mathcal{K}_i$, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ are all closed under "logical consequence" since the distributivity axiom K (sometimes referred to as *logical omniscience*) holds for all of them.

Theorem 5.1. $\mathcal{K}_i$, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ are closed under logical consequence:

$$\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \to \mathcal{K}_i\psi$$
$$\mathcal{E}_G\varphi \wedge \mathcal{E}_G(\varphi \to \psi) \to \mathcal{E}_G\psi$$
$$\mathcal{C}_G\varphi \wedge \mathcal{C}_G(\varphi \to \psi) \to \mathcal{C}_G\psi$$
$$\mathcal{D}_G\varphi \wedge \mathcal{D}_G(\varphi \to \psi) \to \mathcal{D}_G\psi$$

We have already seen a proof of the first formula with $\mathcal{K}_i$ in Figure 3.3. Using the $\mathcal{K}i$, $\to e$, and $\mathcal{K}e$ rules we can show that agent i knows that ψ holds, since agent i knows that φ implies ψ, and agent i knows that φ holds:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \vdash \mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi)}\,\text{hyp}}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \vdash \mathcal{K}_i(\varphi \to \psi)}\,\wedge e2}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi),\ \boxed{\mathcal{K}_i \vdash \varphi \to \psi}}\,\mathcal{K}e \qquad \dfrac{\dfrac{\dfrac{}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \vdash \mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi)}\,\text{hyp}}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \vdash \mathcal{K}_i\varphi}\,\wedge e1}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi),\ \boxed{\mathcal{K}_i \vdash \varphi}}\,\mathcal{K}e}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi),\ \boxed{\mathcal{K}_i \vdash \psi}}\,\to e}{\mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \vdash \mathcal{K}_i\psi}\,\mathcal{K}i}{\vdash \mathcal{K}_i\varphi \wedge \mathcal{K}_i(\varphi \to \psi) \to \mathcal{K}_i\psi}\,\to i$$

Of course we can also do this proof in Coq with our implementation. If we define parameters `p` and `q` as lifted propositions and parameter `i` as an agent, we can specify the following theorem:

```
Parameters p q: P.
Parameter i: A.

Theorem HR338_K: Valid(
    K i p m/\ K i (p m-> q)
  m->
    K i q
).
```

To prove this theorem, we will use the Coq equivalents of the same rules as used in the proof tree above. The two branches of the proof tree are indicated with bullets, here using the symbol `+`:

```
Proof. mv.
mimp_i H.
mK_i.
mimp_e p.
+ mK_e i w.
  mcon_e1 (K i (p m-> q)).
  mhyp H.
+ mK_e i w.
  mcon_e2 (K i p).
  mhyp H.
Qed.
```

The proofs of the other formulas for $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$ are really similar. In fact, we use the same proof rules, but with $\mathcal{E}i$, $\mathcal{C}i$, or $\mathcal{D}i$ instead of $\mathcal{K}i$, and $\mathcal{E}e$, $\mathcal{C}e$, or $\mathcal{D}e$ instead of $\mathcal{K}e$. Here are the theorems and proofs of the logical omniscience of the operators $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$:

```
Parameters g: G.

Theorem HR338_E: Valid(
    E g p m/\ E g (p m-> q)
  m->
    E g q
).
Proof. mv.
mimp_i H.
mE_i.
mimp_e p.
+ mE_e g w.
  mcon_e1 (E g (p m-> q)).
  mhyp H.
+ mE_e g w.
  mcon_e2 (E g p).
  mhyp H.
Qed.
```

```
Theorem HR338_C: Valid(
    C g p m/\ C g (p m-> q)
  m->
    C g q
).
Proof. mv.
mimp_i H.
mC_i.
mimp_e p.
+ mC_e g w.
  mcon_e1 (C g (p m-> q)).
  mhyp H.
+ mC_e g w.
  mcon_e2 (C g p).
  mhyp H.
Qed.
```

```
Theorem HR338_D: Valid(
    D g p m/\ D g (p m-> q)long
  m->
    D g q
).
Proof. mv.
mimp_i H.
mD_i.
mimp_e p.
+ mD_e g w.
  mcon_e1 (D g (p m-> q)).
  mhyp H.
+ mD_e g w.
  mcon_e2 (D g p).
  mhyp H.
Qed.
```

While we have not drawn proof trees for these proofs, you can probably imagine what they might look like, similar to the proof tree for $\mathcal{K}_i$.

5.2 Proof of Example 2.2

Now let us have a look at Example 2.2. This example is somewhat similar to the formulas of logical omniscience that we have proven in Section 5.1, but now showing that the knowledge of two agents can be combined in the distributed knowledge of those two agents:

Theorem 5.2. Example 2.2 says that:

$$\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}} q$$

In order to prove this, we need to make use of the $\mathcal{KD}$ rule to show that $\mathcal{D}_{\{1,2\}}p$ and $\mathcal{D}_{\{1,2\}}(p \to q)$ both hold based on $\mathcal{K}_1 p$ and $\mathcal{K}_2(p \to q)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q)}\,\text{hyp}}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{K}_2(p \to q)}\,\wedge e2 \quad \dfrac{}{2 \in \{1,2\}}\,\in G}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{D}_{\{1,2\}}(p \to q)}\,\mathcal{KD}}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q), \boxed{\mathcal{D}_{\{1,2\}}} \vdash p \to q}\,\mathcal{D}e \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q)}\,\text{hyp}}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{K}_1 p}\,\wedge e1 \quad \dfrac{}{1 \in \{1,2\}}\,\in G}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{D}_{\{1,2\}} p}\,\mathcal{KD}}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q), \boxed{\mathcal{D}_{\{1,2\}}} \vdash p}\,\mathcal{D}e}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q), \boxed{\mathcal{D}_{\{1,2\}}} \vdash q}\,{\to}e}{\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \vdash \mathcal{D}_{\{1,2\}} q}\,\mathcal{D}i}{\vdash \mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \to \mathcal{D}_{\{1,2\}} q}\,{\to}i$$

With our implementation, the proof looks as follows in Coq:

```
Parameters p q: P.
Parameters i1 i2: A.

Theorem example2_2: Valid(
    K i1 p m/\  K i2 (p m-> q)
  m->
    D [i1, i2] q
).
Proof. mv.
mimp_i H.
mD_i.
mimp_e p.
+ mD_e [i1, i2] w.
  mKD i1.
  mcon_e1 (K i2 (p m-> q)).
  mhyp H.
+ mD_e [i1, i2] w.
  mKD i2.
  mcon_e2 (K i1 p).
  mhyp H.
Qed.
```

As you can see, we do not need to use the $\in G$ rule in our Coq proof, as Coq automatically applies our `minG` tactic when using the `mEK`, `mCK`, or `mKD` tactics, as we explained in Section 4.2.3.

5.3 Proof of Hakli and Negri's example

Hakli and Negri [7] have given an example that is slightly more complex than Example 2.2. In their example we have three agents: agent 1 knows that p holds, agent 2 knows that p implies q, and agent 3 knows that p and q together imply r. This should mean that it is part of the distributed knowledge of these three agents, that r must hold:

Theorem 5.3. Hakli and Negri's example says that:

$$\mathcal{K}_1 p \wedge \mathcal{K}_2(p \to q) \wedge \mathcal{K}_3(p \wedge q \to r) \to \mathcal{D}_{\{1,2,3\}} r$$

While we could construct a proof tree for this theorem, it is more convenient and less prone to errors to prove such formulas in Coq directly. For a larger formula like this one, the proof tree would also become fairly big, making it hard to read. So here we will only show the Coq proof:

```
Parameters p q r: P.
Parameters i1 i2 i3: A.

Theorem exampleHakliNegri: Valid(
    K i1 p m/\ K i2 (p m-> q) m/\ K i3 (p m/\ q m-> r)
  m->
    D [i1, i2, i3] r
).
Proof. mv.
mimp_i H.
mD_i.
mimp_e (p m/\ q).
+ mcon_i.
  * mD_e [i1, i2, i3] w.
    mKD i1.
    mcon_e1 (K i2 (p m-> q) m/\ K i3 (p m/\ q m-> r)).
    mhyp H.
  * mimp_e p.
    - mD_e [i1, i2, i3] w.
      mKD i1.
      mcon_e1 (K i2 (p m-> q) m/\ K i3 (p m/\ q m-> r)).
      mhyp H.
    - mD_e [i1, i2, i3] w.
      mKD i2.
      mcon_e1 (K i3 (p m/\ q m-> r)).
      mcon_e2 (K i1 p).
      mhyp H.
+ mD_e [i1, i2, i3] w.
  mKD i3.
  mcon_e2 (K i2 (p m-> q)).
  mcon_e2 (K i1 p).
  mhyp H.
Qed.
```

As you can see, we used similar tactics as for the proof of Example 2.2, but with a few more steps. Because Coq keeps track of everything that you still need to prove and the assumptions that you can use, proving a larger formula like this one is no problem at all in Coq.

5.4 Proof of the Overlapping Birthday Case

For our final example, we will have a look at the birthday case as discussed by Hakli and Negri [7], and also mentioned by e.g. van der Hoek [11]. The birthday case says that it is distributed knowledge of a group of agents whether or not two agents in that group have their birthday on the same date, assuming that every agent knows their own birthday.

Theorem 5.4. In predicate logic the birthday case can be described as follows:

$$\begin{aligned} &\mathcal{D}_G\left(\exists i \in G, \exists j \in G, \exists t : T, (B(i,t) \wedge B(j,t) \wedge i \neq j)\right) \\ \vee \\ &\mathcal{D}_G\left(\neg\exists i \in G, \exists j \in G, \exists t : T, (B(i,t) \wedge B(j,t) \wedge i \neq j)\right) \end{aligned}$$

Here T represents the days in a year, and $B(i,t)$ says that the birthday of agent i is on day t. It is either the case that there do exist two agents i and j in group G with overlapping birthdays, in which case that is part of the distributed knowledge of G, or it is the case that there do not exist two agents i and j in group G with overlapping birthdays, in which case that is distributed knowledge of G.

If we define a type `T` for days, and a function `B: A -> T -> P` for B, then we can define the assumption that every agent knows their own birthday as follows:

```
Parameter T : Type.
Parameter B: A -> T -> P.

Axiom KB: forall (i:A) (t:T) (w:W), (B i t w -> K i (B i t) w).
```

Here, the axiom `KB` that says that if the birthday of an agent `i` is on day `t`, then agent `i` knows that their birthday is on day `t`. As you can see, there is also a world `w` involved, this is because we are using lifted propositions. It could be that in a different world an agent has a different birthday.

However, because Hakli and Negri are using propositional logic (and we are too), they instead show that if for two arbitrary agents `i1` and `i2` their birthdays happen to be on the same arbitrary day `t`, then it is distributed knowledge for those two agents that they have the same birthday:

```
Parameters i1 i2: A.
Parameter t: T.

Theorem OverlappingBirthdaysSimplified: Valid(
    B i1 t m/\ B i2 t
  m->
    D [i1, i2] (B i1 t m/\ B i2 t)
).
```

We can prove this theorem by applying the `KB` axiom to prove that `K i1 (B i1 t) w` and `K i2 (B i2 t) w` hold after using the `KD` tactic, which allows us to prove that it is indeed distributed knowledge of two agents that their birthdays overlap, if they do:

```
Proof. mv.
mimp_i H.
mD_i.
mcon_i.
+ mD_e [i1, i2] w.
  mKD i1.
  apply KB.
  mcon_e1 (B i2 t).
  mhyp H.
+ mD_e [i1, i2] w.
  mKD i2.
  apply KB.
  mcon_e2 (B i1 t).
  mhyp H.
Qed.
```

But since our embedding is a shallow embedding, we can also use predicate logic in our proofs. Our implementation includes modal operators `mforall` and `mexists`, as defined by Benzmüller and Woltzenlogel Paleo [2], and our natural deduction tactics also include introduction and elimination rules for these operators, as seen in Appendix B.2.

To prove Theorem 5.4 with our implementation, we will also need a lifted proposition for group membership, since the modal operators only work for lifted propositions. Since groups are seen as constant and do not depend on any specific world, we can simply add an argument for a world `w` that does not get used:

```
Definition min (g:G) (i:A) (w:W) := g i.
```

Now we can extend the simplified version of the birthday case to say that if there exist two agents with an overlapping birthday in some arbitrary group `g`, then this is distributed knowledge for group `g`:

```
Parameter g: G.

Lemma BirthdaysDoOverlap: Valid(
    (mexists i1, mexists i2, mexists t,
       min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
  m->
    D g (mexists i1, mexists i2, mexists t,
          min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
).
```

To prove this, we need to use the $\exists i$ and $\exists e$ rules from predicate logic. Since our implementation is a shallow embedding, we can also use the default Coq rules in our proofs, so for convenience we have also used the `destruct` tactic. The proof can be found in Appendix C.

To prove the other part of Theorem 5.4, we also need to specify that every agent has only one birthday. We specify this by defining two axioms: one which says that for every agent there exists a day that is that agent's birthday, and one that says that if two days are both the birthday of an agent, then they must be the same day.

```
Axiom atLeastOneBirthday:
  forall (i:A) (w:W), exists (t:T), B i t w.
Axiom atMostOneBirthday:
  forall (i:A) (t1 t2:T) (w:W), ((B i t1 m/\ B i t2) w) -> t1 = t2.
```

Using these axioms we can prove the remaining of Theorem 5.4, which says that if there do not exist two agents with overlapping birthdays, then that is distributed knowledge:

```
Lemma BirthdaysDoNotOverlap: Valid(
    m~ (mexists i1, mexists i2, mexists t,
          min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
  m->
    D g (m~ (mexists i1, mexists i2, mexists t,
               min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t))
).
```

The proof for this part of the theorem is even longer, and can again be found in Appendix C.

Using the lemmas `BirthdaysDoOverlap` and `BirthdaysDoNotOverlap` we can fully prove Theorem 5.4 using the Law of Excluded Middle (`LEM`):

```
Theorem OverlappingBirthdaysFull: Valid(
    D g (mexists i1, mexists i2, mexists t,
           min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
  m\/
    D g (m~ (mexists i1, mexists i2, mexists t,
               min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t))
).
Proof. mv.
mimp_e (
    (mexists i1, mexists i2, mexists t,
       min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
  m\/
    (m~ (mexists i1, mexists i2, mexists t,
           min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t))
).
+ apply LEM.
+ mimp_i H.
  mdis_e (
      (mexists i1, mexists i2, mexists t,
         min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
    m\/
      (m~ (mexists i1, mexists i2, mexists t,
             min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t))
  ) w C1 C2.
  * mhyp H.
  * mdis_i1. apply BirthdaysDoOverlap. mhyp C1.
  * mdis_i2. apply BirthdaysDoNotOverlap. mhyp C2.
Qed.
```

Chapter 6

Related Work

There have been a variety of different approaches towards the proof theory of distributed knowledge. For example, Hakli and Negri [7] present proof theory for distributed knowledge within epistemic modal logic, using their modal sequent calculus system $\mathbf{G3KE_D}$.

An advantage of the sequent calculus systems $\mathbf{G3KE_D}$ is that the relations between different arbitrary worlds are clearly defined within the proof rules, instead of using the more abstract dashed boxes which we used for our $\mathbf{S5^n}$ proof rules. For example, Hakli and Negri define the proof rules $R\mathcal{D}_G$ and $L\mathcal{D}_G$, which are similar to our introduction and elimination rules $\mathcal{D}i$ and $\mathcal{D}e$:

Our proof rules	Hakli and Negri [7]'s proof rules
$\dfrac{\Gamma,\ \boxed{\mathcal{D}_G\ \vdash \varphi}}{\Gamma \vdash \mathcal{D}_G\varphi}\ \mathcal{D}i$	$\dfrac{\{x\, R_i\, y\}_{i\in G}, \Gamma \Rightarrow \Delta, y : \varphi}{\Gamma \Rightarrow \Delta, x : \mathcal{D}_G\varphi}\ R\mathcal{D}_G$
$\dfrac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma,\ \boxed{\mathcal{D}_G\ \overline{\Gamma} \vdash \varphi}}\ \mathcal{D}e$	$\dfrac{y : \varphi, x : \mathcal{D}_G\varphi, \{x\, R_i\, y\}_{i\in G}, \Gamma \Rightarrow \Delta}{x : \mathcal{D}_G\varphi, \{x\, R_i\, y\}_{i\in G}, \Gamma \Rightarrow \Delta}\ L\mathcal{D}_G$

As you can see, the Hakli and Negri [7]'s proof rules specify that a world y is related to world x by $\{x\, R_i\, y\}_{i\in G}$, by which they mean that $x\, R_i\, y$ must hold for every agent $i \in G$. This is similar to the $R_{\mathcal{D}_G}$ relation we defined in Definition 3.6, which is used internally in our $\mathcal{D}_G$-labeled dashed boxes.

Another approach to distributed knowledge is provided by Ghidini and Serafini [6], who present a context-based calculus for a logic they define called *Distributed First Order Logic*, which is used to reason about the *belief* of agents rather than the knowledge of agents. This logic is used to reason about subsystems in a distributed system, where knowledge can be shared between the various subsystems.

In this thesis, we instead provide a proof system for the epistemic modal logic $\mathbf{S5^n}$, which is largely based on Huth and Ryan [12], but using Gentzen-style proof trees. Note that while Huth and Ryan do provide proof rules for shared and common knowledge, they omit the proof theory for distributed knowledge for simplicity.

There also do already exist multiple Coq implementations of modal logic. For example, Sadrzadeh [18] presents an embedding of linear modal logic $\mathbf{KDT4_{lin}}$ in Coq. Furthermore, De Wind [3] and Lescanne [16] have both created embeddings of epistemic modal logic in Coq. An interesting aspect of De Wind's implementation is that she explicitly defines Kripke models in Coq, which means that it can be used to also prove formulas for specific Kripke models instead of being only able to reason about the validity of a formula in general. These embeddings focus mostly on common knowledge, which is used to for instance solve the Three Wise Men puzzle. Lescanne does also implement shared knowledge, however, neither De Wind, Lescanne, nor Sadrzadeh discuss distributed knowledge.

All of these previously mentioned Coq embeddings of modal logics are *deep embeddings*, which means that they use custom datatypes to represent modal formulas. Our implementation, based on Benzmüller and Woltzenlogel Paleo [2], is a *shallow embedding*, which, as explained in Chapter 4, means that we use and extend Coq's default predicate logic operators to construct modal formulas, which makes the embedding very expressive and easily extendable.

Benzmüller and Woltzenlogel Paleo [2] provide the necessity ($\Box$) and possibility ($\Diamond$) operators of the modal logic **S5**, as well as proof tactics for the introduction and elimination of these operators. We took this implementation as our starting point for our implementation since their necessity operator can relatively easily be adapted to become the knowledge operator $\mathcal{K}_i$ for multi-agent systems in $\mathbf{S5^n}$.

Chapter 7

Conclusions

We have discussed the modal logic $\mathbf{S5^n}$, based on Huth and Ryan [12], and we have shown how their proof rules can be adapted for Gentzen-style proofs. We have extended this proof theory with deduction rules for the distributed knowledge operator $\mathcal{D}_G$, which includes not only the introduction, elimination, and axiom-based rules, but also a useful $\mathcal{KD}$-rule, which allows you to prove that something is distributed knowledge in a group based on the knowledge of one particular agent in that group.

We have implemented the logic $\mathbf{S5^n}$ in the proof assistant Coq, based on an **S5** implementation by Benzmüller and Woltzenlogel Paleo [2], and we have implemented all the previously described deduction rules as proof tactics in Coq. Finally, we have shown some example proofs to showcase what our implementation is capable of, but mainly focusing on distributed knowledge.

Because our implementation is a shallow embedding, it not only works for propositional logic but also for predicate logic. And while we have provided proof tactics that can be used for formal natural deduction proofs, the default Coq tactics can also be used with our implementation.

The full implementation can be found in Appendix B, and the Coq code, including some extra examples, can be found online at `https://gitlab.science.ru.nl/mphilipse/coq-multi-agent-systems`.

Bibliography

[1] Robert J. Aumann. Backward induction and common knowledge of rationality. *Games and Economic Behavior*, 8(1):6–19, 1995.

[2] Christoph Benzmüller and Bruno Woltzenlogel Paleo. Interacting with Modal Logics in the Coq Proof Assistant. In Lev D. Beklemishev and Daniil V. Musatov, editors, *Computer Science – Theory and Applications*, pages 398–411, Cham, 2015. Springer International Publishing.

[3] Paulien de Wind. Modal Logic in Coq. Master's thesis, Vrije Universiteit Amsterdam, The Netherlands, May 2001.

[4] Ronald Fagin, Joseph Y. Halpern, and Moshe Y. Vardi. What Can Machines Know? On the Properties of Knowledge in Distributed Systems. *J. ACM*, 39(2):328–376, April 1992.

[5] Kirsten Foss and Nicolai Foss. Authority in the Context of Distributed Knowledge. DRUID Working Papers 03-08, DRUID, Copenhagen Business School, Department of Industrial Economics and Strategy/Aalborg University, Department of Business Studies, February 2003.

[6] Chiara Ghidini and Luciano Serafini. A Context-Based Logic for Distributed Knowledge Representation and Reasoning. In Paolo Bouquet, Massimo Benerecetti, Luciano Serafini, Patrick Brézillon, and Francesca Castellani, editors, *Modeling and Using Context*, pages 159–172, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[7] Raul Hakli and Sara Negri. Proof Theory for Distributed Knowledge. In Fariba Sadri and Ken Satoh, editors, *Computational Logic in Multi-Agent Systems*, pages 100–116, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Joseph Halpern, Yjoram Moses, and Mark Tuttle. A Knowledge-Based Analysis of Zero Knowledge. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 132–147, New York, NY, USA, 1988. Association for Computing Machinery.

[9] Joseph Y. Halpern and Yoram Moses. Knowledge and Common Knowledge in a Distributed Environment. *J. ACM*, 37(3):549–587, July 1990.

[10] Joseph Y. Halpern and Yoram Moses. A guide to completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[11] Wiebe van der Hoek. Logical Foundations of Agent-Based Computing. In *Selected Tutorial Papers from the 9th ECCAI Advanced Course ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School on Multi-Agent Systems and Applications*, EASSS '01, pages 50–73, Berlin, Heidelberg, 2001. Springer-Verlag.

[12] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.

[13] T. Imielinski. Relative Knowledge in a Distributed Database. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '87, pages 197–209, New York, NY, USA, 1987. Association for Computing Machinery.

[14] Cezary Kaliszyk, Femke van Raamsdonk, Freek Wiedijk, Hanno Wupper, Maxim Hendriks, and Roel de Vrijer. Deduction using the ProofWeb system. 2008.

[15] Leigh Lambert. Modal logic in computer science. Master's thesis, Rochester Institute of Technology, New York, United States, June 2004.

[16] Pierre Lescanne. Mechanizing common knowledge logic using Coq. In *Annals of Mathematics and Artificial Intelligence*, volume 48, pages 15–43, 2006.

[17] Stanley J. Rosenschein. Formal Theories of Knowledge in AI and Robotics. *New Generation Computing*, 3(4):345–357, October 1986.

[18] Mehrnoosh Sadrzadeh. Modal Linear Logic in Higher Order Logic, an experiment in Coq. In David Basin and Wolff Burkhart, editors, *Theorem Proving in Higher Order Logics*, pages 75–93, 2003.

[19] Josef Svenningsson and Emil Axelsson. Combining Deep and Shallow Embedding for EDSL. In Hans-Wolfgang Loidl and Ricardo Peña, editors, *Trends in Functional Programming - 13th International Symposium, TFP 2012, St. Andrews, UK, June 12-14, 2012, Revised Selected Papers*, volume 7829 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2012.

[20] The Coq Development Team. *The Coq Reference Manual, Release 8.11.0*. Inria, 2020.

[21] Marie-Jeanne Toussaint. Formal verification of probabilistic properties in cryptographic protocols. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 412–426, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[22] J.F.A.K. van Benthem, H.P. van Ditmarsch, J.S. Lodder, J. Ketting, and W.P.M. Meyer-Viol. *Logica voor Informatica*. Pearson Education Benelux, 3rd edition, 2003.

[23] Martin Wildmoser and Tobias Nipkow. Certifying Machine Code Safety: Shallow Versus Deep Embedding. In Konrad Slind, Annette Bunker, and Ganesh Gopalakrishnan, editors, *Theorem Proving in Higher Order Logics*, pages 305–320, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[24] Marianne Winslett. *Epistemic Aspects of Databases*, pages 133–174. Oxford University Press, Inc., USA, 1995.

Appendix A

Natural Deduction Rules

In this appendix we give an overview of the proof rules that we have used in this thesis. In Appendix A.1 you can see the deduction rules that we use for propositional logic, and in Appendix A.2 you can see the additional rules for the knowledge operators in $\mathbf{S5^n}$.

In these rules, the modal context Γ is defined as in Definition 3.9 for $\mathbf{S5^n}$, while the normal context $\overline{\Gamma}$ is defined as in Definition 3.3. The symbols φ, ψ, and ω represent propositions. In the rules for $\mathbf{S5^n}$, i represents an agent and G represents a group of agents.

A.1 Deduction rules for propositional logic

Here are the Gentzen-style natural deduction rules for propositional logic, from the (unpublished) coursed material of the course "Logic and Applications" by Hubbers, based on "Logica voor Informatica" by Van Benthem et al. [22]:

$$\frac{}{\Gamma, \varphi, \overline{\Gamma} \vdash \varphi}\ \text{hyp}$$

$$\frac{\Gamma \vdash \varphi \to \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ {\to}e \qquad\qquad \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \to \psi}\ {\to}i$$

$$\frac{\Gamma \vdash \neg\varphi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ \neg e \qquad\qquad \frac{\Gamma, \neg\psi \vdash \neg\varphi \qquad \Gamma, \neg\psi \vdash \varphi}{\Gamma \vdash \psi}\ \neg e{*}$$

$$\frac{\Gamma, \psi \vdash \neg\varphi \qquad \Gamma, \psi \vdash \varphi}{\Gamma \vdash \neg\psi}\ \neg i$$

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ {\wedge}e1 \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ {\wedge}e2 \qquad \frac{\Gamma \vdash \varphi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ {\wedge}i$$

$$\frac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma, \varphi \vdash \omega \qquad \Gamma, \psi \vdash \omega}{\Gamma \vdash \omega}\ {\vee}e$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}\ {\vee}i1 \qquad\qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}\ {\vee}i2$$

$$\frac{\Gamma \vdash \varphi \leftrightarrow \psi \qquad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\ {\leftrightarrow}e1 \qquad\qquad \frac{\Gamma \vdash \varphi \leftrightarrow \psi \qquad \Gamma \vdash \psi}{\Gamma \vdash \varphi}\ {\leftrightarrow}e2$$

$$\frac{\Gamma, \varphi \vdash \psi \qquad \Gamma, \psi \vdash \varphi}{\Gamma \vdash \varphi \leftrightarrow \psi}\ {\leftrightarrow}i$$

A.2 Additional rules for S5$^{\mathbf{n}}$

These are the additional proof rules for the knowledge operators of **S5$^{\mathbf{n}}$**:

$$\frac{\Gamma, \boxed{\mathcal{K}_i \ \vdash \varphi}}{\Gamma \vdash \mathcal{K}_i\varphi}\,\mathcal{K}i \qquad \frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma, \boxed{\mathcal{K}_i\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{K}e$$

$$\frac{\Gamma, \boxed{\mathcal{E}_G \ \vdash \varphi}}{\Gamma \vdash \mathcal{E}_G\varphi}\,\mathcal{E}i \qquad \frac{\Gamma \vdash \mathcal{E}_G\varphi}{\Gamma, \boxed{\mathcal{E}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{E}e$$

$$\frac{\Gamma, \boxed{\mathcal{C}_G \ \vdash \varphi}}{\Gamma \vdash \mathcal{C}_G\varphi}\,\mathcal{C}i \qquad \frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma, \boxed{\mathcal{C}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{C}e$$

$$\frac{\Gamma, \boxed{\mathcal{D}_G \ \vdash \varphi}}{\Gamma \vdash \mathcal{D}_G\varphi}\,\mathcal{D}i \qquad \frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma, \boxed{\mathcal{D}_G\ \overline{\Gamma} \vdash \varphi}}\,\mathcal{D}e$$

$$\frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma \vdash \varphi}\,\mathcal{K}T \qquad \frac{\Gamma \vdash \mathcal{K}_i\varphi}{\Gamma \vdash \mathcal{K}_i\mathcal{K}_i\varphi}\,\mathcal{K}4 \qquad \frac{\Gamma \vdash \neg\mathcal{K}_i\varphi}{\Gamma \vdash \mathcal{K}_i\neg\mathcal{K}_i\varphi}\,\mathcal{K}5$$

$$\frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{C}_G\mathcal{C}_G\varphi}\,\mathcal{C}4 \qquad \frac{\Gamma \vdash \neg\mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{C}_G\neg\mathcal{C}_G\varphi}\,\mathcal{C}5$$

$$\frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma \vdash \varphi}\,\mathcal{D}T \qquad \frac{\Gamma \vdash \mathcal{D}_G\varphi}{\Gamma \vdash \mathcal{D}_G\mathcal{D}_G\varphi}\,\mathcal{D}4 \qquad \frac{\Gamma \vdash \neg\mathcal{D}_G\varphi}{\Gamma \vdash \mathcal{D}_G\neg\mathcal{D}_G\varphi}\,\mathcal{D}5$$

$$\frac{\Gamma \vdash \mathcal{K}_i\varphi \text{ for each } i \in G}{\Gamma \vdash \mathcal{E}_G\varphi}\,\mathcal{K}\mathcal{E} \qquad \frac{\Gamma \vdash \mathcal{E}_G\varphi \qquad i \in G}{\Gamma \vdash \mathcal{K}_i\varphi}\,\mathcal{E}\mathcal{K}$$

$$\frac{\Gamma \vdash \mathcal{C}_G\varphi}{\Gamma \vdash \mathcal{E}_G \ldots \mathcal{E}_G\varphi}\,\mathcal{C}\mathcal{E} \qquad \frac{\Gamma \vdash \mathcal{C}_G\varphi \qquad i_j \in G}{\Gamma \vdash \mathcal{K}_{i_1} \ldots \mathcal{K}_{i_k}\varphi}\,\mathcal{C}\mathcal{K}$$

$$\frac{\Gamma \vdash \mathcal{K}_i\varphi \qquad i \in G}{\Gamma \vdash \mathcal{D}_G\varphi}\,\mathcal{K}\mathcal{D}$$

$$\frac{}{\Gamma \vdash i \in \{\ldots, i, \ldots\}}\,{\in}G$$

All of these rules, except those containing the distributed knowledge operator $\mathcal{D}_G$ and the rule ${\in}G$, are from the (unpublished) coursed material of the course "Logic and Applications" by Hubbers, based on Huth and Ryan [12].

Appendix B

Coq Implementation

B.1 $\mathbf{S5^n}$ in Coq

Here is our full Coq implementation of the multi-agent epistemic modal logic $\mathbf{S5^n}$, including the knowledge operators $\mathcal{K}_i$, $\mathcal{E}_G$, $\mathcal{C}_G$, and $\mathcal{D}_G$, and the proof rules from Appendix A.2. This implementation is based on the **S5** implementation by Benzmüller and Woltzenlogel Paleo [2].

```
(*
    Multi Agent Systems - S5^n Coq Implementation

    Special thanks to Engelbert Hubbers and Freek Wiedijk

    Based on Modal Logic implementation by Bruno Woltzenlogel Paleo and Christoph
     Benzmueller
*)


Parameter W: Type. (* Type for worlds *)
Parameter A: Type. (* Type for agents *)

Definition P := W -> Prop. (* Type of lifted propositions *)

Parameter R: A -> W -> W -> Prop. (* Accessibility relation *)

Definition G := A -> Prop. (* Groups of agents *)
(* Return True iff the agent is in the group *)


(* Notation of sets of agents, based on ListNotation *)
Notation "[ ]" := (fun i:A => False) (format "[ ]").
Notation "[ i1 ]" := (fun i:A => i=i1).
Notation "[ i1 , i2 , .. , iN ]" := (fun i:A => (i=i1 \/ (i=i2 \/ .. (i=iN \/ False)
      ..))).

Notation "'gforall'  i : g , p" := (forall i : A, g i -> p)
  (at level 200, i ident, right associativity) : type_scope.
Notation "'gexists'  i : g , p" := (exists i : A, g i /\ p)
  (at level 200, i ident, right associativity) : type_scope.

(* Tactic for proving that an agent i is in group g *)
```

```
Ltac minG :=
  match goal with
  | |- _ \/ _ => (left; reflexivity) || (right; minG)
  | |- _ = _  => reflexivity
  end ||
  fail "(could not find equal element in disjunction)".


(* Modal connectives *)
Definition mequal {A: Type}(x y: A)(w:W) := x = y.
Notation "x m= y" := (mequal x y) (at level 99, right associativity).

Definition mnot (p:P)(w:W) := ~ (p w).
Notation "m~ p" := (mnot p) (at level 74, right associativity).

Definition mand (p q:P)(w:W) := (p w) /\ (q w).
Notation "p m/\ q" := (mand p q) (at level 79, right associativity).

Definition mor (p q:P)(w:W) := (p w) \/ (q w).
Notation "p m\/ q" := (mor p q) (at level 79, right associativity).

Definition mimplies (p q:P)(w:W) := (p w) -> (q w).
Notation "p m-> q" := (mimplies p q) (at level 99, right associativity).

Definition mequiv (p q:P)(w:W) := (p w) <-> (q w).
Notation "p m<-> q" := (mequiv p q) (at level 99, right associativity).


(* Modal quantifiers *)
Definition mA {t:Type}(p:t -> P)(w:W) := forall x, p x w.
Notation "'mforall' x , p" := (mA (fun x => p))
  (at level 200, x ident, right associativity) : type_scope.
Notation "'mforall' x : t , p" := (mA (fun x:t => p))
  (at level 200, x ident, right associativity,
    format "'[' 'mforall' '/ ' x : t , '/ ' p ']'")
  : type_scope.

Definition mE {t:Type}(p:t -> P)(w:W) := exists x, p x w.
Notation "'mexists' x , p" := (mE (fun x => p))
  (at level 200, x ident, right associativity) : type_scope.
Notation "'mexists' x : t , p" := (mE (fun x:t => p))
  (at level 200, x ident, right associativity,
    format "'[' 'mexists' '/ ' x : t , '/ ' p ']'")
  : type_scope.


(* Knowledge operators *)
Definition K (i:A) (p:P) := fun x => forall y, (R i x y) -> (p y).
Definition E (g:G) (p:P) := fun x => gforall i : g, K i p x.
Fixpoint E' (n:nat) (g:G) (p:P) :=
  match n with
  | 0 => E g p
  | S n' => E' n' g (E g p)
end.
Definition C (g:G) (p:P) := fun x => forall n : nat, E' n g p x.
Definition D (g:G) (p:P) := fun x => forall y, (gforall i : g, R i x y) -> p y.


(* Modal validity of lifted propositions *)
Definition Valid (p:P) := forall w : W, p w.
Ltac mv := match goal with [|- (Valid _)] => intro end.
```

```
(* Accessibility relations R_E, R_D, and R_C (Huth & Ryan, page 338) *)
Definition RE (g:G) (x y:W) := gexists i : g, R i x y.
Fixpoint RE' (n:nat) (g:G) (x y:W) :=
  match n with
  | O => RE g x y
  | S n' => exists z:W, RE' n' g x z /\ RE g z y
end.
Definition RC (g:G) (x y:W) := exists n : nat, RE' n g x y.
Definition RD (g:G) (x y:W) := gforall i : g, R i x y.

(* RE accurately describes E *)
Lemma REE: forall g p x, (E g p x) <-> (forall y, (RE g x y) -> (p y)).
Proof.
intros.
unfold RE.
unfold E.
unfold K.
split.
+ intros.
  destruct H0 as [i H0].
  destruct H0.
  cut (R i x y).
  apply H.
  exact H0.
  exact H1.
+ intros.
  apply H.
  exists i.
  split.
  exact H0.
  exact H1.
Qed.

(* RE' accurately describes E' *)
Lemma RE'E': forall n g x p, (E' n g p x) <-> (forall y, (RE' n g x y) -> (p y)).
Proof.
intros n g x.
induction n.
+ split.
  * intros.
    unfold E' in H.
    unfold RE' in H0.
    cut (RE g x y).
    generalize y.
    apply REE.
    exact H.
    exact H0.
  * simpl.
    apply REE.
+ split.
  * simpl.
    intros.
    destruct H0 as [z H0].
    destruct H0.
    generalize (IHn (E g p)).
    intros.
    destruct H2.
    apply (proj1 (REE g p z)); auto.
  * simpl.
    intros.
    generalize (IHn (E g p)).
```

```
    intros.
    apply H0.
    intros.
    apply (REE g p y).
    intros z H2.
    apply H.
    exists y.
    split.
    exact H1.
    exact H2.
Qed.

(* RC accurately describes C *)
Lemma RCC: forall g x p, (C g p x) <-> (forall y, (RC g x y) -> (p y)).
Proof.
split.
+ intros H y H0.
  unfold RC in H0.
  destruct H0 as [n H0].
  cut (RE' n g x y).
  generalize y.
  apply RE'E'.
  unfold C in H.
  generalize n.
  exact H.
  exact H0.
+ intro H.
  unfold C.
  intro n.
  apply RE'E'.
  intros y H0.
  apply H.
  unfold RC.
  exists n.
  exact H0.
Qed.

(* RD accurately describes D *)
Lemma RDD: forall g p x, (D g p x) <-> (forall y, (RD g x y) -> (p y)).
Proof.
unfold D.
unfold RD.
split.
+ intros H y H0.
  apply H.
  exact H0.
+ intros H y H0.
  apply H.
  exact H0.
Qed.


(* Lemmas for proof tactics *)
Lemma Ke_lemma:
forall (i:A) (x:W) (p:P) (y:W),
  (K i p x) -> (R i x y) -> p y.
Proof.
intros i x p y H.
generalize y.
exact H.
Qed.
```

```
Lemma Ei_lemma:
forall (g:G) (x:W) (p:P),
  (forall y, (RE g x y) -> p y) -> (E g p x).
Proof.
intros g x p H.
apply REE.
exact H.
Qed.

Lemma Ee_lemma:
forall (g:G) (x:W) (p:P) (y:W),
  (E g p x) -> (RE g x y) -> p y.
Proof.
intros g x p y H.
generalize y.
apply (REE g p x).
exact H.
Qed.

Lemma Ci_lemma:
forall (g:G) (x:W) (p:P),
  (forall y, (RC g x y) -> p y) -> (C g p x).
Proof.
intros g x p H.
apply (RCC g x p).
exact H.
Qed.

Lemma Ce_lemma:
forall (g:G) (x:W) (p:P) (y:W),
  (C g p x) -> (RC g x y) -> p y.
Proof.
intros g x p y H.
generalize y.
apply (RCC g x p).
exact H.
Qed.

Lemma Di_lemma:
forall (g:G) (x:W) (p:P),
  (forall y, (RD g x y) -> p y) -> (D g p x).
Proof.
intros g x p H.
apply (RDD g p x).
exact H.
Qed.

Lemma De_lemma:
forall (g:G) (x:W) (p:P) (y:W),
  (D g p x) -> (RD g x y) -> p y.
Proof.
intros g x p y H.
generalize y.
apply (RDD g p x).
exact H.
Qed.

Lemma KE_lemma:
forall (g:G) (p:P) (x:W),
  (gforall i:g, K i p x) -> E g p x.
Proof.
intros g p x H.
```

```
exact H.
Qed.

Lemma EK_lemma:
forall (g:G) (p:P) (x:W),
  E g p x -> gforall i:g, K i p x.
Proof.
intros g p x H.
exact H.
Qed.

Lemma CE_lemma:
forall (g:G) (p:P) (x:W) (n:nat),
  C g p x -> E' n g p x.
Proof.
intros g p x n H.
generalize n.
exact H.
Qed.

Lemma KD_lemma:
forall (i:A) (g:G) (p:P) (x:W),
  g i -> K i p x -> D g p x.
Proof.
intros i g p x H H0.
unfold D.
intros y H1.
apply H0.
apply H1.
exact H.
Qed.


(* Convenient tactics for knowledge operators *)
Ltac mK_i := let w := fresh "w" in let R := fresh "R0"
             in (intro w at top; intro R).

Ltac mK_e i w := match goal with
  | |- _ ?w0 => apply (Ke_lemma i w); [> clear dependent w0 | try assumption]
end.

Ltac mE_i := let w := fresh "w" in let R := fresh "RE0"
             in (apply Ei_lemma; intro w at top; intro R).

Ltac mE_e g w := match goal with
  | |- _ ?w0 => apply (Ee_lemma g w); [> clear dependent w0 | try assumption]
end.

Ltac mC_i := let w := fresh "w" in let R := fresh "RC0"
             in (apply Ci_lemma; intro w at top; intro R).

Ltac mC_e g w := match goal with
  | |- _ ?w0 => apply (Ce_lemma g w); [> clear dependent w0 | try assumption]
end.

Ltac mD_i := let w := fresh "w" in let R := fresh "RD0"
             in (apply Di_lemma; intro w at top; intro R).

Ltac mD_e g w := match goal with
  | |- _ ?w0 => apply (De_lemma g w); [> clear dependent w0 | try assumption]
end.
```

```
Ltac mKE := let i := fresh "i" in let H := fresh "H"
            in apply KE_lemma; intros i H; repeat destruct H.

Ltac mEK g := apply EK_lemma with g; [> | try minG].

Ltac mCE := match goal with
  | |- E' ?n ?g' (E ?g' ?p') _ => unfold E'; fold (E' (S n) g' p'); mCE
  | |- E' ?n ?g' ?p' _  => apply CE_lemma
  | |- E ?g' (E ?g' ?p') _ => fold (E' 1 g' p'); mCE
  | |- E ?g' ?p' _ => fold (E' 0 g' p'); apply CE_lemma
end.

Ltac mCK' g:= match goal with
  | |- K ?i' (K _ _) ?w' => mK_i; mCK' g; only 1 : mK_e i' w'; only 1 : mEK g
  | |- K ?i' ?p' _ => mEK g
end.

Ltac mCK g := mCK' g; only 1 : mCE.

Ltac mKD i := apply KD_lemma with i; [> try minG | ].


(* Modal accessibility axioms *)
Axiom Rreflexivity: forall i x, R i x x.
Axiom Rtransitivity: forall i x y z, (R i x y) -> (R i y z) -> (R i x z).
Axiom Rsymmetry: forall i x y, (R i x y) -> (R i y x).
Theorem Reuclidean: forall i x y z, (R i x y) -> (R i x z) -> (R i y z).
Proof.
intros.
cut (R i y x).
+ intro.
  apply Rtransitivity with (y := x).
  exact H1.
  exact H0.
+ apply Rsymmetry.
  exact H.
Qed.


(* Modal axioms *)
Lemma KT_lemma: forall (w:W) (i:A) (p:P), K i p w -> p w.
Proof.
intros w i p.
intro H.
apply (Ke_lemma i w).
exact H.
apply Rreflexivity.
Qed.

Lemma K4_lemma: forall (w:W) (i:A) (p:P), K i p w -> K i (K i p) w.
Proof.
intros w i p H.
mK_i.
mK_i.
mK_e i w.
exact H.
apply Rtransitivity with (y := w0).
exact R0.
exact R1.
Qed.

Lemma K5_lemma: forall (w:W) (i:A) (p:P), (m~ K i p) w -> K i (m~ (K i p)) w.
```

```
Proof.
intros w i p H.
mK_i.
unfold mnot in *.
unfold not in *.
unfold K in *.
intro.
destruct H.
intros.
apply H0.
apply Reuclidean with (x := w).
exact R0.
exact H.
Qed.

Lemma RE'transitivity: forall n m g x y z, (RE' n g x y) -> (RE' m g y z) -> (RE' (1
     + n + m) g x z).
Proof.
intros n m i x y.
induction m.
+ intros.
  simpl.
  unfold RE' in H0.
  exists y.
  split.
  rewrite <- plus_n_O.
  exact H.
  exact H0.
+ intros.
  simpl.
  simpl in H0.
  destruct H0.
  destruct H0.
  exists x0.
  split.
  * assert (n + S m = 1 + n + m).
    - simpl.
      rewrite <- plus_n_Sm.
      reflexivity.
    - rewrite H2.
      apply IHm.
      exact H.
      exact H0.
  * exact H1.
Qed.

Lemma RE'reverse: forall n g x y, (RE' (S n) g x y) -> (exists z, RE g x z /\ RE' n
    g z y).
Proof.
intros n g.
induction n.
+ intros.
  simpl in H.
  destruct H.
  destruct H.
  exists x0.
  split.
  exact H.
  exact H0.
+ intros.
  simpl in H.
  destruct H.
```

```
  destruct H.
  destruct H.
  destruct H.
  destruct IHn with x x0.
  * simpl.
    exists x1.
    split.
    exact H.
    exact H1.
  * destruct H2.
    exists x2.
    split.
    exact H2.
    simpl.
    exists x0.
    split.
    exact H3.
    exact H0.
Qed.

Lemma RE'symmetry: forall n g x y, (RE' n g x y) -> (RE' n g y x).
Proof.
intros n g.
induction n.
+ intros.
  simpl.
  unfold RE' in H.
  unfold RE.
  unfold RE in H.
  destruct H.
  destruct H.
  exists x0.
  split.
  exact H.
  apply Rsymmetry.
  exact H0.
+ intros.
  apply RE'reverse in H.
  destruct H.
  destruct H.
  simpl.
  exists x0.
  split.
  * apply IHn.
    exact H0.
  * unfold RE.
    unfold RE in H.
    destruct H.
    destruct H.
    exists x1.
    split.
    exact H.
    apply Rsymmetry.
    exact H1.
Qed.

Lemma RE'Euclidean: forall n m g x y z, (RE' n g x y) -> (RE' m g x z) -> (RE' (1 +
     n + m) g y z).
Proof.
intros n m g x y.
intros.
apply RE'transitivity with x.
```

```
+ apply RE'symmetry.
  exact H.
+ exact H0.
Qed.

Lemma RCtransitivity: forall g x y z, (RC g x y) -> (RC g y z) -> (RC g x z).
Proof.
intros g x y z H H0.
unfold RC in *.
destruct H.
destruct H0.
exists (1 + x0 + x1).
apply RE'transitivity with (y:=y).
exact H.
exact H0.
Qed.

Lemma RCEuclidean: forall g x y z, (RC g x y) -> (RC g x z) -> (RC g y z).
Proof.
intros g x y z H H0.
unfold RC in *.
destruct H.
destruct H0.
exists (1 + x0 + x1).
apply RE'Euclidean with (x:=x).
exact H.
exact H0.
Qed.

Lemma C4_lemma: forall (w:W) (g:G) (p:P), C g p w -> C g (C g p) w.
Proof.
intros w g p H.
mC_i.
mC_i.
apply (proj1 (RCC g w p)) with (y:=w1) in H .
exact H.
apply RCtransitivity with w0.
exact RC0.
exact RC1.
Qed.

Lemma C5_lemma: forall (w:W) (g:G) (p:P), (m~ (C g p)) w -> C g (m~ (C g p)) w.
Proof.
intros w g p H.
mC_i.
unfold mnot in *.
unfold not in *.
intro H0.
apply H.
apply RCC.
intros w1 RC1.
apply (proj1 (RCC g w0 p)).
exact H0.
apply RCEuclidean with (x:=w).
exact RC0.
exact RC1.
Qed.

Lemma DT_lemma: forall (w:W) (g:G) (p:P), D g p w -> p w.
Proof.
intros w g p H.
unfold D in H.
```

```
apply H.
intros.
apply Rreflexivity.
Qed.

Lemma D4_lemma: forall (w:W) (g:G) (p:P), D g p w -> D g (D g p) w.
Proof.
intros w g p H.
mD_i.
mD_i.
unfold RD in *.
unfold D in H.
apply H.
intros.
apply Rtransitivity with (y:=w0).
+ cut (g i).
  generalize i.
  exact RD0.
  exact H0.
+ cut (g i).
  generalize i.
  exact RD1.
  exact H0.
Qed.

Lemma D5_lemma: forall (w:W) (g:G) (p:P), (m~ (D g p)) w -> D g (m~ (D g p)) w.
Proof.
intros w g p H.
mD_i.
unfold mnot in *.
unfold RD in RD0.
unfold not in *.
unfold D in *.
intro H0.
apply H.
intros y H1.
apply H0.
intros.
apply Reuclidean with (x:=w).
+ cut (g i).
  generalize i.
  exact RD0.
  exact H2.
+ cut (g i).
  generalize i.
  exact H1.
  exact H2.
Qed.

Ltac mKT i := apply KT_lemma with i.
Ltac mK4 := apply K4_lemma.
Ltac mK5 := apply K5_lemma.
Ltac mC4 := apply C4_lemma.
Ltac mC5 := apply C5_lemma.
Ltac mDT g := apply DT_lemma with g.
Ltac mD4 := apply D4_lemma.
Ltac mD5 := apply D5_lemma.

Create HintDb modal.
Hint Unfold mimplies mnot mor mand K E C D mA mE Valid : modal.
```

B.2 Natural deduction rules in Coq

Here are the proof tactics corresponding to the natural deduction rules from Appendix A.1. These rules are based on ProofWeb [14], but they have been modified to fit the modal operators from Appendix B.1.

```
(*
    Natural deduction proof rules for Modal.v

    Based on ProofWeb.v
*)

Require Import Modal.
Require Import Classical.

Ltac mcon_i :=
  match goal with
  | |- (_ m/\ _) _ => split
  end ||
  fail "(the goal is not a conjunction)".

Lemma mcone1:
forall right left w,
  (left m/\ right) w -> left w.
Proof.
unfold "m/\ ".
intros r l w H.
destruct H.
assumption.
Qed.

Ltac mcon_e1 R := apply (mcone1 R).

Lemma mcone2:
forall left right w,
  (left m/\ right) w -> right w.
Proof.
unfold "m/\ ".
intros r l w H.
destruct H.
assumption.
Qed.

Ltac mcon_e2 L := apply (mcone2 L).

Ltac mdis_i1 :=
  match goal with
  | |- (_ m\/ _) _ => left
  end ||
  fail "(the goal is not a disjunction)".

Ltac mdis_i2 :=
  match goal with
  | |- (_ m\/ _) _ => right
  end ||
  fail "(the goal is not a disjunction)".

Ltac mdis_e X w H1 H2 :=
  match X with
  | (_ m\/ _) =>
    let x := fresh "H" in
```

```
    assert (x : (X w));
    [ idtac | elim x; clear x; [intro H1 | intro H2] ]
  end ||
  fail "(the argument is not a disjunction or the labels already exist)".

Ltac mimp_i X :=
  match goal with
  | |- (_ m-> _) _ => intro X
  end ||
  fail "(the goal is not an implication)".

Lemma mimpe:
forall (prem conc:P) (w:W),
  (prem w) -> ((prem m-> conc) w) -> (conc w).
Proof.
unfold "m->".
intros p c w H H0.
apply H0.
assumption.
Qed.

Ltac mimp_e X := apply (mimpe X).

Ltac miff_i H1 H2 :=
 match goal with
 | |- (_ m<-> _) _ =>
 split; [ (intro H1) | (intro H2) ]
 end ||
 fail "(the goal is not a bi-implication)".

Lemma miffe1:
forall (L R:P) (w:W),
  (L w) -> ((L m<->R) w) -> R w.
Proof.
intros L R w H H0.
apply H0.
exact H.
Qed.

Ltac miff_e1 L := apply miffe1 with L.

Lemma miffe2:
forall (L R:P) (w:W),
  (R w) -> ((L m<->R) w) -> L w.
Proof.
intros L R w H H0.
apply H0.
exact H.
Qed.

Ltac miff_e2 R := apply miffe2 with R.

Ltac mall_i X :=
  match goal with
  | |- (mforall _, _) _ => intro X
  end ||
  fail "(the goal is not a universal quantification)".

Ltac mall_e X w A :=
  match X with
  | (mforall _ : _ , _) => refine ((_ : X w) A)
  end ||
```

```
  fail "(the argument is not a universal quantification)".

Ltac mexi_i X :=
  match goal with
  | |- (mexists x : _ , _) _ => exists X
  end ||
  fail "(the goal is not an existential quantification)".

Ltac mexi_e X w a H :=
  match X with
  | (mexists x : _ , _) => refine ((fun x y => ex_ind y (x : X w)) _ _); [idtac |
     intros a H]
  end ||
  fail "(the argument is not an existential quantification)".

Ltac mabsurd X w := absurd (X w); [
    match goal with
      | |- ~ (?p') ?w' => fold ((m~ p') w')
    end
  |].

Ltac mneg_i X w Y :=
  match goal with
  | |- (m~ _) _ => intro Y; mabsurd X w
  end ||
  fail "(the goal is not a negation)".

Ltac mneg_e X w := mabsurd X w.

Ltac mnegneg_e :=
  match goal with
  | |- ?H => apply (NNPP H)
  end.

Ltac mneg_e' X w Y := mnegneg_e; intro Y; mabsurd X w.

Ltac mhyp X := exact X.
```

Appendix C

Overlapping Birthday Case

Here are the complete proofs of both the simplified and the original version of the overlapping birthday case, as discussed in Section 5.4:

```
Require Import Modal.
Require Import ModalRules.

(* Proof based on the birthday case, as discussed by Hakli and Negri *)


(* Type T for days *)
Parameter T : Type.

(* Birthday of agent i is on day n *)
Parameter B: A -> T -> P.

(* Every agent knows their own birthday *)
Axiom KB: forall (i:A) (t:T) (w:W), (B i t w -> K i (B i t) w).


Section Birthday_Case_Simplification.

(* Some arbitrary day t *)
Parameter t: T.

(* Some arbitrary agents i1 and i2 *)
Parameters i1 i2: A.

Theorem OverlappingBirthdaysSimplified: Valid(
    B i1 t m/\ B i2 t
  m->
    D [i1, i2] (B i1 t m/\ B i2 t)
).
Proof. mv.
mimp_i H.
mD_i.
mcon_i.
+ mD_e [i1, i2] w.
  mKD i1.
  apply KB.
  mcon_e1 (B i2 t).
  mhyp H.
+ mD_e [i1, i2] w.
```

```
  mKD i2.
  apply KB.
  mcon_e2 (B i1 t).
  mhyp H.
Qed.

Reset Birthday_Case_Simplification.


(* Some arbitrary group g *)
Parameter g: G.

(* Lifted proposition for group membership *)
Definition min (g:G) (i:A) (w:W) := g i.

Lemma BirthdaysDoOverlap: Valid(
    (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t
     )
  m->
    D g (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B
     i2 t)
).
Proof. mv.
mimp_i H.
mexi_e (mexists i1 : A, mexists i2 : A, mexists t : T, min g i1 m/\ min g i2 m/\ B
     i1 t m/\ B i2 t) w i1 H1.
mhyp H.
mexi_e (mexists i2 : A, mexists t : T, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t)
      w i2 H2.
mhyp H1.
mexi_e (mexists t : T, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t) w t H3.
mhyp H2.
destruct H3. (* We use some default Coq tactics for convenience *)
destruct H3.
destruct H4.
mD_i.
mexi_i i1.
mexi_i i2.
mexi_i t.
mcon_i.
mhyp H0.
mcon_i.
mhyp H3.
mcon_i.
+ mD_e g w.
  mKD i1.
  mhyp H0.
  apply KB.
  mhyp H4.
+ mD_e g w.
  mKD i2.
  mhyp H3.
  apply KB.
  mhyp H5.
Qed.

(* Every agent has a birthday *)
Axiom atLeastOneBirthday: forall (i:A) (w:W), exists (t:T), B i t w.

(* Every agent has at most one birthday *)
Axiom atMostOneBirthday: forall (i:A) (t1 t2:T) (w:W), ((B i t1 m/\ B i t2) w) -> t1
      = t2.
```

```
Lemma BirthdaysDoNotOverlap: Valid(
    m~ (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B
     i2 t)
  m->
    D g (m~ (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t
     m/\ B i2 t))
).
Proof. mv.
mimp_i H.

mimp_e (
  mforall t1, mforall t2, mforall i1, mforall i2,
    min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m=t2)
).
+ mall_i t1.
  mall_i t2.
  mall_i i1.
  mall_i i2.
  mimp_i H0.
  destruct H0.
  destruct H1.
  destruct H2.
  mneg_i (
    mexists i1 : A, mexists i2 : A, mexists t : T, min g i1 m/\ min g i2 m/\ B i1 t
     m/\ B i2 t
  ) w H4.
  mhyp H.
  mexi_i i1.
  mexi_i i2.
  mexi_i t1.
  mcon_i.
  mhyp H0.
  mcon_i.
  mhyp H1.
  mcon_i.
  mhyp H2.
  replace t1 with t2.
  mhyp H3.
+ mimp_i H0.
  mD_i.
  mneg_i (mforall t1 : T, mforall t2 : T, mforall i1 : A, mforall i2 : A,
            min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m= t2)) w N1.
  * mexi_e (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\
     B i2 t) w0 i1 N2.
    mhyp N1.
    mexi_e (mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t) w0
     i2 N3.
    mhyp N2.
    mexi_e (mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t) w0 t N4.
    mhyp N3.
    destruct N4.
    destruct H2.
    destruct H3.

    mimp_e (mexists t1, B i1 t1).
    apply atLeastOneBirthday.
    mimp_i B1.
    mexi_e (mexists t1 : T, B i1 t1) w t1 B1a.
    mhyp B1.

    mimp_e (mexists t2, B i2 t2).
```

```
    apply atLeastOneBirthday.
    mimp_i B2.
    mexi_e (mexists t2 : T, B i2 t2) w t2 B2a.
    mhyp B2.

    mneg_e (t1 m= t2) w.
    - mimp_e (min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2).
      ++ mcon_i.
         mhyp H1. (* Groups do not depend on worlds *)
         mcon_i.
         mhyp H2.
         mcon_i.
         mhyp B1a.
         mhyp B2a.
      ++ mall_e (mforall i2 : A,
                 min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m= t2)) w
      i2.
         mall_e (mforall i1 : A, mforall i2 : A,
                 min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m= t2)) w
      i1.
         mall_e (mforall t2 : T, mforall i1 : A, mforall i2 : A,
                 min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m= t2)) w
      t2.
         mall_e (mforall t1 : T, mforall t2 : T, mforall i1 : A, mforall i2 : A,
                 min g i1 m/\ min g i2 m/\ B i1 t1 m/\ B i2 t2 m-> m~ (t1 m= t2)) w
      t1.
         mhyp H0.
    - mimp_e (min g i1). (* Convert knowledge of group membership i1 and i2 to world
      w *)
      mhyp H1.
      mimp_i H5.
      mimp_e (min g i2).
      mhyp H2.
      mimp_i H6.
      replace t1 with t.
      replace t2 with t.
      ++ reflexivity.
      ++ apply atMostOneBirthday with i2 w0.
         mcon_i.
         mhyp H4.
         mD_e g w.
         mKD i2.
         mhyp H6.
         apply KB.
         mhyp B2a.
      ++ apply atMostOneBirthday with i1 w0.
         mcon_i.
         mhyp H3.
         mD_e g w.
         mKD i1.
         mhyp H5.
         apply KB.
         mhyp B1a.
 * mhyp H0.
Qed.

Lemma LEM: forall (p:P) (w:W), (p m\/ m~ p) w.
Proof.
intros p w.
mneg_e' (m~ (p m\/ m~ p)) w H.
+ mneg_i (p m\/ m~ p) w H0.
  mhyp H0.
```

```
  mdis_i2.
  mneg_i (p m\/ m~ p) w H1.
  mhyp H0.
  mdis_i1.
  mhyp H1.
+ mhyp H.
Qed.

Theorem OverlappingBirthdaysFull: Valid(
    D g (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B
     i2 t)
  m\/
    D g (m~ (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t
     m/\ B i2 t))
).
Proof. mv.
mimp_e (
    (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2 t
     )
  m\/
    (m~ (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B
     i2 t))
).
+ apply LEM.
+ mimp_i H.
  mdis_e (
      (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\ B i2
      t)
    m\/
      (m~ (mexists i1, mexists i2, mexists t, min g i1 m/\ min g i2 m/\ B i1 t m/\
     B i2 t))
  ) w C1 C2.
  * mhyp H.
  * mdis_i1.
    apply BirthdaysDoOverlap.
    mhyp C1.
  * mdis_i2.
    apply BirthdaysDoNotOverlap.
    mhyp C2.
Qed.
```