New Directions in Model Checking Dynamic Epistemic Logic

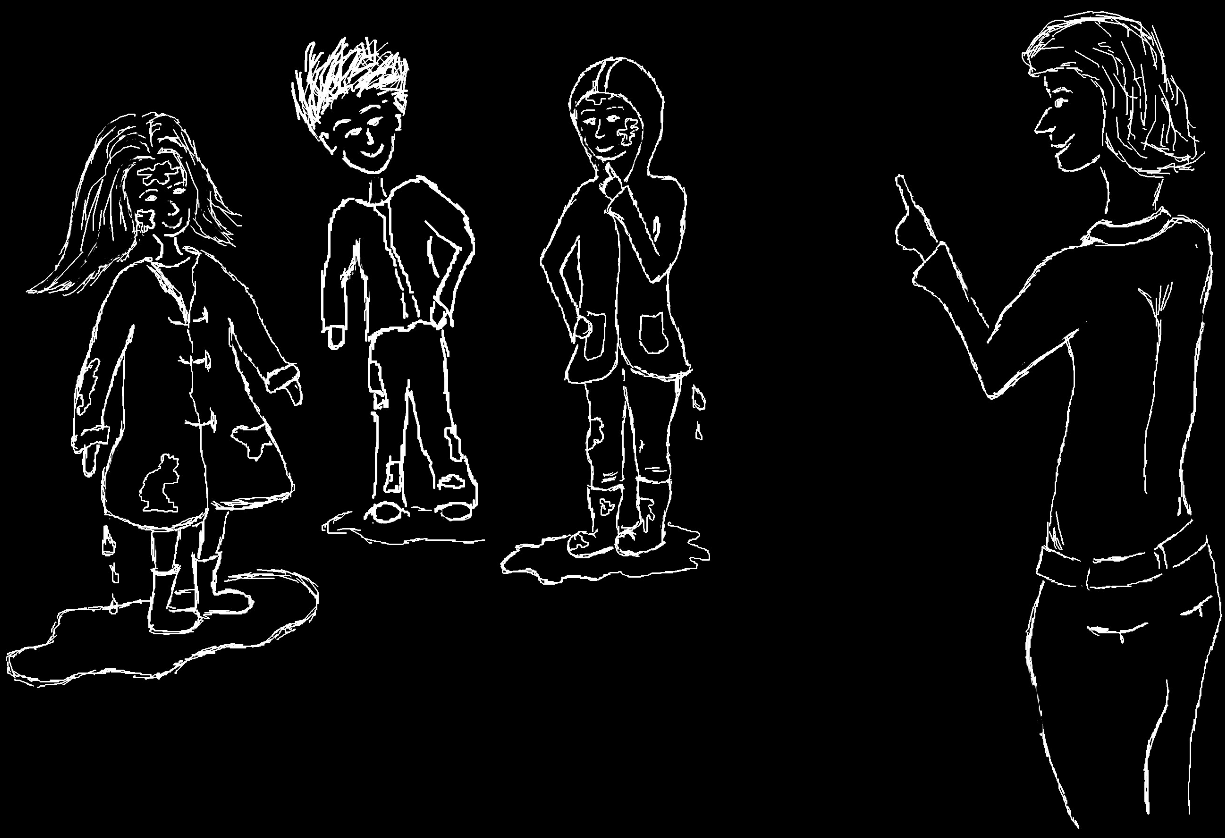
Gattinger, M.

Publication date
2018

Document Version
Final published version

License
Other

New Directions in Model Checking Dynamic Epistemic Logic



Malvin Gattinger

New Directions in Model Checking Dynamic Epistemic Logic

Malvin Gattinger

New Directions in Model Checking Dynamic Epistemic Logic

ILLC Dissertation Series DS-2018-11

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

For further information about ILLC-publications, please contact

Institute for Logic, Language and Computation
Universiteit van Amsterdam
Science Park 107
1098 XG Amsterdam
phone: +31-20-525 6051
e-mail: illc@uva.nl
homepage: https://www.illc.uva.nl/

This work was financially supported and hosted by the Tsinghua-UvA Joint Research Center for Logic. The author also received financial travel support from the Australian Research Council Future Fellowship FT0991785 and the European Research Council Starting Grant *Epistemic Protocol Synthesis* 313360.

Cover drawings by Sarah Vollert.

Printed and bound by Ipskamp Printing.

ISBN: 978–94–028–1025–7

New Directions in Model Checking Dynamic Epistemic Logic

Academisch Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. K.I.J. Maex
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op woensdag 13 juni 2018, te 11.00 uur

door

Benjamin Rene Malvin Gattinger

geboren te Offenbach am Main, Duitsland

Promotiecommisie

Promotor:	Prof. dr. D.J.N. van Eijck	Universiteit van Amsterdam
Co-promotor:	Dr. A. Baltag	Universiteit van Amsterdam
	Prof. dr. K. Su	Griffith University
Overige leden:	Prof. dr. J.F.A.K. van Benthem	Universiteit van Amsterdam
	Dr. A. Herzig	Université Paul-Sabatier
	Dr. C. Schaffner	Universiteit van Amsterdam
	Prof. dr. F.Liu	Tsinghua University
	Prof. dr. F.J.M.M. Veltman	Universiteit van Amsterdam
	Prof. dr. Y. Venema	Universiteit van Amsterdam
	Dr. Y.Wang	Peking University

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Contents

Acknowledgments

Throughout this thesis we do our best to avoid long lists. It is my pleasure to begin with an exception to this rule and thank those who helped me.

Dear Jan van Eijck, thank you for encouraging me to pursue a PhD and for being the most relaxed and relaxing supervisor, again. It was great to have you on my side for these four years. I will remember your office as a magic place where logic, computer science and philosophy happily meet each other. You gave me plenty of advice that I will not forget. Whenever I get frustrated I will now look deep into myself and deep into the reviews — in this order. Finally, thank you for still being available and helping me finish this adventure after your retirement.

苏开乐, thank you for being my co-supervisor, for laying the groundwork on symbolic model checking for Public Announcement Logic and for inviting me to Brisbane where you gave me the time and the freedom to invent knowledge transformers and to hug a koala.

My gratitude also goes to Alexandru Baltag: Thank you for being my co-supervisor and for very helpful comments and complaints about drafts of this thesis. The ideas for future work which you scribbled into my margins will keep me busy for a long time — and that was before I realized that the official ILLC style demands even bigger margins. I already know that I will miss the LIRa seminar and the drinks with your great stories and life lessons. Most of all, thank you for spreading your passion for logic and science in general!

Besides my three official supervisors, I want to thank Hans van Ditmarsch without whom this thesis would not be what it is. Thank you Hans, for establishing logic puzzles and card games as a serious research area, for inviting me to Nancy to study dynamic gossip on whiteboards and cycle-paths, and last but not least for being yourself and a great role model.

I want to thank Johan van Benthem for both the foundations and landmarks in our field, for building an international community I am proud to be part of, for his incredibly detailed feedback emails after listening to my talks and for joining my defense committee.

I am also happy to have the following people on my committee: Andreas Herzig, who already gave me helpful feedback at LORI 2015 and ESSLLI 2017, Christian Schaffner, for whom being a teaching assistant in Cryptography was really fun, Yde Venema, who showed me the joy of research already during the master and who joined my committee on very short notice between Glühwein and oliebollen, and Frank Veltman, who was the first to teach me Basic Logic properly.

Peter van Emde Boas deserves my thanks for coming to many of my talks with rigorous questions and for being the chair of my defense committee.

These four years have been extra exciting because my position was part of the Tsinghua-UvA Joint Research Centre for Logic. I am grateful for three trips to China during which I learned much more than just logic. Among all the wonderful people I met because of the Joint Research Centre, I especially want to thank 刘奋荣 for the invitations to Tsinghua University and for joining my defense committee, 王彦晶 for being one of my favorite collaborators that always made the right choices (including the decision to withdraw one of our papers), for inviting me to Peking University, and for being part of my committee, and 石辰威 for being a great flatmate, colleague and excellent travel guide.

I would also like to thank my other co-authors for the joy of working with them and their permission to include our work here: Rahim Ramezanian, the months we spent looking at triangles to axiomatize Propositional Dynamic Gossip Logic were not in vain — at least I had fun working with you, Louwe B. Kuijer, thank you for catching many of my flawed ideas with your incredible precision and kindness, and Pere Pardo, your care and accuracy were truly helpful.

Any science is team work, even if you do not actually work on the same topics. I will miss my office colleagues, namely Giovanni Cinà, Dieuwke Hupkes and Jouke Witteveen, who were all excellent partners in crime.

Facundo Carreiro and Sumit Sourabh deserve my gratitude because they gave me a somewhat realistic idea of what doing a PhD would be like — also in hindsight.

Nadine Theiler, thank you for staying in Amsterdam when most of our MoL year was moving (far) away, for letting me stay at your place when I was a homeless cyclist, and for many Cineville evenings.

It is common knowledge among PhD students that the challenging part is not to write a thesis but to deal with the bureaucracy. The ILLC office has been indispensable in navigating this chaos. In particular I want to thank Jenny Batsoen, Peter van Ormondt, Debbie Klaassen, and the all-knowing and always helpful Tanja Kassenaar.

Many more people helped me whom I do not know enough to properly thank them. For example, let me thank all the people taking care of the ILLC, NIKHEF and CWI buildings. It is a privilege to have clean offices, managed coffee machines and fast internet. Though it really is time for IPv6 by now, dear UvA.

There is a lot of free/libre open source software without which my research would have been impossible. I would like to thank the authors of the Glorious

Glasgow Haskell Compilation System (GHC) and many Haskell libraries, Debian GNU/Linux, KDE, Firefox, Thunderbird, Nextcloud, Graphviz, LaTeX, Pandoc, Atom, and other software of which I might not even know that I have been using it. Additionally, I am grateful to several small groups of idealists providing online services like disroot.org. For technical help, I owe thanks to many geniuses around the world who I only know by their nicknames on the `#haskell` IRC channel.

I had the privilege of traveling a lot during these four years. My cycling trips to summer schools were even more fun thanks to the warmshowers.org community. As Hans will confirm, cycling can yield new proof ideas which then have to be written down and shared immediately. In these cases eduroam was of great use, wherever I happened to be. Apropos traveling, I would also like to thank Nederlandse Spoorwegen and Deutsche Bahn — especially the teams on board the ICE International. Special thanks also go to the KLM travel clinic and to the train schedule oracle at Delhi central station.

Back home in various senses, I want to thank Pauline Fleischmann, who often knows me better than I do myself, Laura Stumpp, who has an excellent conspiracy theory why I study the gossip problem, and Hannah Weisbach, who was an unexpected but great flatmate.

I am indebted to Sarah Vollert for the cover drawings which she managed to produce in almost no time and based on very vague and chaotic instructions.

My biggest thanks go to Emma Brakkee. Thank you Emma, for being a wonderful companion in this crazy world in general and the academic world in particular. I cannot even count the times you proofread for me or listened to my worries about mistakes and deadlines. Fortunately, I can also share the joy of new proofs and ideas with you, in a growing number of languages.

I first met my two paranymphs Jana Wagemaker and Esteban Landerreche when they were among my favorite students in various courses and am truly happy that they became friends and colleagues shortly after.

Emma, Jana and Esteban deserve not only my gratitude, but also that of anyone who reads this work: They turned around every comma in this thesis and kept me from writing it essentially in Germanglish by nesting way too many colons and dashes in one-sentence paragraphs — almost. I also want to thank Bastian Reitemeier and David Julian Veenstra for reading parts of my thesis. Of course, all the remaining mistakes are my fault.

Finally, my deepest gratitude goes to Pia, Hans and Anne Gattinger. Dear Pia and Hans, neither this thesis nor I would exist without you. I am proud to be your son and grateful for your continuous support in all ways. Dear Anne, thank you for being the best sister and for hosting me when I urgently needed some mountains after all this.

Malvin Gattinger
Amsterdam, April 2018

Introduction

Either a thing is true or it isn't.
If it is true, you should believe it.
And if it isn't, you shouldn't.

Bertrand Russell

Computers are an essential part of modern life: Phones, watches, cars, planes and bicycles are only a small selection of items that nowadays routinely have more computational power than the whole Apollo team had available when it flew to the moon. People trust computers on a daily basis to deliver messages and pictures, to organize their calendars, to deliver the news, to manage their money, to help them find the way, or to translate between different languages.

How can we be sure these devices do what we want them to do? Trusting a device means trusting those who made it.[1] Additionally, it means trusting that they did not make any mistake. A tragic case of trust in a machine was the *Therac-25*, a particle accelerator used in radiation therapy. Due to a concurrency related bug in its software, six patients died after being overdosed [LT93]. How can such problems be avoided? How can we be sure that a program will behave in the way we want?

Formal methods such as model checking are an answer to this question: We can use formal languages to specify what our systems should do and then check if a program, a circuit, or a model thereof fulfills this specification. The strength of this approach is that such checks can themselves be done automatically, by computers. One is always testing (at least) three things, all against each other: The specification, the model and the tool used to compare them.

Methods for verification were studied already in the 1960s and 1970s, but most of them were based on theorem proving [Eme08]. A major breakthrough in the 1980s was the advent of *symbolic* methods for model checking that no longer need

[1] Given the recent revelations about surveillance and almost regular security problems affecting the whole internet, this is already a big leap of faith, but not the topic of this thesis.

to spell out large models explicitly to check them [Bur+90]. The success of model checking in practice ranges from finding problems in formal protocols to improving real-time systems ensuring the stability of buildings during earthquakes [CW96]. By now, model checking and other forms of verification are industry standards and will hopefully prevent tragedies like the *Therac-25* in the future.

Given this success, it is natural to ask where else model checking can be applied, and in order to do so, which kinds of systems and problems can be modeled and checked in formal languages. Most existing model checking tools work with a variant of temporal logic. One way to extend these logics is to add *epistemic* operators, allowing us to express situations or problems involving knowledge or belief. Consider for example the following puzzle:

> Three cryptographers go out to have dinner. After a delicious meal the waiter tells them that the bill has already been paid. The cryptographers know that either one of them paid, or the National Security Agency (NSA). They want to find out which of the two is the case but also respect the wish to stay anonymous: If one of them paid they do not want that person to be revealed. What should they do? And suppose they find a procedure, how can they verify that it works?

Dynamic Epistemic Logic (DEL) is a logic developed during the last twenty years which can easily describe this sort of scenario. However, so far no symbolic implementation of it existed and while computers can easily deal with small toy examples like the above, as soon as the number of agents — in this case dining cryptographers — becomes larger, the models no longer fit into memory and we need better methods. This leads us to the main research question of this thesis.

> Research Question 1.
> *Can we find symbolic model checking methods for DEL?*

To answer this question we need to find symbolic equivalents of the models which are used in the standard semantics of DEL, namely Kripke models and action models. Symbolic representations for Kripke models have already been developed for temporal logics. This thesis essentially provides a general method to import these techniques to DEL. Given a theoretical answer to our first research question, we also want to know how usable it is in practice.

> Research Question 2.
> *How can symbolic model checking for DEL be implemented?*

An implementation and its use are not only a goal of our research, but at the same time a source of inspiration and corrective feedback. Any logicians who also happen to be programmers will confirm that implementation goes both ways. We first need a well-defined language, data structures and semantics before we can even start implementing something like a model checker. But once we start implementing we can also learn from an implementation what works well and what does not. A strictly typed functional language like Haskell, which closely resembles mathematical syntax, can provide new insights that motivate us to go back and improve our original definitions. Then, after a fruitful back and forth between theory and implementation, we want to measure the quality of our implementation.

Research Question 3.
How good is the performance of symbolic methods for DEL?

To answer this, we should compare our new methods in two directions: First, there are existing model checkers for DEL which have been used in previous work and we expect symbolic methods to be much faster and to use less memory. Second, there are multiple model checkers for temporal logics with knowledge. Such temporal logics can model similar situations as DEL, but they are more explicit about time steps. Given models of the same example in both frameworks, it makes sense to compare which specifications can be checked by different tools, and how fast.

In the literature on DEL it is common to define new modalities and update mechanisms, such that it often makes sense to talk about dynamic epistemic *logics* in the plural. We are therefore interested in both general ways to implement the most standard versions of DEL efficiently and tailor-made solutions for specific problems. We first define and discuss general methods that can be applied independent of the particular example at hand, and then move to two specific applications, the first of which is the knowledge of numeric variables.

Research Question 4.
How can we model knowledge of variables and values?

As part of the bigger research program “beyond knowing that” [Wan18], which studies epistemic logics with different modalities, formalizations of “knowing what” or “knowing the value” recently received more attention [GW16; Bal16]. We will compare different approaches to modeling such knowledge and discuss how they differ in expressivity of the languages and size of models.

The second application we discuss in detail are so-called gossip protocols. The classical gossip problem, also known as the telephone problem, asks how many phone calls are needed between a group of agents to spread secrets to everyone, given that each agent starts only with their own secret. More generally, gossip provides a formal model of any peer-to-peer network in which information has to be synchronized. Recent applications of gossip can be found in decentralized communication systems [Irv16] and cryptocurrencies [SLZ16; Bai17].

Dynamic gossip is a generalization of the gossip setting in which phone numbers are exchanged in addition to secrets. We no longer assume that everyone can call everyone, but instead there is a reachability graph which constantly changes while the protocol runs. As expected, this complicates the setting and new protocols are needed. Given the decentralized nature of gossip, we are mainly interested in protocols that can be executed by agents without any central scheduler. Epistemic logic is an excellent tool to analyze dynamic gossip and we will describe protocols and their execution in a variant of DEL. Our next research question is whether besides describing existing protocols from the literature we can also use our logic to define new ones.

> Research Question 5.
> *Can we improve gossip protocols using epistemic logic?*

As part of this investigation we will show how model checking can be used to analyze dynamic gossip. It is then a natural combination of our topics to ask whether the symbolic methods can also be applied to the gossip problem.

> Research Question 6.
> *Can we use model checking for DEL to analyze gossip protocols?*

Concluding this introduction, the fields in which this thesis takes place are *Logic and Computation* — in particular their intersection. We are less concerned with purely logical questions such as proof theory, but more with those aspects of our logics that are relevant for implementations, for example the size of models and their representations. Vice versa, we do not focus on purely computational questions such as the computational complexity of model checking a given logic, but rather how we can tweak the syntax and semantics of a formal language to find the right balance between expressivity, usability and model checking performance.

Throughout the thesis we make heavy use of *functional programming* in the strongly typed language *Haskell*. This allows us to structure our implementations similar to the mathematical definitions and will make our programs both easier to read and safer to run. Moreover, we try to follow best practices in academic software engineering, as recently discussed in [All+17]. All our tools are released as free software and all benchmarks are automated and documented in such a way that they are easy to reproduce. We give links to source code and implementations in the relevant chapters. There will also be a website for the thesis with further links and errata at `https://malv.in/phdthesis`.

Key Contributions

The main contributions of this thesis are the following:

1. Methods for symbolic model checking a range of logics, starting with plain Epistemic Logic, via Public Announcement Logic, up to Dynamic Epistemic Logic with action models, including factual change.

2. As part of these methods, symbolic analogues for Kripke models and action models: knowledge structures and knowledge transformers for S5 logics and belief structures and transformers for the general case. In some sense, our main contribution here is that we do not contribute anything fundamentally new, because we prove that these symbolic structures are equivalent to the well-known explicit models.

3. SMCDEL, an implementation of symbolic model checking Dynamic Epistemic Logic based on binary decision diagrams. It can be used as a Haskell library, but also as a stand-alone program with a command-line and a web interface. We release all our tools as free software under an open source license.

4. Reproducible benchmarks to compare the performance of epistemic model checkers on different examples from the literature, including epistemic puzzles and security protocols.

5. The new *Public Inspection Logic* formalizing the knowledge and public inspection of variables. We provide sound and complete axiomatizations for the single and the multi-agent case.

6. A proof that in dynamic gossip all gossip graphs are reachable as parts of larger graphs with more agents.

7. A new epistemic modality describing protocol-dependent knowledge as a variant of conditional knowledge.

8. A family of new epistemic protocols for the dynamic gossip problem, obtained by strengthening existing protocols in a natural way.

9. An impossibility theorem saying that there is no strongly successful strengthening of the “Learn New Secrets” protocol.

10. A symbolic modeling of dynamic gossip using knowledge transformers with factual change.

Outline

Here we give a short summary of each chapter.

In Chapter 1 we give an introduction to the different fields of research this thesis contributes to, starting with a list of logics for which we summarize the standard syntax and semantics: Epistemic Logic, Public Announcement Logic, Dynamic Epistemic Logic and Epistemic Temporal Logic. We also mention some existing results on the relation between the dynamic and temporal approach. The last three sections of the first chapter then introduce Model Checking. We discuss the state explosion problem and how it has been approached for temporal logics using symbolic instead of explicit models. Last but not least we define Binary Decision Diagrams which are the basis of most existing work on symbolic model checking and also of our implementation.

Chapter 2 contains the main theoretical contribution of this thesis: a symbolic representation for all models and updates of Dynamic Epistemic Logic. We start with $\mathsf{S5}$ Public Announcement Logic; then we extend our methods step by step to general action models with factual change. For each variant of DEL we proceed in the same way: First we define the new symbolic structures and how to interpret DEL formulas on them. Next, we give translations to go back and forth between explicit and symbolic representations. Finally, we prove that these translations are truthful, to show that our symbolic structures and transformers describe exactly the same classes as the well-known Kripke and action models.

This framework is then implemented in Chapter 3. We describe how all boolean reasoning done in the previous chapter can be done efficiently using Binary Decision Diagrams (BDDs). At the same time we translate our mathematical ideas into the functional and typed programming language Haskell. The result is SMCDEL, a symbolic model checker for different variants of DEL. We highlight some of the design choices made during the development, such as type-safe variable management, and give simple examples how to use SMCDEL.

In Chapter 4 we continue with more involved examples from the epistemic logic literature which have traditionally been analyzed with Kripke models. We show what the equivalent symbolic structures and transformers look like formally and in the implementation with BDDs. Some examples suggest themselves as benchmarks and we use them to compare the performance of our implementation to existing model checking software, both for dynamic and temporal logics.

For the last two chapters we zoom in on two specific variants and concrete applications of Dynamic Epistemic Logic.

Chapter 5 concerns the knowledge of numeric variables, i.e. knowing-what or knowing-the-value, in contrast to factual knowing-that. We first discuss binary encodings and our previous work on register models. Then we present Public Inspection Logic (PIL), a new logic of knowing and inspecting values. It abstracts and simplifies the reasoning about variables and their dependencies by removing values from the language. This leads to a sound and complete axiomatization of

PIL which relates it to existing theories of dependencies in relational databases and existing work on dependence and independence logic. Finally, we compare the three approaches to model numeric knowledge — binary encoding, register models and PIL — and discuss further related work.

In Chapter 6 we discuss Dynamic Gossip, a generalization of the classic gossip or telephone problem: How can a set of agents efficiently distribute a set of secrets? The chapter contains two main new results, about the reachability of gossip graphs and strengthening of protocols. First, we show that following the rules of dynamic gossip not all gossip graphs are reachable from initial graphs. However, given a large enough number of agents, we can construct any gossip graph as a subgraph. Second, we use dynamic epistemic logic to formalize dynamic gossip protocols. We present a new operator for protocol-dependent knowledge, and multiple ways of strengthening gossip protocols using this operator. We then show that there is no perfect strengthening of the "Learn New Secrets" protocol. For both results it was helpful to implement explicit model checking procedures which we also present in this chapter. Finally, we show how to apply the symbolic model checking methods from the first four chapters to the gossip problem.

Sources of the Chapters

Parts of this thesis have been published before. Here we give a quick overview which chapters and sections are based on what.

- Chapter 1 is a new summary of basic concepts and ideas from the literature.
- Chapter 2 is based on [Ben+15] and the extended version:

  Johan van Benthem, Jan van Eijck, Malvin Gattinger, and Kaile Su. "Symbolic Model Checking for Dynamic Epistemic Logic – S5 and Beyond". In: *Journal of Logic and Computation (JLC)* (2017). DOI: `10.1093/logcom/exx038`. URL: `https://is.gd/DELBDD` [Ben+17]
- Sections 2.7 and 2.8 are based on an ESSLLI 2017 student session paper:

  Malvin Gattinger. "Towards Symbolic Factual Change in DEL". in: *Proceedings of the ESSLLI 2017 Student Session.* Edited by Karoliina Lohiniva and Johannes Wahle. 2017, pages 14–24. URL: `https://is.gd/symbolicfactualchange` [Gat17b]
- The implementation discussed in Chapter 3 was first published online in June 2015 and has since been updated regularly. It is released under the GNU General Public License v2.0 and can be found under `https://github.com/jrclogic/SMCDEL` [Gat18].
- Chapter 4 brings together examples and benchmarks from the already mentioned publications and some new material.

- Section 5.2 is based on:

  Jan van Eijck and Malvin Gattinger. “Elements of Epistemic Crypto Logic”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS ’15. Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pages 1795–1796. ISBN: 978-1-4503-3413-6. URL: `https://dl.acm.org/citation.cfm?id=2773441` [EG15]

- Most parts of Chapter 5 are from:

  Jan van Eijck, Malvin Gattinger, and Yanjing Wang. “Knowing Values and Public Inspection”. In: *Seventh Indian Conference on Logic and Its Applications: ICLA 2017, Kanpur, India*. Edited by Sujata Ghosh and Sanjiva Prasad. 2017, pages 77–90. ISBN: 978-3-662-54069-5. DOI: `10.1007/978-3-662-54069-5_7`. URL: `https://arxiv.org/abs/1609.03338` [EGW17]

- Chapter 6 is based on:

  Hans van Ditmarsch, Malvin Gattinger, Louwe B. Kuijer, and Pere Pardo. *How Come You Don’t Call Me? Common Knowledge of Gossip Protocols*. Submitted. 2018 [Dit+18]

We also mention the following publication which is not included in this thesis but related. It presents an explicit model checker for a variant of DEL with agent types, including liars.

- Malvin Gattinger. “A Model Checker for the Hardest Logic Puzzle Ever”. In: *PhDs in Logic VIII, Darmstadt*. 2016 [Gat16]

List of Symbols

$\mathbb{N}$	natural numbers, starting with 0
$\mathcal{P}$	powerset
a, b, etc.	agents
i, j, etc.	agent variables
p, q, etc.	atomic propositional variables
p', p°, p^*, etc.	fresh variables, copies of p
V	vocabulary
V', V°, V^*, etc.	fresh vocabularies, copies of V
$\mathcal{L}_B$	boolean language
$\mathcal{L}$	epistemic language
$\mathcal{L}_P$	public announcement language
$\mathcal{L}_D$	dynamic epistemic language
$\mathcal{L}_S$	symbolic dynamic epistemic language
φ, ψ, etc.	formulas
$[p \mapsto \psi]\varphi$	substitution: replace p with ψ in φ
$\mathcal{M}$	Kripke model
W	set of worlds
π	valuation function
$\sim$, $\sim_i$	equivalence relations
R, R_i	relations
$\mathcal{A}$	action model
$\mathcal{F}$	structure
θ	state law
O, O_i	observational variables
Ω, Ω_i	observational laws
$\mathcal{X}$	transformer

Chapter 1

Basics

> Most papers in computer science describe how their author learned what someone else already knew.
>
> Peter Landin

This thesis combines ideas from epistemic logic and computer science. In this preliminary chapter we introduce the basic building blocks of our theory. Depending on their background, the reader should feel free to skip over sections about structures or methods already known to them.

We follow standard set theoretic notation and write $\mathcal{P}$ for the powerset. We assume a finite set of agents $I = \{1, \dots, n\}$ to which we also refer using names like Alice and Bob or for short $a, b, \dots$, or variables $i, j, \dots$.

For our formal languages we assume a countably infinite supply of fresh atomic propositional variables. A finite subset of these is also called *vocabulary* and denoted by V, U or similar letters. Primes, stars and circles as in $p', p^*, p^\circ, V', V^*, \dots$ will denote (sets of) fresh variables that we use as copies of previously defined propositions. We often write "iff" to abbreviate "if and only if".

1.0.1. Definition. The language $\mathcal{L}_B(V)$ of *boolean* formulas over a vocabulary V is given by the Backus–Naur form $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi$ where $p \in V$. We also use the common abbreviations $\bot := \neg\top$, $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \to \psi := \neg(\varphi \wedge \neg\psi)$. We also use big conjunction and disjunction as usual: $\bigwedge\{\varphi_1, \dots, \varphi_n\} := \varphi_1 \wedge \dots \wedge \varphi_n$, similarly for $\bigvee$.

A less common notation we will use frequently is $\sqsubseteq$ to abbreviate a formula which says that out of the propositions in the second argument exactly those in the first argument are true: $A \sqsubseteq B := \bigwedge A \wedge \bigwedge\{\neg p \mid p \in B \setminus A\}$.

We define exclusive disjunction by $\varphi \oplus \psi := (\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi)$. This operator is also known as XOR and generalizes to an n-ary connective that is true iff an odd number of the φ_i formulas is true: $\oplus\{\varphi_1, \dots, \varphi_n\} := (\dots(\varphi_1 \oplus \varphi_2) \dots \oplus \varphi_{n-1}) \oplus \varphi_n$.

1.0.2. Definition. A boolean assignment for a vocabulary V assigns to each atomic proposition a truth value. It is thus a function of type $s\colon V \to \{\mathsf{True}, \mathsf{False}\}$. When the vocabulary V is fixed, by a slight abuse of notation, we identify an assignment s with the subset of atomic propositions that it makes true, i.e. $s = \{p \in V \mid s(p) = \mathsf{True}\} \subseteq V$. We write $\vDash$ for the usual boolean semantics:

1. $s \vDash \top$ always holds
2. $s \vDash p$ iff $p \in s$
3. $s \vDash \neg\varphi$ iff not $s \vDash \varphi$
4. $s \vDash \varphi \wedge \psi$ iff $s \vDash \varphi$ and $s \vDash \psi$

A formula φ is *valid* iff it satisfies all assignments and then we write $\vDash \varphi$. We call two formulas φ and ψ *semantically equivalent* and write $\varphi \equiv \psi$ iff they satisfy exactly the same assignments.

Later we will also write $\vDash$ for other semantics between more complex models or structures and formulas. It will be clear from the context of what type the arguments are and thus which semantics we mean. In all languages we make heavy use of substitution and boolean quantification as follows.

1.0.3. Definition. For any two formulas φ and ψ and any propositional variable p, let $[p \mapsto \psi]\varphi$ denote the result of replacing every p in φ by ψ. For any finite set of propositional variables $A = \{p_1, \ldots, p_n\}$, let $[A \mapsto \psi]\varphi$ denote the result of simultaneously substituting ψ for all elements of A in φ.

For any two finite sets of the same size $A = \{p_1, \ldots, p_n\}$ and $B = \{q_1, \ldots, q_n\}$ let $[A \mapsto B]\varphi$ denote the result of simultaneously substituting each q_k for the corresponding p_k in φ for all $k \in \{1, \ldots, n\}$. Note that strictly speaking A and B need to be ordered lists for this and we use an implicit bijection between them.

The boolean quantifier $\forall p\varphi$ abbreviates $[p \mapsto \top]\varphi \wedge [p \mapsto \bot]\varphi$. For any finite set $A = \{p_1, \ldots, p_n\}$, let $\forall A\varphi := \forall p_1 \forall p_2 \ldots \forall p_n \varphi$. We define its dual as $\exists p\varphi := \neg\forall p(\neg\varphi)$ which gives us the equivalence $\exists p\varphi \equiv [p \mapsto \top]\varphi \vee [p \mapsto \bot]\varphi$. Similarly for finite sets A, $\exists A$ is the dual of $\forall A$. We also define an "out of" substitution: For any two finite sets $A \subseteq B$, let $[A \sqsubseteq B]\varphi := [A \mapsto \top][(B \setminus A) \mapsto \bot]\varphi$.

1.0.4. Example. The following true statements illustrate our basic definitions.

- $\{p_3, p_4, p_5\} \ \vDash\ p_5 \wedge \neg p_6 \wedge \{p_3, p_5\} \sqsubseteq \{p_k \mid 1 \leq k \leq 100 \text{ and } k \text{ is odd}\}$
- $\forall p(p \vee q) \equiv q$
- $\vDash\ \exists p(p \vee q)$
- $[\{p\} \sqsubseteq \{p, s\}]((p \wedge q) \vee (r \to s)) = (\top \wedge q) \vee (r \to \bot) \equiv q \vee \neg r$

Throughout this thesis we will often use boolean *formulas* to denote the boolean *functions* they represent: We only care about the semantics and not the particular syntax of a boolean formula. For the theory in Chapter 2 this difference actually does not matter, but in Chapter 3 we implement all our boolean reasoning with Binary Decision Diagrams, on which syntactic identity and semantic equivalence coincide (see Section 1.9).

1.1 Epistemic Logic on Kripke Models

Most of this thesis is about epistemic logic, the study of knowledge, belief and other epistemic attitudes using formal languages and mathematical models. Almost all approaches to epistemic logic use *modal logics*. Those are logics with operators that besides knowledge can also model the passing of time, the execution of a program, possibility or any other modality. Additionally, modal logic has become an independent field of study in mathematics and computer science. For a general and thorough introduction for which we do not have time and space here, see [BRV01]. Coming back to epistemic logic, one if not *the* classic reference is [Fag+95]. We start by defining our main language.

1.1.1. Definition. Given a vocabulary V, the language of epistemic logic $\mathcal{L}(V)$ extends the boolean language $\mathcal{L}_B(V)$ from Definition 1.0.1 and is given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C_\Delta\varphi$$

where $p \in V$, $i \in I$ and $\Delta \subseteq I$. We also use the other boolean connectives $\bot$, $\vee$, $\to$ in $\mathcal{L}(V)$ and define the abbreviation $K_i^? \varphi := K_i\varphi \vee K_i\neg\varphi$.

We read the formula $K_i\varphi$ as "agent i knows *that* φ is true", $K_i^?\varphi$ means "agent i knows *whether* φ is true", and the formula $C_\Delta\varphi$ says that φ is common knowledge among agents in the group Δ. If $\Delta = \{i\}$ for a single agent $i \in I$, then we also just write i instead of $\{i\}$. Hence, K_i and C_i are the same as, but for clarity we define K as a primitive connective with its own (simpler) semantics. The standard semantics for $\mathcal{L}$ are given by means of Kripke models as follows.

1.1.2. Definition. A *frame* for a set of agents $I = \{1, \ldots, n\}$ is a tuple $\mathcal{M} = (W, R)$, where W is a finite set of *possible worlds* and R is a family of binary relations over W indexed by agents: $R_i \subseteq W \times W$ for each $i \in I$.

A *Kripke model* for a set of agents I and vocabulary V is a tuple $\mathcal{M} = (W, \pi, R)$, where (W, R) is a frame for I and $\pi \colon W \to \mathcal{P}(V)$ is a *valuation function*: $\pi(w) \subseteq V$ for each $w \in W$.

By convention, we use $W^\mathcal{M}$, $R_i^\mathcal{M}$ and $\pi^\mathcal{M}$ to refer to the components of $\mathcal{M}$ but we omit the superscript $\mathcal{M}$ if it is clear from the context which model we are concerned with.

For any group of agents $\Delta \subseteq I$ we denote the transitive closure of the union of their relations by $R_\Delta := \left(\bigcup_{i\in\Delta} R_i\right)^*$ which we will use to interpret common knowledge below. A model $\mathcal{M}$ is *finite* iff $W^{\mathcal{M}}$ is finite. A model is an *S5 Kripke model* iff, for every i, the relation R_i is an equivalence relation. In this case we also write $\sim_i$ for R_i.

A *pointed Kripke model* is a pair $(\mathcal{M}, w)$ of where w is a world of $\mathcal{M}$.

For each agent i we thus have a relation R_i telling us which worlds the agent considers possible. In the semantics below we then use this relation to define knowledge in terms of possibility: i knows something iff it is the case at all the worlds that i considers possible. Phrased differently and assuming that R_i is an equivalence relation: i knows something iff it is true at all those worlds that i cannot distinguish from the actual world.

This definition of Kripke models is standard in the literature, but we should highlight a part of it that is often left implicit: Kripke models come with a vocabulary V that defines the codomain of the valuation function π. We already make this explicit now because later on we will deal with multiple different vocabularies and have to be precise which languages over which vocabulary can be interpreted on which models and structures.

1.1.3. Definition. Semantics for $\mathcal{L}(V)$ *on pointed Kripke models* are given inductively as follows.

1. $(\mathcal{M}, w) \vDash \top$ always holds.
2. $(\mathcal{M}, w) \vDash p$ iff $p \in \pi^{\mathcal{M}}(w)$.
3. $(\mathcal{M}, w) \vDash \neg\varphi$ iff not $(\mathcal{M}, w) \vDash \varphi$.
4. $(\mathcal{M}, w) \vDash \varphi \wedge \psi$ iff $(\mathcal{M}, w) \vDash \varphi$ and $(\mathcal{M}, w) \vDash \psi$
5. $(\mathcal{M}, w) \vDash K_i\varphi$ iff for all $w' \in W$, if $R_i ww'$, then $(\mathcal{M}, w') \vDash \varphi$.
6. $(\mathcal{M}, w) \vDash C_\Delta\varphi$ iff for all $w' \in W$, if $R_\Delta ww'$, then $(\mathcal{M}, w') \vDash \varphi$.

If we consider all Kripke models, the set of valid formulas obtained from these semantics is the logic usually called K. Additionally, we know that restricting the class of *frames* to specific kinds of relations corresponds to adding specific axioms. For example, the class of transitive frames is characterized by the axiom $K_ip \to K_iK_ip$. Moreover, such axioms also have an intuitive reading in epistemic logic, as summarized in Table 1.1.

Any reflexive and Euclidean relation is in fact an *equivalence relation*. The corresponding modal logic is usually abbreviated as $\mathsf{S5}$ and is the one used most often to model knowledge [Fag+95; DHK07].

However, the $\mathsf{S5}$ notion of knowing should not be identified with the natural language use of “know”. The logic $\mathsf{S5}$ describes a *strong* notion of knowledge in

Name	Axiom	Class of Relations	Epistemic Property
D	$K_i\varphi \to \neg K_i \neg\varphi$	serial	Consistency
T	$K_i\varphi \to \varphi$	reflexive	Truth
4	$K_i\varphi \to K_i K_i\varphi$	transitive	Positive Introspection
5	$\neg K_i\varphi \to K_i \neg K_i\varphi$	Euclidean	Negative Introspection

Table 1.1: Modal correspondences and epistemic counterparts.

the following sense. Whatever is known also has to be true, any agent who knows something also knows that she knows it and if an agent does not know something, she knows that she does not know it. In particular the latter two, positive and negative introspection, are controversial in Philosophy [HS15].

Besides this hard notion of knowledge also other modalities and their dynamics have been formalized. *Belief* for example can be modeled using weaker modal logics with more general relational semantics based on arbitrary relations. We also consider these general, non-S5 models here. When working with such models, to emphasize that the underlying relations do not have to be equivalence relations and we are no longer talking about knowledge, we write R_i instead of $\sim_i$ for the epistemic relations and $\Box_i$ instead of K_i for the modal operator. Still, we do *not* change the semantics to interpret $\Box_i$:

$$(\mathcal{M}, w) \vDash \Box_i\varphi \quad \text{iff} \quad \text{for all } w' \in W : \text{If } R_i w w' \text{ then } (\mathcal{M}, w') \vDash \varphi.$$

1.1.4. Example. Figure 1.1 shows an $\mathsf{S5}$ Kripke model $\mathcal{M}_1$ and a non-$\mathsf{S5}$ model $\mathcal{M}_2$. Both models consist of two worlds and describe the epistemic state of two agents called Alice and Bob. We highlight the actual world with a double border. When drawing $\mathsf{S5}$ models we leave out the reflexive arrows and instead of two arrows back and forth we draw one undirected edge between worlds. We illustrate our semantics with the following true statements.

- $\mathcal{M}_1, w_1 \vDash q \wedge \neg p \wedge C_{\text{Alice,Bob}} q \wedge K_{\text{Alice}} \neg p \wedge \neg K_{\text{Bob}} p$
- $\mathcal{M}_2, w_1 \vDash q \wedge \neg p \wedge \Box_{\text{Alice}} \neg p \wedge \Box_{\text{Bob}} p \wedge \Box_{\text{Bob}} \Box_{\text{Alice}} p$

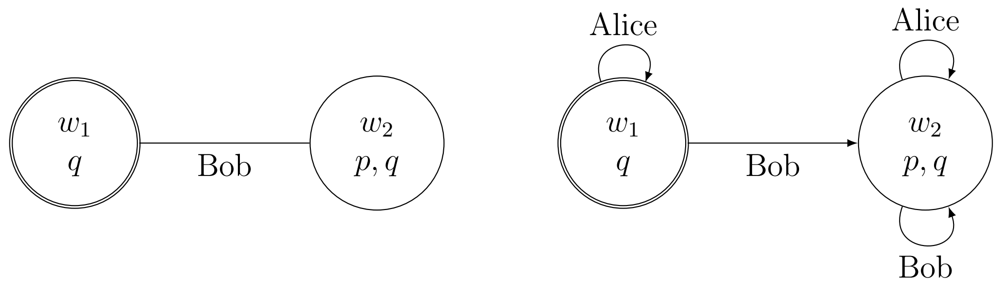


Figure 1.1: $\mathsf{S5}$ model $\mathcal{M}_1$ and non-$\mathsf{S5}$ model $\mathcal{M}_2$.

For this thesis we put aside the philosophical quest for the "real" notion or definition of knowledge. We first present our framework for the widely used S5 but then also extend our methods to weaker logics. Hence no matter which set of axioms and class of models might be the right one for a particular task, our methods can be applied.

Still, all agents in our framework know all the logical consequences of what they know, i.e. $K_i(\varphi \to \psi) \to (K_i\varphi \to K_i\psi)$ is valid and if φ is valid, then so is $K_i\varphi$. All epistemic logics based on Kripke models as defined above are *normal* modal logics and thereby have this property of *logical omniscience*. One could say that all our agents are perfect logicians. This assumption is unrealistic for humans or other real agents which are computationally bounded, but for concrete examples and applications we can ignore the problem as we only care about what our models say about specific formulas. In many settings it is actually the more careful choice to let agents be perfect reasoners: If a protocol ensures that someone cannot know something even if they are logically omniscient, then this also holds for agents with bounded rationality. The converse however, is not true in general.

When working with Kripke models, a useful and important notion is that of bisimulation. It provides a semantic characterization when two models are equivalent. We consider a modal logic well-behaved when this semantic notion coincides with the syntactic condition that models satisfy the same formulas. As the following famous theorems state, this is the case for the epistemic logics we will consider in this thesis.

1.1.5. Definition. Suppose that we have two Kripke models $\mathcal{M}_1 = (W^1, \pi^1, R^1)$ and $\mathcal{M}_2 = (W^2, \pi^2, R^2)$ for the same vocabulary and the same set of agents. A relation $Z \subseteq W^1 \times W^2$ linking possible worlds from $\mathcal{M}_1$ to those from $\mathcal{M}_2$ is a *bisimulation* iff for all linked worlds $(w^1, w^2) \in Z$ the following three conditions hold:

- Propositional agreement: $\pi^1(w^1) = \pi^2(w^2)$
- Forth: For every agent i and for every v^1 such that $R^1_i w^1 v^1$ there is a v^2 such that $R^2_i w^2 v^2$ and $(v^1, v^2) \in Z$.
- Back: For every agent i and for every v^2 such that $R^2_i w^2 v^2$ there is a v^1 such that $R^1_i w^1 v^1$ and $(v^1, v^2) \in Z$.

Two pointed Kripke models $(\mathcal{M}^1, w^1)$ and $(\mathcal{M}^2, w^2)$ are *bisimilar* iff there is a bisimulation Z such that $(w^1, w^2) \in Z$.

1.1.6. Theorem. *If two pointed Kripke models for the same vocabulary V and the same set of agents are bisimilar, then they satisfy the same formulas of $\mathcal{L}(V)$.*

We can also relate bisimulation and equivalence in the other direction, but only for image-finite models. Intuitively, this is because any modal formula only depends on finitely many worlds.

1.1.7. Definition. We call a Kripke model $\mathcal{M} = (W, \pi, R)$ *image-finite* iff for every agent i and every possible world $w \in W$ the set $\{v \in W \mid R_i wv\}$ is finite.

Note that in particular all finite Kripke models are image-finite. Given that we are mainly interested in model checking in this thesis, finite models are our main object of study and the following theorem applies.

1.1.8. Theorem (Hennessy-Milner Theorem). *If two pointed image-finite Kripke models for the same vocabulary V and the same set of agents satisfy the same formulas of $\mathcal{L}(V)$, then they are bisimilar.*

For proofs of Theorem 1.1.6 and Theorem 1.1.8, we refer the interested reader to [BRV01] in which they are listed as Theorem 2.20 and Theorem 2.24, respectively. A key insight is that the relation of semantic equivalence itself is a bisimulation.

1.2 Public Announcement Logic

Besides modeling the knowledge of agents we are also interested in how their epistemic states can change. The first logical approach to changes of knowledge is the seminal [Pla07], first published in 1989. The logic presented there is nowadays called Public Announcement Logic (PAL) and extends epistemic logic with a modality to describe incoming information.

1.2.1. Definition. Given a vocabulary V, the language $\mathcal{L}_P(V)$ for PAL is given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C_\Delta\varphi \mid [!\varphi]\varphi$$

where $p \in V$, $i \in I$ and $\Delta \subseteq I$.

We also define the abbreviation $[?!\psi]\varphi := [!\psi]\varphi \wedge [!\neg\psi]\varphi$.

The new formula $[!\psi]\varphi$ indicates that after a *public announcement* of ψ, φ holds. Formally, $[!\psi]$ is a dynamic operator which takes us to a new model consisting only of those worlds where ψ was true. After the update we then evaluate φ in the new model.

The operator $[!\psi]$ can thus be read as "After the public announcement that ψ is true, it will be the case that …". Similarly, the abbreviation $[?!\varphi]$ can be read as "After a public announcement *whether* φ holds, it will be the case that …".

1.2.2. Definition. We interpret $\mathcal{L}_P(V)$ on Kripke models for the vocabulary V by adding the following clause to the previous Definition 1.1.3:

$$(\mathcal{M}, w) \vDash [!\psi]\varphi \text{ iff } (\mathcal{M}, w) \vDash \psi \text{ implies } (\mathcal{M}^\psi, w) \vDash \varphi$$

where $\mathcal{M}^\psi$ is a new model defined by the set $W^{\mathcal{M}^\psi} := \{w \in W^{\mathcal{M}} \mid (\mathcal{M}, w) \vDash \psi\}$, the relations $R_i^{\mathcal{M}^\psi} := R_i^M \cap (W^{\mathcal{M}^\psi})^2$ and the valuation $\pi^{\mathcal{M}^\psi}(w) := \pi^{\mathcal{M}}(w)$.

Public announcements can create common knowledge, as Example 1.2.3 and Fact 1.2.4 below show. In fact they are often the only way to establish common knowledge, because any (partially) private announcement leaves room for speculation: Someone might not have received the same information, or might think that someone else did not receive it, or any other nesting of suspicions.

However, public announcements do *not* always create common knowledge. The classic counterexamples are so-called *Moore sentences* as in Example 1.2.5.

1.2.3. Example. Whenever $p \vee q$ is truthfully and publicly announced, it will be common knowledge among all agents after the announcement. Formally, the formula $[!p \vee q]C(p \vee q)$ is valid.

1.2.4. Fact. For any boolean formula $\varphi \in \mathcal{L}_B$, the formula $[!\varphi]C\varphi \in \mathcal{L}_P$ is valid.

1.2.5. Example. The sentence "It is raining in Amsterdam and you don't know it." can be announced truthfully, but it will not be common knowledge afterwards because it will not be true any longer. Formally, $[!p \wedge \neg K_i p]$ never leads to a model where $p \wedge \neg K_i p$ is common knowledge. In contrast, $[!p \wedge \neg K_i p]K_i p$ is valid.

Interestingly, public announcement operators do not actually allow us to say anything new: $\mathcal{L}_P$ has the same expressivity as $\mathcal{L}$. Moreover, there is an easy translation procedure to remove public announcement operators which is part of the standard axiomatization of PAL. As the focus of this thesis is on semantics and model checking instead of proof theory, we will not discuss a complete axiomatization. Still it should be noted that these axioms alone are not enough and PAL can be axiomatized in different ways, as shown in [WC13].

1.2.6. Fact. The following $\mathcal{L}_P$ formulas called *reduction axioms* are valid.

- $[!\varphi]p \leftrightarrow (\varphi \rightarrow p)$
- $[!\varphi]\neg\psi \leftrightarrow (\varphi \rightarrow \neg[!\varphi]\psi)$
- $[!\varphi](\psi_1 \wedge \psi_2) \leftrightarrow ([!\varphi]\psi_1 \wedge [!\varphi]\psi_2)$
- $[!\varphi]K_i\psi \leftrightarrow (\varphi \rightarrow K_i(\varphi \rightarrow [!\varphi]\psi))$

Hence for every formula in $\mathcal{L}_P$ without C there is an equivalent formula in $\mathcal{L}$.

We exclude the common knowledge operator C in Fact 1.2.6, because announcements cannot simply be pushed through it. Instead, a more expressive language with conditional knowledge is needed to obtain reduction axioms for public announcement logic with common knowledge. We refer to [BEK06] and [DHK07, Section 8.8] for further details on the expressivity of PAL with common knowledge.

Given that PAL is thus equally expressive as plain epistemic logic, one might wonder if it is still useful. However, formalizing concrete examples using the public announcement operator is usually more natural.

Moreover, the translation increases the size of the formula. The difference in length can even be exponential and there are properties which $\mathcal{L}$ can only express in an exponentially longer formula than $\mathcal{L}_P$, no matter which translation is used. However, the satisfiability problems for $\mathcal{L}_P$ and $\mathcal{L}$ still have the same complexity. We refer to [Lut06] for these results.

Before going to more general updates we mention an equivalent definition of public announcements which is also common in the literature: Instead of removing all non-ψ worlds when ψ is announced, we can cut all links leading to them. Formally, for each agent i let the new relation be $R_i^{M^\psi} := \{(v,w) \mid R_i vw \text{ and } \mathcal{M}, w \vDash \psi\}$ and leave W and π as they are. Yet another and again equivalent definition would be to cut all links between worlds which differ on the announced formula: $R_i^{M^\psi} := \{(v,w) \mid R_i wv \text{ and } (\mathcal{M}, v \vDash \psi \text{ iff } \mathcal{M}, w \vDash \psi)\}$. In this last variant announcing φ and announcing $\neg\varphi$ is the same action and we could say "announcing whether" instead of "announcing that". Computationally however, models obtained by cutting links are worse because they might contain more "garbage" in the form of unreachable worlds.

1.3 Dynamic Epistemic Logic with Action Models

The previous section describes the most primitive way in which knowledge among multiple agents can change: a truthful public announcement made by a trusted authority, received and accepted by everyone. But there are many other types of communication and events that affect knowledge and belief: We can tell someone something in secret, hidden completely or partially from others. Carol might observe *that* Alice is talking to Bob but not know *what* Alice is telling him. Moreover, such events can be *deceiving*: Carol might believe that Alice tells Bob that she got a job, but actually she tells him that she got a cat.

Besides purely epistemic events of communication we can also have *factual change* that can be public or not: Suppose I flip a coin at random and then look at the result without showing it to you. This changes a fact in the world, namely which side of the coin is up, and makes it known to me but hidden from you.

These more complex actions of communication and change can be modeled using so-called *action models*. They provide a natural formal way to talk about the dynamics of information in the same way that Kripke models formalize static information. The general idea is to think of events as possible worlds. A Kripke model says which different situations the agents can distinguish. An action model then formalizes which different events the agents can tell apart

Action models were first presented in [BMS98] which might be called the starting point of modern Dynamic Epistemic Logic (DEL) in general. The logic of action models was then generalized in [BEK06] to also accommodate factual change, using a version of Propositional Dynamic Logic (PDL). An axiomatization for factual change was also developed in [DK08] where it is called *ontic* change, in contrast to purely *epistemic* updates.

From a more general viewpoint, even if one does not want to use the full logics presented in those seminal papers, they still provide a general method to obtain sound and complete reduction axioms for languages with dynamic operators. A rule of thumb is that if an epistemic update can be represented as an action model, then it is straightforward to find reduction axioms for it.

The following definition describes action models and how they can be applied to Kripke models. Our definition of postconditions differs from the standard in [BEK06] because we only allow boolean formulas. This however does not change the expressivity [DK08].

1.3.1. Definition. Suppose we have some vocabulary V. An *action model* is a tuple $\mathcal{A} = (A, R^{\mathcal{A}}, \mathsf{pre}, \mathsf{post})$ where A is a set of atomic events, $R^{\mathcal{A}}$ is a family of relations $R_i \subseteq A \times A$ for each i, $\mathsf{pre}\colon A \to \mathcal{L}(V)$ is a function which assigns to each event a formula called the *precondition* and $\mathsf{post}\colon A \times V \to \mathcal{L}_B(V)$ is a function which at each event assigns to each atomic proposition a boolean formula called the *postcondition.* We call $\mathcal{A}$ an $\mathsf{S5}$ action model iff all the relations are equivalence relations.

Given a Kripke model $\mathcal{M}$ and an action model $\mathcal{A}$ using the same vocabulary, we define their *product* by $\mathcal{M} \times \mathcal{A} := (W^{\text{new}}, R_i^{\text{new}}, \pi^{\text{new}})$ where

- $W^{\text{new}} := \{(w, a) \in W \times A \mid \mathcal{M}, w \vDash \mathsf{pre}(a)\}$
- $R_i^{\text{new}} := \{((w, a), (v, b)) \mid R_i^{\mathcal{M}} wv \text{ and } R_i^{\mathcal{A}} ab\}$
- $\pi^{\text{new}}((w, a)) := \{p \in V \mid \mathcal{M}, w \vDash \mathsf{post}_a(p)\}$

We will first focus on action models without *factual change*, namely tuples (A, R, pre) without the component post. To update with such an action model the last clause should be $\pi^{\text{new}}((w, a)) := \pi(w)$, which means we just keep the old valuation function. Equivalently, we could say that post maps each atomic proposition to itself (as a formula, so technically this is not an identity function).

An *action* is a pair $(\mathcal{A}, a)$ where $a \in A$. To update a pointed Kripke model with an action we define $(\mathcal{M}, w) \times (\mathcal{A}, a) := (\mathcal{M} \times \mathcal{A}, (w, a))$.

1.3.2. Fact. The product of an $\mathsf{S5}$ Kripke model and an $\mathsf{S5}$ action model is again an $\mathsf{S5}$ Kripke model.

To illustrate the definitions of action models and the product update, let us first consider a simple S5 example without factual change.

1.3.3. EXAMPLE. Alice has applied for a post-doc position and Bob knows this. While Alice and Bob are in the same room, a messenger enters and gives Alice an envelope with the university logo on it. She reads the letter and learns that she got the position while Bob observes this but he does not see the content of the letter. Bob only learns that Alice learns *whether* she got the position.

Let p stand for the atomic proposition "Alice gets the position." The initial situation can be represented by model $\mathcal{M}$ shown on the left in Figure 1.2. Both Alice and Bob do not know whether p and this is common knowledge among them.

Alice reading the letter is modeled as the action model $\mathcal{A}$ in the middle of Figure 1.2. The two events in this case stand for the two possible contents of the letter, thus they have the preconditions $?p$ and $?\neg p$. To indicate that $\mathcal{A}$ is an action model and not a Kripke model we draw events as rectangles instead of circles and we prefix precondition with a question mark.

The result of the product update $\mathcal{M} \times \mathcal{A}$ is shown on the right in Figure 1.2. Again note that Bob did not learn whether p, but he did learn that Alice learns whether p. In fact, we have $\mathcal{M} \times \mathcal{A} \vDash C_{\text{Alice,Bob}}(K_{\text{Alice}}p \vee K_{\text{Alice}}\neg p)$ which means that it is now common knowledge between them that she knows whether p holds.

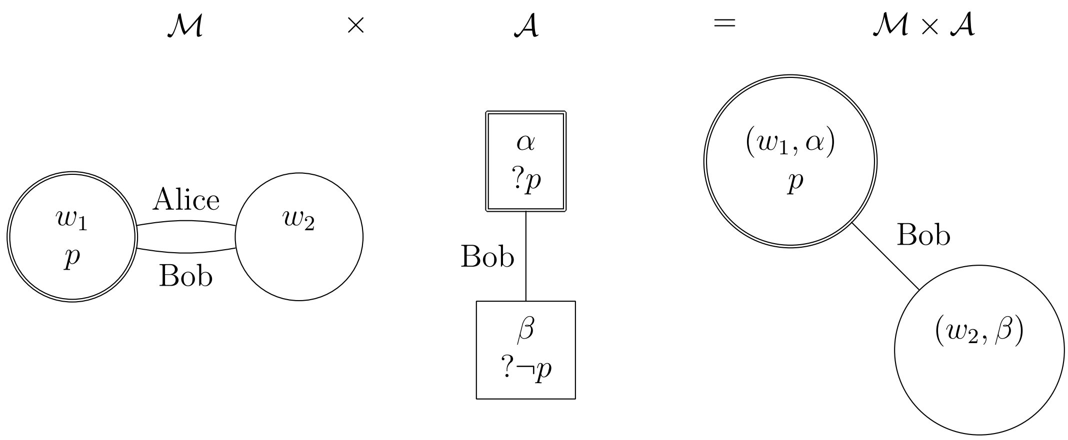


Figure 1.2: Alice reads the letter, observed by Bob.

We can also use action models to model changes of belief instead of knowledge as the following non-S5 example shows.

1.3.4. EXAMPLE. Suppose in Example 1.3.3 Alice reads the letter in private, such that Bob does not notice anything. We model such a fully private announcement of p to Alice in Figure 1.3. In the resulting pointed model p holds, Alice knows it but Bob does not. Moreover, Bob has a false belief that Alice still does not know it. Formally, we have $p \wedge \Box_{\text{Alice}}p \wedge \neg\Box_{\text{Bob}}p \wedge \Box_{\text{Bob}}\neg\Box_{\text{Alice}}p$.

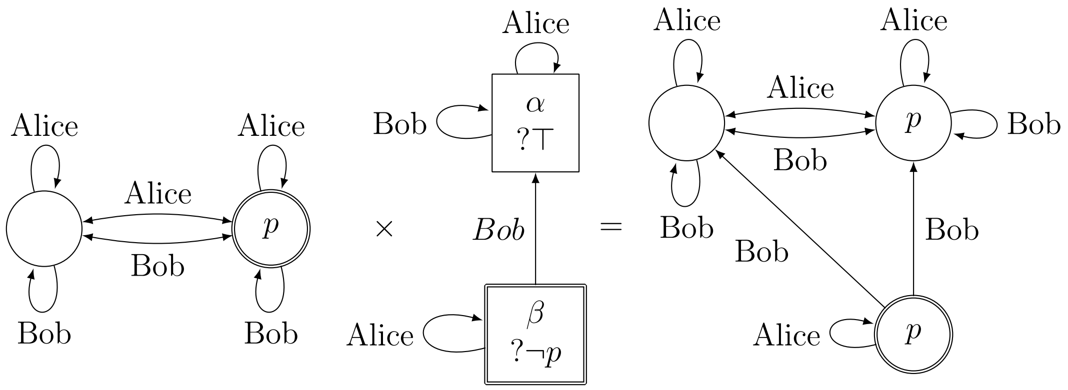


Figure 1.3: Alice secretly reads the letter.

Example 1.3.4 only concerns epistemic change. In the next example we also change facts about the world using postconditions.

1.3.5. Example. Consider a coin lying on a table with heads up: p is true and this is common knowledge. Suppose we then toss it randomly and hide the result from agent a but reveal it to agent b. Figure 1.4 shows a Kripke model of this the initial situation, an action model representing the coin flip and the resulting Kripke model.

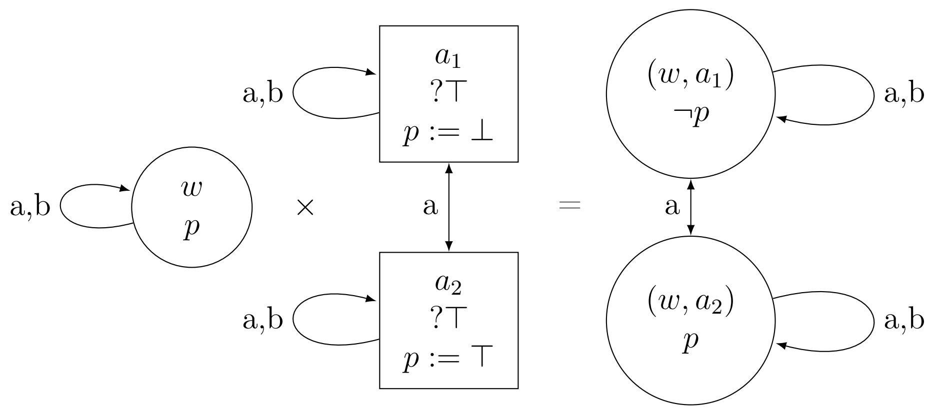


Figure 1.4: A coin flip hidden from agent a.

One of the main features of DEL is that we can add action models as operators to our language. Similar to the public announcement operator $[!\varphi]$ we get a new modality for each pointed action model.

1.3.6. Definition. Given a vocabulary V, the language of Dynamic Epistemic Logic $\mathcal{L}_D(V)$ with dynamic operators for action models extends $\mathcal{L}(V)$ and is given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C_\Delta\varphi \mid [\mathcal{A}, a]\varphi$$

where $p \in V$, $i \in I$, $\Delta \subseteq I$ and $(\mathcal{A}, a)$ is an action as in Definition 1.3.1.

1.3.7. Definition. We interpret dynamic operators for action models as follows:

$$\mathcal{M}, w \vDash [\mathcal{A}, a]\varphi \text{ iff } \mathcal{M}, w \vDash \mathsf{pre}(a) \text{ implies } \mathcal{M} \times \mathcal{A}, (w, a) \vDash \varphi$$

Action models live in two worlds: On the one hand they are semantic objects similar to Kripke models. On the other hand, in $\mathcal{L}_D$ they are syntactic objects similar to public announcements in $\mathcal{L}_P$. We thus have formulas in action models and action models in formulas. Should we get worried whether $\mathcal{L}_D$ is well-defined?

This problem has been discussed extensively in the literature. Giving a correct well-founded account of a most general DEL language is tricky. We can of course allow preconditions and postconditions to also include dynamic operators, but if formulas should remain finite objects and evaluating them should always be possible, then we have to forbid self-reference. Preconditions in an action $\mathcal{A}$ may not include a dynamic operator $[\mathcal{A}, a]$ for the same $\mathcal{A}$. Also any general version of this can lead to contradictions: If $\mathcal{A}$ includes a precondition involving $[\mathcal{A}', a']$ and $\mathcal{A}'$ has a precondition involving $[\mathcal{A}, a]$, then we might have to jump back and forth and never be able to evaluate them. More formally, consider a graph of action models and formulas where an edge means "occurs in". Any cycle in this graph means that model checking would never terminate. For a detailed discussion and an inductive definition to avoid such circularity, see [DHK07, Section 6.1].

We now take the easy way out in this thesis: Definitions 1.3.1 and 1.3.6 are meant exactly as we stated them. Note especially that Definition 1.3.1 uses $\mathcal{L}$ and not $\mathcal{L}_D$. Preconditions are not allowed to contain any dynamic operators. The same holds for postconditions which we restricted even further, to only boolean formulas. It is then clear that our language and its semantics are well-founded.

Fortunately, this does not restrict the applicability of what follows, because similar to the restriction to boolean postconditions, restricting preconditions does not restrict the class of updates we can describe [DK08].

If we do not have the common knowledge operator C, then similar to PAL, also DEL with action models is equally expressive as plain epistemic logic, because of the following reduction axioms. This does not make languages with action models useless — they are much more convenient and can be exponentially more succinct.

1.3.8. Fact. The following $\mathcal{L}_D$ formulas called *reduction axioms* are valid.

- $[\mathcal{A}, a]p \leftrightarrow (\mathsf{pre}^{\mathcal{A}}(a) \rightarrow \mathsf{post}^{\mathcal{A}}_a(p))$
- $[\mathcal{A}, a]\neg\psi \leftrightarrow (\mathsf{pre}^{\mathcal{A}}(a) \rightarrow \neg[\mathcal{A}, a]\psi)$
- $[\mathcal{A}, a](\psi_1 \wedge \psi_2) \leftrightarrow ([\mathcal{A}, a]\psi_1 \wedge [\mathcal{A}, a]\psi_2)$
- $[\mathcal{A}, a]K_i\psi \leftrightarrow (\mathsf{pre}^{\mathcal{A}}(a) \rightarrow \bigwedge_{b \sim_i a} K_i[\mathcal{A}, b]\psi)$

Hence for every formula in $\mathcal{L}_D$ without C there is an equivalent formula in $\mathcal{L}$.

Concluding this section, we stress the generality of action models: Basically any transformation of Kripke models can be seen as a product update with an action model — including public, semi-private and private announcements. The product update can almost get us from any model to any other. Already action models without factual change reach all refinements of a model and for any formula φ there is an action model that, whenever it is applicable, will make φ true [Hal13].

1.4 Arrow Updates

We have seen that action models provide a very general method to change Kripke models. In some settings though, less expressivity might be wanted. Also, the definition of action models and the product update above focuses on *which worlds are kept or created.* Intuitively, action models generalize the "deleting worlds" definition of public announcements and not the "cutting links" idea — though formally action models can of course do both.

An alternative method to describe dynamics of knowledge focuses on how epistemic relations are changed: *Arrow Update Logic* from [KR11b] describes model transformations using triples of the form (ψ, i, χ) where ψ and χ are formulas and i is an agent. The intuitive reading of such a triple is that the epistemic edges for agent i should be restricted to those going from a ψ-world to a χ-world.

1.4.1. Definition. An *arrow* is a tuple (ψ, i, χ) where $\psi, \chi \in \mathcal{L}$ and $i \in I$. We define dynamic operators for any finite set of arrows U with these semantics:

$$\mathcal{M}, w \vDash [U]\varphi \text{ iff } \mathcal{M} * U, w \vDash \varphi$$

where $\mathcal{M} * U$ is a new model defined by:

- $W^{\mathcal{M}*U} := W$
- $R_i^{\mathcal{M}*U} wv :\iff R_i^{\mathcal{M}} wv$ and there are $\psi, \chi \in \mathcal{L}$ such that $(\psi, i, \chi) \in U$ and $\mathcal{M}, w \vDash \psi$ and $\mathcal{M}, v \vDash \chi$
- $\pi^{\mathcal{M}*U} := \pi$

1.4.2. Example. The letter story from Example 1.3.3 can also be described as an arrow update with $\{(p, \text{Alice}, p), (\neg p, \text{Alice}, \neg p), (\top, \text{Bob}, \top)\}$. In contrast, Example 1.3.4, where Alice reads the letter in private, cannot be modeled with an arrow update, because arrow updates never increase the number of worlds.

In [KR11b] it is shown that every arrow update can be emulated by an action model. However, the action model might be exponentially larger than the arrow update. Arrow updates can thus be seen as a form of abstraction or symbolic representation of much larger action models, similar to the transformers we present in the next chapter.

Arrow Update Logic was further generalized in [KR11a] and it was shown that generalized arrow updates describe the same class of updates as action models without postconditions for factual change. For this thesis, we will only consider the basic version given by Definition 1.4.1.

1.5 Temporal Logics on Interpreted Systems

The dynamic languages we defined in Sections 1.2 to 1.4 focus on *what* happens and their dynamic operators are interpreted by changing the model. There is also a plethora of *temporal* logics which focus on *when* something is the case. Their modal operators do not describe actions but instead refer to the passage of time. Such temporal logics are not our main object of study, but most existing work on model checking, in particular symbolic representation, was developed for such languages with temporal operators. Therefore, in this section we provide definitions for a basic temporal logic as a reference.

We follow [LP15] and use an epistemic version of the branching time logic CTL*. In particular, we do not distinguish between state and path formulas which is sometimes done to describe fragments of CTL* like CTL and LTL [CGP99, Section 3.2].

1.5.1. Definition. The CTLK language $\mathcal{L}_T(V)$ for a vocabulary V is given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \land \varphi \mid K_i\varphi \mid C_\Delta\varphi \mid AX\varphi \mid AG\varphi \mid A(\varphi U \varphi)$$

where $p \in V$, $i \in I$ and $\Delta \subseteq I$.

The dual of AX is $EX\varphi := \neg AX\neg\varphi$ and similarly for the other operators.

We can read A as "in all possible paths", together with the operators X for "at the next step", G for "always in the future" and U for "until".

We now deviate from [LP15] in two ways: First, our global states are primitives and not built from local states for agents and environments. But we still have an equivalence relation over states for each agent. Second, our transitions are given directly and not labeled by actions. Note that $CTLK$ does not refer to actions in the language.

1.5.2. Definition. An *interpreted system* is a tuple $\mathcal{S} = (S, S_0, T, \sim, \pi)$ where S is a set of states, $S_0 \subseteq S$ is a set of initial states, $T \colon S \to \mathcal{P}(S)$ is a transition function, $\sim$ is a family of equivalence relations $\sim_i$ for each agent i and π is a valuation function $\pi \colon S \to \mathcal{P}(V)$.

A path through a system $\mathcal{S}$ is a maximal sequence $\sigma = g_0 g_1 \ldots$ of states such that for all $k \geq 0$ we have $T(g_k) \ni g_{k+1}$.

We write $\mathsf{paths}(g)$ for the set of all paths starting at g. For any path $\sigma = g_0 g_1 \ldots$ we denote its kth state by $\sigma(k) := g_k$.

1.5.3. Definition. We interpret $\mathcal{L}_T$ on an interpreted system with the standard semantics for boolean operators as follows:

1. $(\mathcal{S}, g) \vDash K_i\varphi$ iff for all $g' \in G$, if $g \sim_i g'$, then $(\mathcal{S}, g') \vDash \varphi$.

2. $(\mathcal{S}, g) \vDash C_\Delta\varphi$ iff for all $g' \in W$, if $w \sim_\Delta w'$, then $(\mathcal{S}, g') \vDash \varphi$ where $\sim_\Delta := \left(\bigcup_{i\in\Delta} \sim_i\right)^*$.

3. $(\mathcal{S}, g) \vDash AX\varphi$ iff for all $\sigma \in \mathsf{paths}(g)$ we have $\mathcal{S}, \sigma(1) \vDash \varphi$.

4. $(\mathcal{S}, g) \vDash AG\varphi$ iff for all $\sigma \in \mathsf{paths}(g)$, for all k we have $\mathcal{S}, \sigma(k) \vDash \varphi$.

5. $(\mathcal{S}, g) \vDash A(\varphi U \psi)$ iff for all $\sigma \in \mathsf{paths}(g)$, there is a k such that $\mathcal{S}, \sigma(k) \vDash \psi$ and for all j such that $0 \leq j \leq k$ we have $\mathcal{S}, \sigma(j) \vDash \varphi$.

The avid reader will immediately wonder what the connections between DEL and ETL are. Do they describe the same sort of agents, situations and protocols? Can we translate back and forth between them? These questions have been partially answered. We summarize some results connecting the dynamic and the temporal approach in the next section.

1.6 Comparing Dynamic and Temporal Logics

The main difference between temporal and dynamic epistemic logics is how they model actions and time. In temporal logics time is represented *inside* a model, with a transition function. In DEL in contrast, time *changes* the model and is something outside of it. Whereas in temporal logics the model already contains the information about all possible actions, for example as labels for the transition function, in DEL the actions are in action models to be applied or formulas to be evaluated.

Still, from a third perspective, these are merely two different ways to talk about the same thing and we can directly connect a DEL model to a temporal model as follows. Consider an initial Kripke model for DEL and some set of actions, for example different public announcements that could be made. We let the Kripke model be the root of a tree and add an edge for each action (e.g. announcement) that can be made, leading to a new Kripke model. The result is called “DEL-induced model” in the literature [Ben+09] or also the resulting “tree” or “forest”, depending on whether we start with one or multiple DEL Kripke models as roots. Figure 1.5 illustrates this idea.

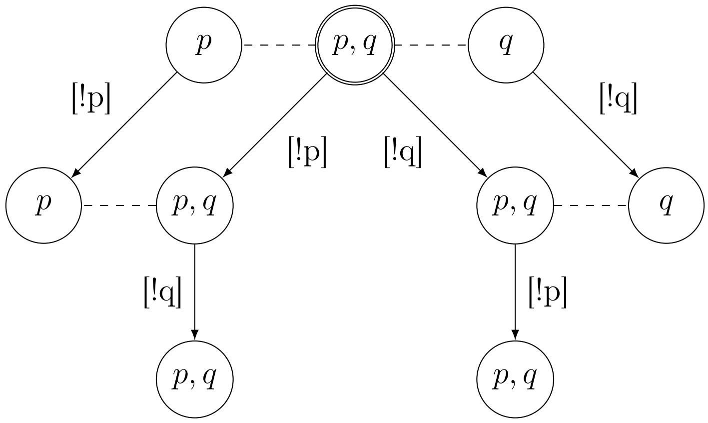


Figure 1.5: Induced tree starting from a DEL model.

We now expect an equivalence of some sort between an original DEL model and the induced tree, if we consider the announcement-arrows to be the time relation for a temporal model and give a translation of DEL formulas to ETL formulas. Additionally, we might wonder which temporal models can be generated in this way. When is there a Kripke model for DEL which generates a given temporal model or a tree that is isomorphic to it?

The connection can be made precise by the following theorem which was first shown in [Ben+09, p. 505]. For simplicity we state and use the simpler formulation from [BS15, Section 7.7].

1.6.1. THEOREM. *For ETL models $\mathcal{H}$ the following two are equivalent:*

(a) $\mathcal{H}$ is isomorphic to some DEL-induced model $\mathsf{Forest}(M, \sigma)$

(b) $\mathcal{H}$ satisfies Perfect Recall [and thereby Synchronicity], Uniform No Miracles and Definable Executability.

Given this result, we can use DEL as an alternative to the synchronous, perfect recall and no miracles fragment of ETL. To really use it for model checking however, we also need to study how single models and formulas can be translated.

While the focus of [Ben+09; BS15] is on meta-logical properties such as an axiomatization of PAL with restrictions via ETL, we find a more application and model checking oriented comparison between ETL and DEL in [DHR13].

The authors start with a common problem for modeling the same scenario in DEL and ETL: To translate public announcements or other dynamic operators one needs to add additional variables "with values corresponding to unknown (i.e. before the announcement is made), and true (after a truthful announcement)" [DHR13]. This was done more or less ad hoc for concrete examples before, for example in [Dit+06], but the syntactic translation and semantic transformation given in [DHR13] is the first systematic approach. We use these translations in Section 4.4 where we compare the performance of dynamic and temporal model checking.

1.7 Model Checking

The model checking problem is easy to state: Given a model and a formula, is the formula true in this model? More precisely, in our case: Given a pointed Kripke model $(\mathcal{M}, w)$ and a formula $\varphi \in \mathcal{L}$, do we have $\mathcal{M}, w \vDash \varphi$ or not? To a pure mathematical logician, this is possibly the most boring task or question one can ask about a logic: If the logic is defined properly, we can simply go through the semantics definition for $\vDash$ to answer the question. We might have to recurse a number of times, but for most logics this number is bounded by the size of the formula and seems straightforward. So what is the problem and why is model checking so difficult that it has become its own field of research?

Model checking is hard because we have to consider concrete data structures. Implementing standard logical semantics naively would mean that we explicitly spell out and list all worlds of a Kripke model, for example in a lookup table. But already for toy examples like the muddy children the number of worlds is exponential in the number of agents and propositions — it eventually becomes so large that the models no longer fit into the memory of our computers. This is known as the *state explosion problem.*

A solution to this problem appeared on the horizon when Randal Bryant presented Binary Decision Diagrams [Bry86] which we introduce below in Section 1.9. This new representation of boolean functions and circuits quickly led to *symbolic* model checking, starting with the seminal [Bur+90] and [CGL94]. In contrast to explicit methods, the idea here is to work with a symbolic representation of the model. A good description of a model should be compact, but still allow us to evaluate all the formulas we are interested in. Hence we should not lose any relevant information by moving to a symbolic representation. In a Kripke model for example, names or symbols referring to specific worlds are not relevant, but the valuation function is needed because it influences the interpretation of formulas.

Model checking is often seen as an alternative to theorem proving: Suppose we have a description of a system — a circuit, a protocol, a machine or a process — and a specification which properties this system should have. If both the system and the specification can be formalized in the same logical language, say as φ and ψ, then we can answer the question whether the system fulfills the specification by proving or disproving the implication: Is $\varphi \to \psi$ provable? However, theorem proving in general is not fully mechanical but involves creativity or heuristics. In the worst case, like for first-order logic, it can even be undecidable. In contrast, model checking does not need heuristics and is fully automated. Since the 1980s it has become the standard technique for formal verification. Given a model $\mathcal{M}$ of our system we check that it satisfies a specification: Does $\mathcal{M} \vDash \varphi$ hold?

For a thorough introduction to the field of model checking, see the classic [CGP99] or the new [Cla+18]. We also note that model checking is only one of the decision problems associated with every logic. In model checking we only ask whether a given formula holds in a given model. It has to be distinguished from various other tasks: *Theorem proving*, where as mentioned above the goal is to prove that a given formula is valid, i.e. true in all models; *Proof checking*, where such a proof is part of the input the goal is to verify a given proof; *Satisfiability*, which asks whether there is a model in which the given formula is true; and finally *model generation*, where one additionally has to return a witness model in which the given formula is true.

Corresponding to this landscape of decision problems is a plethora of automated tools, some of which only solve a specific task, others also combining a model checker, theorem prover or satisfiability solver in the same program.

More background on the comparison and competition between model checking and theorem proving can be found in the manifesto [HV91].

In the next section we present standard representation methods for sets of possible worlds and relations on them. Our main goal in the subsequent chapters is then to adapt and apply these techniques to Dynamic Epistemic Logic. In particular we are interested in a *direct* symbolic representation and semantics for DEL, which is both more mathematically interesting and more efficient than translating to a temporal logic in order to use existing symbolic methods afterwards.

1.8 Symbolic Representation

One of the best ways to tackle the state explosion problem is the observation, that even though we might be dealing with a huge model, we do not actually need the whole model to check a given formula. That is, we rarely have to look up the valuation function at all possible worlds and follow all epistemic relations to decide a formula. For example, to check $\neg K_i p$, it is enough to find one i-reachable world where p is false and we can stop as soon as we found one. Moreover, the truth of this formula does not depend on how large the model is and which other propositions the valuation function covers.

The goal of symbolic representation therefore is to describe a model in a more compact way and only unpack those parts of the description which matter for the formula we want to check.

In this section we will explain three principal ideas which allow us to symbolically represent worlds, equivalence relations over them and finally arbitrary relations. Our general motto is: *Make laws, not lists!* Do not spell out the model explicitly but summarize it with a rule to check whether something is part of the model or not. Intuitively, instead of "these are the worlds and relations" we say "this is how we decide whether a world exists in the model and how we decide whether two worlds are connected".

We start with a symbolic representation for *sets of possible worlds*: If we have unique valuations then we can identify worlds with the set of propositions that are true at them.

1.8.1. Definition. Suppose we have a finite vocabulary V, a set of possible worlds W and a valuation function $\pi\colon W \to \mathcal{P}(V)$ which is injective, i.e. all the valuations are different. A boolean formula $\theta \in \mathcal{L}_B(V)$ is a *symbolic encoding* of W iff for all $s \subseteq V$ we have:

$$s \vDash \theta \iff \exists w \in W : s = \pi(w)$$

Whenever π is injective, a symbolic encoding can be computed as follows.

1.8.2. Fact. Recall the "out of" abbreviation $\sqsubseteq$ from Definition 1.0.1. Given V, W and π as in Definition 1.8.1, the formula

$$\theta := \bigvee_{w \in W} (\pi(w) \sqsubseteq V)$$

and all formulas equivalent to θ are symbolic encodings of W.

1.8.3. Example. Consider the vocabulary $\{p, q\}$. Suppose we want to encode the set of worlds $W = \{0, 1, 2\}$ with the valuation function π saying $\pi(0) := \{p\}$, $\pi(1) := \{q\}$ and $\pi(2) := \{p, q\}$. Then we can use $\theta := p \vee q \in \mathcal{L}_B(\{p, q\})$ and identify W with the set $\{s \subseteq \{p, q\} \mid s \vDash p \vee q\}$.

Not all valuation functions are injective. However, there is a simple trick to obtain symbolic encodings for sets of possible worlds with non-unique valuations: We just add additional propositions to distinguish the worlds.

1.8.4. Example. Again consider the vocabulary $\{p, q\}$. Suppose we want to encode the set of worlds $W = \{0, 1, 2\}$ with the valuation function π saying $\pi(0) := \{p\}$, $\pi(1) := \{p\}$ and $\pi(2) := \{p, q\}$. Note that π is not injective because $\pi(0) = \pi(1)$. But we can lift π to a bigger vocabulary $\{p, q, r\}$ where r is fresh. Let $\pi'(0) := \{p\}$, $\pi'(1) := \{p, r\}$ and $\pi'(2) := \{p, q\}$. Then $p \wedge \neg(q \wedge r) \in \mathcal{L}_B(\{p, q, r\})$ is a symbolic encoding of W and π'.

With this symbolic encoding we lose the *names* of the worlds: V and θ no longer mention the symbols we used to refer to individual possible worlds. This is perfectly fine, because after all those were just names to talk about our model and not part of the model itself. Anything we say about or do with a Kripke model does not depend on how a world is called.

Given a boolean function which describes a set of worlds, we can also try to describe a *relation* over this set by directly referring to the propositional variables. The goal again is to save memory and avoid an explicit set or list of pairs.

For equivalence relations we can already do much better by using partitions. For example, the relation $\{(0, 0), (0, 1), (1, 1), (1, 0), (2, 2)\}$ can be represented by $[[0, 1], [2]]$. This representation has been implemented in [Eij14c] and is used in the model checker DEMO-S5, a variant of DEMO which is optimized for equivalence relations [Eij14a]. However, we still mention all worlds explicitly and thus might have to store a long list, even if the relation actually contains not much or no information at all. For example, if R is the total relation over W, the size of its representation as a partition is still $|W|$.

A truly symbolic way to encode equivalence relations over sets of worlds with unique valuations is to use *observational variables*. They are also used in model checking temporal epistemic logics and determine the *local state* of agents [Bur+90; LQR15]. The way we use observational variables is inspired by the problem-specific approaches in [Luo+08; MS04] and the model checker MCTK for temporal logics [SSL07].

The key idea is to describe an equivalence relation over possible worlds by a subset of V: Two worlds are related if the valuation function agrees at them on this subset. Intuitively, an agent with knowledge described by this subset can only distinguish worlds if there is a difference she can *observe*.

1.8.5. DEFINITION. Suppose we have V, W, π and θ as in Definition 1.8.1. We say that *a set of observational variables* $O \subseteq V$ *encodes* an equivalence relation $\sim$ over the set of worlds encoded by θ iff we have for all worlds s and t that $s \sim t \iff O \cap s = O \cap t$.

1.8.6. EXAMPLE. Figure 1.6 shows a Kripke model based on the set of worlds W and the valuation function π from Example 1.8.3. Again we leave out the reflexive arrows and use undirected edges to draw equivalence relations. We can describe the relations in this model in three different ways:

- Explicitly spelling out the relations as lists or sets of pairs:

$$R_{\text{Alice}} = \{(0,0),(1,1),(2,2),(1,2),(2,1)\}$$

$$R_{\text{Bob}} = \{(0,0),(1,1),(2,2),(2,0),(0,2)\}$$

- As partitions: $R_{\text{Alice}} = [[2,1],[0]]$ and $R_{\text{Bob}} = [[2,0],[1]]$
- With observational variables: $O_{\text{Alice}} = \{q\}$ and $O_{\text{Bob}} = \{p\}$

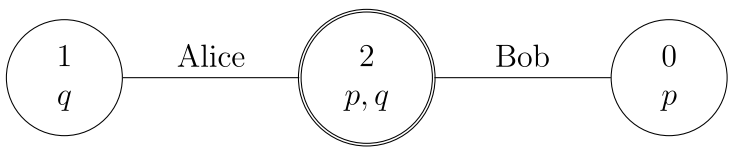


Figure 1.6: Alice observes q and Bob observes p.

It becomes clear that observational variables provide a concise way to represent the epistemic state of an agent. Unfortunately, we cannot represent all relations in this way. First, it is clear that only equivalence relations can be encoded like this. But second, not even all equivalence relations over distinctly valuated worlds are representable with observational variables, as the following example shows.

1.8.7. Example. In the left part of Figure 1.7 the knowledge of Alice and Bob is given by two equivalence relations. Note that we omit the reflexive arrows as usual and the edges are not directed because of symmetry. It is easy to see that $O_{\text{Bob}} = \{p\}$ encodes the knowledge of Bob. But the knowledge of Alice cannot be described by saying which subset of the vocabulary $V = \{p, q\}$ she observes.

We would want to say that she observes $p \wedge q$, but to encode this with observational variables we have to add a new variable r to distinguish the two equivalence classes of Alice, as shown in the right part of Figure 1.7.

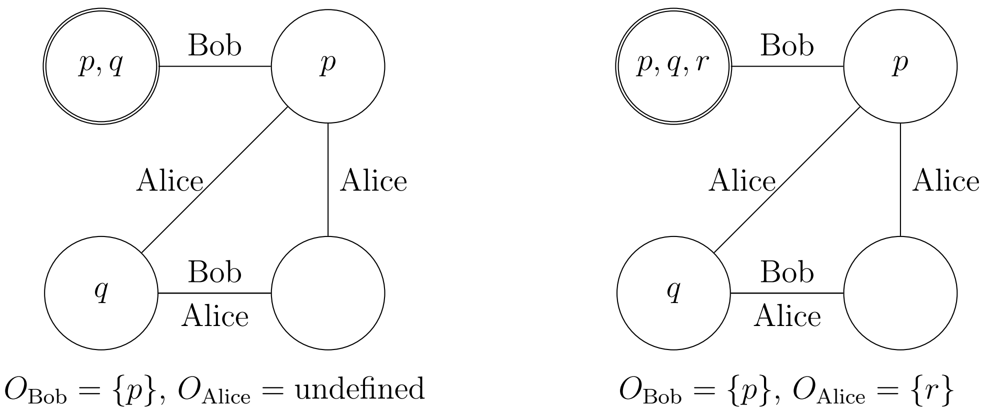


Figure 1.7: Observational variables need to be added.

We therefore introduce yet another, more general way to encode relations. A relation over $\mathcal{P}(V)$ can be represented as a boolean formula over a double vocabulary $V \cup V'$. Just like observational variables, this boolean encoding of relations has been widely used for model checking temporal logics [CGL94].

Suppose again that all states satisfy a unique set of propositions. Then any relation over states is also a relation over sets of propositions. To encode these relations we use the same idea as [GR02] where BDDs have also been used to model belief revision. We replace observational variables $O \subseteq V$ with a boolean formula $\Omega \in \mathcal{L}_B(V \cup V')$. This formula uses a double vocabulary: Suppose our original vocabulary is the set $V = \{p, q\}$, then such an Ω is a boolean formula over the twice as large vocabulary $\{p, q, p', q'\}$. The formula Ω is true exactly for those *pairs of assignments* that are connected by the relation. For example, to represent an edge from $\{p, q\}$ to $\{q\}$, the assignment $\{p, q, q'\}$ should makes Ω true. The opposite edge corresponds to $\{q, p', q'\}$. The following definition makes this precise. For more details, see also [CGP99, Section 5.2].

1.8.8. Definition. If s is an assignment for V, then s' is the corresponding assignment for V'. For example, $\{p_1, p_3\}' = \{p_1', p_3'\}$. If φ is a formula, $(\varphi)'$ is the result of priming all propositions. For example, $(p_1 \to \neg p_2)' = (p_1' \to \neg p_2')$. If s and t' are assignments for V and V' respectively such that $V \cap V' = \varnothing$ and φ is a formula over $V \cup V'$, we also write $st' \vDash \varphi$ instead of $s \cup t' \vDash \varphi$. Suppose we have a relation R on $\mathcal{P}(V)$. A boolean formula $\Omega \in \mathcal{L}_B(V \cup V')$ is a *symbolic encoding* of R iff we have for all $s, t \subseteq V$ that Rst iff $st' \vDash \Omega$.

1.8.9. Fact. Suppose we have V and R as in Definition 1.8.8. Then the following $\Phi(R) \in \mathcal{L}_B(V \cup V')$ and any equivalent boolean formula is a symbolic encoding of R:

$$\Phi(R) := \bigvee_{(s,t)\in R} ((s \sqsubseteq \mathsf{V}) \wedge (t \sqsubseteq \mathsf{V})')$$

This encoding of relations as boolean functions plays an important role in the following chapters. Hence we illustrate it with two examples before moving on.

1.8.10. Example. Figures 1.8 to 1.10 show an example from [GR02, p. 136] how to go from a relation to its encoding as a boolean function. We start with a relation R over states with the vocabulary $V = \{p_1, p_2\}$. That is, $R \subseteq (\mathcal{P}(\{p_1, p_2\}))^2$. The formula $\Phi(R)$ shown in Figure 1.9 is a disjunction with one disjunct for each edge in the graph of R. We use V for the source and V' for the target.

For example, the second disjunct $\neg p_1 \wedge \neg p_2 \wedge \neg p_1' \wedge p_2'$ is for the edge from the top left state $\varnothing$ to the top right state $\{p_2\}$. But there is no edge from the top right to the bottom right state, hence $\neg p_1 \wedge p_2 \wedge p_1' \wedge p_2'$ is not a disjunct of $\Phi(R)$.

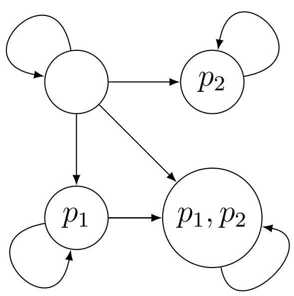


Figure 1.8: Relation R.

$$\begin{aligned}
& (\neg p_1 \wedge \neg p_2 \wedge \neg p_1' \wedge \neg p_2') \\
\vee\ & (\neg p_1 \wedge \neg p_2 \wedge \neg p_1' \wedge p_2') \\
\vee\ & (\neg p_1 \wedge \neg p_2 \wedge p_1' \wedge \neg p_2') \\
\vee\ & (\neg p_1 \wedge \neg p_2 \wedge p_1' \wedge p_2') \\
\vee\ & (\neg p_1 \wedge p_2 \wedge \neg p_1' \wedge p_2') \\
\vee\ & (p_1 \wedge \neg p_2 \wedge p_1' \wedge p_2') \\
\vee\ & (p_1 \wedge \neg p_2 \wedge p_1' \wedge \neg p_2') \\
\vee\ & (p_1 \wedge p_2 \wedge p_1' \wedge p_2')
\end{aligned}$$

Figure 1.9: $\Phi(R)$.

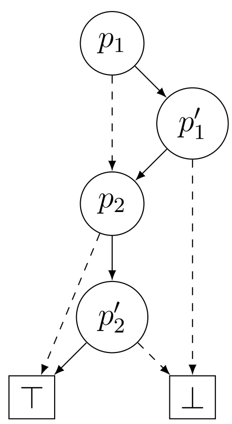


Figure 1.10: $\mathsf{Bdd}(\Phi(R))$.

In our implementation the formula $\Phi(R)$ is never constructed explicitly. Instead we represent it using the Binary Decision Diagram (BDD) as shown in Figure 1.10 and to be explained in Section 1.9.

1.8.11. Example. Consider the equivalence relation describing the knowledge of Alice in the left model of Figure 1.7. As noted above, this relation can not be encoded by saying which atomic propositions Alice observes, but instead we would like to say that she observes whether $p \land q$ is true. Switching from observational variables to observation laws allows us to do exactly that.

We could use the same way as in Example 1.8.10 to obtain the formula $\Phi(R_{\text{Alice}})$: For each directed edge, add a disjunct which describes the starting world in V and the reached world in V'. Note that in the left part of Figure 1.7 we have three undirected edges for Alice. Moreover, there are four identity arrows which we did not draw. In total we would therefore get ten disjuncts.

But in this case there is an intuitive shortcut: $\Phi(R_{\text{Alice}}) \equiv (p \land q) \leftrightarrow (p' \land q')$. Two worlds are indistinguishable for Alice if both satisfy $p \land q$ or both do not — she observes $p \land q$. There is an edge from one world to another iff their combined boolean assignment satisfies $(p \land q) \leftrightarrow (p' \land q')$. For example, the diagonal edge from the bottom left to the top right world is represented by $\{q, p'\} \vDash (p \land q) \leftrightarrow (p' \land q')$. On the other hand, there is no edge from the top left to the top right world and indeed we have $\{p, q, p'\} \nvDash (p \land q) \leftrightarrow (p' \land q')$.

To conclude this section, in Table 1.2 we give an overview how different elements of a Kripke model can be encoded symbolically. In the next chapter we will combine all of these methods to encode entire Kripke models. We will also see that the symbolic representations preserve enough information such that we can still evaluate the same languages on symbolic encodings of our models.

Explicit	Symbolic
set of worlds W and valuation π	vocabulary V and formula $\theta \in \mathcal{L}_B(V)$
equivalence relation $\sim \subseteq W \times W$	observational variables $O \subseteq V$
arbitrary relation $R \subseteq W \times W$	observational law $\Phi(R) \in \mathcal{L}_B(V \cup V')$

Table 1.2: Overview of symbolic representation methods.

1.9 Binary Decision Diagrams

We have seen in the previous sections that instead of listing all worlds and relations of a Kripke model explicitly we can encode them with boolean formulas of propositional logic. But why should these formulas be easier to handle than lists of possible worlds? The answer is that we will not actually deal with boolean formulas but instead directly work with the boolean function they represent.

Boolean functions can be represented nicely using *Binary Decision Diagrams*, which have been called "one of the only really fundamental data structures that came out in the last twenty-five years" [Knu08]. They were first presented by Randal Bryant in [Bry86] and have since been applied to a plethora of problems throughout computer science.

1.9.1. Definition. A *binary decision diagram* for a vocabulary V is a directed acyclic graph where non-terminal nodes are from V with two outgoing edges and terminal nodes are $\top$ or $\bot$. Outgoing edges are distinguished by drawing them dashed or solid. The *size* $|B|$ of a binary decision diagram B is its number of nodes. A binary decision diagram is *ordered* according to a total order $<$ of V iff for any edge from a node p to a node q we have $p < q$. A binary decision diagram is *reduced* iff it does not contain two subgraphs which are isomorphic as labeled graphs. By the abbreviation *BDD* we always mean an ordered and reduced binary decision diagram.

We read a BDD from top to bottom. At every non-terminal node we need to provide a truth value for the proposition this node asks for. If it is true we follow the solid outgoing arrow, otherwise the dashed one. Finally we reach a terminal node telling us the value of the boolean function encoded by this BDD.

1.9.2. Example. Consider the boolean function given by the formula $\neg(p_1 \wedge \neg p_2) \to p_3$. Figure 1.11 shows a full decision tree for this function and the BDD obtained by identifying all isomorphic subgraphs.

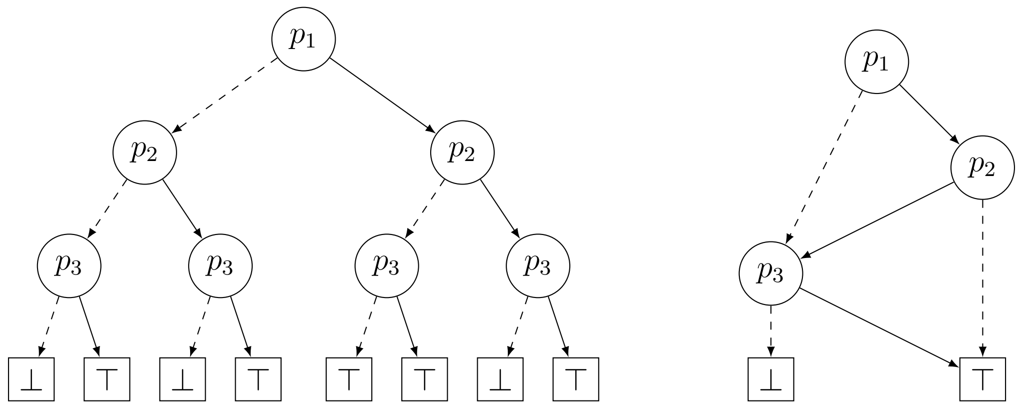


Figure 1.11: Full decision tree and BDD of $\neg(p_1 \wedge \neg p_2) \to p_3$.

Even though BDDs are more compact, we do not lose any information. To check whether a given assignment satisfies the function represented by this BDD, we start at the root and follow the arrows as follows: If the variable at the current node is true according to the given assignment, go along the solid arrow, otherwise the dashed one.

We can check that $\{p_1, p_3\} \vDash \neg(p_1 \wedge \neg p_2) \to p_3$ using the BDD on the right in Figure 1.11: We start at the top node p_1 which is true, hence we follow the solid arrow to a node which asks for p_2. This is false in our assignment, thus we now follow the dashed arrow to the result $\top$ which means that $\{p_1, p_3\}$ satisfies the boolean function. Note that the BDD did not even ask about p_3. This reflects that we also have $\{p_1\} \vDash \neg(p_1 \wedge \neg p_2) \to p_3$.

BDDs have several advantages over truth tables, the classical explicit representation of boolean functions. In many cases BDDs are less redundant and thus smaller than a corresponding truth table. While the worst-case size is the same as for truth-tables or full decision trees, in many practical applications the boolean functions have an additional structure and using a good variable ordering leads to compact BDDs. Probably the best feature of BDDs however, is that they are *canonical* in the following sense.

1.9.3. THEOREM. *Given a total order on the propositional variables there is exactly one reduced and ordered binary decision diagram for each boolean function.*

For a proof, see the classic [Bry86].

1.9.4. DEFINITION. For any formula $\varphi \in \mathcal{L}_B$ we also call *the* BDD of the boolean function given by φ the BDD of φ itself and denote it by $\mathsf{Bdd}(\varphi)$.

Theorem 1.9.3 means that two formulas are equivalent if and only if their BDDs are identical. In particular, once we have the BDD of a formula, it is trivial to check whether it is a tautology or a contradiction: A formula is a tautology if and only if its BDD consists of a single terminal node $\top$ and it is a contradiction if and only if its BDD is the single terminal node $\bot$.

Additionally, BDDs can be manipulated efficiently. Given BDDs of φ and ψ we can compute the BDD of $\varphi \wedge \psi$, $\varphi \to \psi$ and other boolean combinations as follows: Given $\mathsf{Bdd}(\varphi)$ and $\mathsf{Bdd}(\psi)$ and a binary boolean operator $\star$, we can compute a new ordered binary decision diagram for $\varphi \star \psi$ essentially by traversing the two given BDDs in parallel and then reduce the result to obtain $\mathsf{Bdd}(\varphi \star \psi)$. Both the intermediate result and hence $|\mathsf{Bdd}(\varphi \star \psi)|$ are bounded by $|\mathsf{Bdd}(\varphi)| \cdot |\mathsf{Bdd}(\psi)|$. Moreover, this can be done in running time of order $|\mathsf{Bdd}(\varphi \star \psi)|$ as shown in [Knu11, p. 219].

To summarize this, we can say that BDDs are *hard to construct, but easy to use.* Generating the BDD for a given formula is as hard as the boolean satisfiability problem (i.e. NP-complete), but once we have one or more BDDs then it is easy to combine or evaluate them. It therefore makes sense in an implementation to translate boolean formulas to BDDs as early as possible.

For an in-depth introduction to BDDs we refer the interested reader to the original [Bry86], the classic [Knu11, p. 202-280] and the entertaining [Knu08].

When presenting our theoretical framework in the next chapter we will write down many boolean formulas, but in the implementation later on those will be replaced with BDDs representing the boolean function. We are therefore sloppy on purpose and will from now on identify a boolean formula with the boolean *function* that it represents. That is, we simply write φ also if the implementation will use $\mathsf{Bdd}(\varphi)$. Additionally, we will consider two formulas to be the same if their represented functions are the same. In particular, for functions on $\mathcal{L}_B$, we will also call a formula a fixpoint if it is a fixpoint regarding semantic equivalence.

Chapter 2

Symbolic Model Checking DEL

It is not a good idea to name a state after its valuation.

[DHK07, page 22]

We now present our main framework: a symbolic representation of Kripke models and updates which provides symbolic model checking for Dynamic Epistemic Logic. Our goal is to connect the two worlds of symbolic model checking and DEL in order to gain new insights on both sides.

On one side, there are many frameworks for symbolic model checking interpreted systems using temporal logics [SSL07; LQR15]. On the other hand, there are explicit model checkers for Dynamic Epistemic Logic (DEL) [Eij07; Eij14a]. The latter provide superior usability as they allow specification in dynamic languages directly, but inferior performance. This reflects that the cradle of DEL was logic and philosophy, not computer science: For logicians, models are just abstract mathematical objects whose size does not matter.

Our framework can be applied to different variants of DEL, hence we will build it step by step: First we only deal with $\mathsf{S5}$ Public Announcement Logic, then we extend our ideas to cover all epistemic change with action models, still in the S5 setting. After that we generalize to non-$\mathsf{S5}$ logics for belief. Finally, we also encode factual change with a symbolic representation for action models with postconditions.

Table 2.1 gives an overview of various flavors of DEL and shows in which sections of this chapter we cover them. For each logic we first define symbolic

	Public Announcements	Action Models	with factual change
S5:	Sections 2.2 to 2.4	Section 2.5	Section 2.8
General:	Section 2.6	Section 2.7	Sections 2.8 and 2.9

Table 2.1: Different flavors of DEL and where we discuss them.

analogues of the standard semantics and then provide translations to show that they are equivalent. To be less repetitive, we only give proofs for the most simple case of S5 PAL and the most general setting with factual change.

2.1 Related Work

Existing work on how to optimize the model checking performance of DEL mainly focuses on specific examples, such as the Dining Cryptographers [MS04], the Sum and Product riddle [Luo+08] or Russian Cards [Dit+06]. Given these successful specific approaches, a general method for symbolic model checking full DEL is desirable. A first step is [SSL07] which presents symbolic model checking for temporal logics of knowledge. Based on [SLZ04], it gives us a boolean translation of the knowledge operators in S5, but does not cover announcements or other dynamics. We extend these ideas to non-S5 logics with dynamic operators.

An alternative representation for Kripke models and action models was recently developed in [CS17]. Their so-called succinct models describe sets of worlds symbolically with boolean formulas as defined in Definition 1.8.1. This is the same encoding as we use for our structures, but epistemic relations and factual change are encoded differently in succinct models, with *mental programs* instead of observational variables or boolean formulas as introduced in Section 1.8. Notably, model checking DEL is still in PSPACE when models and actions are represented in this succinct way. No complexity is known for our structures and transformers so far, but we expect it to be the same as for succinct models and actions.

Another related line of work also focuses on the idea of *observation* and started in [HLM15]. The authors also assume that all knowledge is encoded by which agents can observe which propositional variables. They also add propositional variables to encode meta-observations of the form "agent a observes whether agent b observes proposition p" etc. This allows, for example, a more intuitive modeling of gossip [HM17]. The encoding allows one to eliminate knowledge operators in all formulas, similarly to the reduction we have on our structures. An important difference to our framework however, is that we do not add all such extra propositional variables to the language. Instead we only add a few fresh propositional variables to individual models in order to make valuations unique and all epistemic relations describable. Our added propositions are not part of the original language to be interpreted on our structures. We merely give a new representation for Kripke models and do not introduce a new logic, whereas the "Poor Man's Epistemic Logic" from [HLM15] has a very different axiomatization than standard DEL.

Finally, our knowledge structures are similar in spirit to the "hypercubes" from [LMR00] which represent interpreted systems. As discussed in Section 1.5, this means hypercubes can only be used for languages with temporal but not with dynamic operators for events.

2.2 Knowledge Structures

While the Kripke semantics from the previous chapter is standard in logic, it cannot serve directly as an input to current sophisticated model-checking techniques. For this purpose, in this section we introduce a new format, *knowledge structures*. Their main advantage is that they also allow knowledge and results of announcements to be computed via purely boolean operations.

2.2.1. Definition. Suppose we have a set I of n agents. A *knowledge structure* is a tuple $\mathcal{F} = (V, \theta, O)$ where V is a finite set of propositional variables, $\theta \in \mathcal{L}_B(V)$ is a boolean formula over V and O is a family of subsets of V indexed by agents, such that $O_i \subseteq V$ for each agent i.

The set V is the *vocabulary* of $\mathcal{F}$ and the formula θ is the *state law* of $\mathcal{F}$. Crucially, θ comes from $\mathcal{L}_B(V)$ and thus is only allowed to contain boolean operators. The variables in O_i are called agent i's *observable variables*. We also write $(V, \theta, O_1, \ldots, O_n)$ for (V, θ, O).

Recall from Definition 1.0.2 that we identify a boolean assignment with the subset $s \subseteq V$ of atomic propositions that it makes true. A boolean assignment for V that satisfies θ is called a *state* of the structure $\mathcal{F}$, i.e. the set of states is represented symbolically as in Definition 1.8.1. Any knowledge structure only has finitely many states. Given a state s of $\mathcal{F}$, we call $(\mathcal{F}, s)$ a *scene* and define the *local state* of an agent i at s as $s \cap O_i$.

We prepare our interpretation of common knowledge as follows. Given a knowledge structure (V, θ, O) and a set of agents Δ, let $\mathcal{E}_\Delta$ be the following relation on states of $\mathcal{F}$ and let $\mathcal{E}_\Delta^*$ denote its transitive closure.

$$(s, t) \in \mathcal{E}_\Delta \text{ iff there exists an } i \in \Delta \text{ with } s \cap O_i = t \cap O_i$$

2.2.2. Example. Consider this knowledge structure:

$$\mathcal{F} := (V = \{p, q\},\ \theta = p \to q,\ O_1 = \{p\},\ O_2 = \{q\})$$

Here the vocabulary consists of two propositions. The state law is $p \to q$, hence the states of $\mathcal{F}$ are the three assignments satisfying that formula. To simplify notation we write assignments as the set of propositions they make true. The states of $\mathcal{F}$ are thus $\varnothing$, $\{q\}$ and $\{p, q\}$. Moreover, $\mathcal{F}$ describes two agents who each observe one of the propositions. Intuitively, this can be understood as information about knowing *whether*: Agent 1 knows whether p is true and agent 2 knows whether q is true. We also use this knowledge structure in Example 2.2.4 below and compute an equivalent Kripke model in Example 2.4.4.

We now interpret the language of public announcement logic, $\mathcal{L}_P(V)$ from Definition 1.2.1, on knowledge structures. Definitions 2.2.3 and 2.2.6 refer to each other and therefore run in parallel, both proceeding inductively by the structure of φ.

2.2.3. Definition. Semantics for $\mathcal{L}_P(V)$ *on scenes* are defined inductively as follows.

1. $(\mathcal{F}, s) \vDash \top$ always holds.
2. $(\mathcal{F}, s) \vDash p$ iff $s \vDash p$.
3. $(\mathcal{F}, s) \vDash \neg\varphi$ iff not $(\mathcal{F}, s) \vDash \varphi$
4. $(\mathcal{F}, s) \vDash \varphi \wedge \psi$ iff $(\mathcal{F}, s) \vDash \varphi$ and $(\mathcal{F}, s) \vDash \psi$
5. $(\mathcal{F}, s) \vDash K_i\varphi$ iff for all states t of $\mathcal{F}$, if $s \cap O_i = t \cap O_i$, then $(\mathcal{F}, t) \vDash \varphi$.
6. $(\mathcal{F}, s) \vDash C_\Delta\varphi$ iff for all states t of $\mathcal{F}$, if $(s, t) \in \mathcal{E}_\Delta^*$, then $(\mathcal{F}, t) \vDash \varphi$.
7. $(\mathcal{F}, s) \vDash [!\psi]\varphi$ iff $(\mathcal{F}, s) \vDash \psi$ implies $(\mathcal{F}^\psi, s) \vDash \varphi$. Here the new structure after the announcement is given by
$$\mathcal{F}^\psi := (V, \theta \wedge \|\psi\|_\mathcal{F}, O)$$
where $\|\psi\|_\mathcal{F} \in \mathcal{L}_B(V)$ is from Definition 2.2.6 and notably $\theta \wedge \|\psi\|_\mathcal{F}$ is again a boolean formula.

We write $(\mathcal{F}, s) \equiv_V (\mathcal{F}', s')$ iff these two scenes agree on all formulas. If we have $(\mathcal{F}, s) \vDash \varphi$ for all states s of $\mathcal{F}$, then we say that φ is *valid on* $\mathcal{F}$ and write $\mathcal{F} \vDash \varphi$.

Before defining boolean equivalents of formulas, we can already explain some connections between the Kripke semantics in Definition 1.2.2 and Definition 2.2.3. The semantics of the boolean connectives are the same. For the knowledge operators, on Kripke models we use the accessibility relation R_i on worlds. On knowledge structures this is replaced with the condition $s \cap O_i = t \cap O_i$, inducing an equivalence relation on states. We can already guess that knowledge structures encode S5 Kripke models.

2.2.4. Example. Consider again the knowledge structure $\mathcal{F}$ from Example 2.2.2. We can easily check that $(\mathcal{F}, \varnothing) \vDash K_1\neg p$ holds: The only states t of $\mathcal{F}$ such that $\varnothing \cap O_1 = t \cap O_1$ are $\varnothing$ and $\{q\}$, and we have $(\mathcal{F}, \varnothing) \vDash \neg p$ and $(\mathcal{F}, \{q\}) \vDash \neg p$.

Similarly we can check that $(\mathcal{F}, \{p, q\}) \vDash K_1 q$: There is no state t other than $\{p, q\}$ such that $\{p, q\} \cap O_1 = t \cap O_1$, because the state law $\theta = p \to q$ rules out $\{p\}$. Intuitively, even though agent 1 does not observe q, at state $\{p, q\}$ she does observe that p is true and together with the state law $p \to q$ this implies q. In general, the state law of a knowledge structure is always valid on it and therefore common knowledge among all agents. In this case: $\mathcal{F} \vDash C_{\{1,2\}}(p \to q)$.

Our intuitive understanding of observational variables as knowing whether can now also be stated formally.

2.2.5. FACT. If $p \in O_i$ in a knowledge structure $\mathcal{F}$, then $\mathcal{F} \vDash K_i p \vee K_i \neg p$.

That is, any agent observing a proposition will know whether it is true. To illustrate this, note that in Example 2.2.4 we have $\mathcal{F} \vDash K_1 p \vee K_1 \neg p$ and $\mathcal{F} \vDash K_2 q \vee K_2 \neg q$.

The following definition of local boolean equivalents is the crucial ingredient that enables symbolic model checking on our structures.

2.2.6. DEFINITION. For any knowledge structure $\mathcal{F} = (V, \theta, O)$ and any formula $\varphi \in \mathcal{L}(V)$ we define its *local boolean translation* $\|\varphi\|_{\mathcal{F}}$ as follows.

1. For the true constant, let $\|\top\|_{\mathcal{F}} := \top$.

2. For atomic propositions, let $\|p\|_{\mathcal{F}} := p$.

3. For negation, let $\|\neg\psi\|_{\mathcal{F}} := \neg\|\psi\|_{\mathcal{F}}$.

4. For conjunction, let $\|\psi_1 \wedge \psi_2\|_{\mathcal{F}} := \|\psi_1\|_{\mathcal{F}} \wedge \|\psi_2\|_{\mathcal{F}}$.

5. For knowledge, let $\|K_i\psi\|_{\mathcal{F}} := \forall(V \setminus O_i)(\theta \rightarrow \|\psi\|_{\mathcal{F}})$.

6. For common knowledge, let $\|C_\Delta\psi\|_{\mathcal{F}} := \mathsf{gfp}\Lambda$ where Λ is the following operator in the lattice of boolean formulas modulo semantic equivalence $\mathcal{L}_B(V) \,/\!\equiv$ and $\mathsf{gfp}\Lambda$ denotes a representative of its greatest fixed point:

$$\Lambda(\alpha) := \|\psi\|_{\mathcal{F}} \wedge \bigwedge_{i \in \Delta} \forall(V \setminus O_i)(\theta \rightarrow \alpha)$$

7. For public announcements, let $\|[\psi]\xi\|_{\mathcal{F}} := \|\psi\|_{\mathcal{F}} \rightarrow \|\xi\|_{\mathcal{F}^\psi}$. where $\mathcal{F}^\psi$ is as given by Definition 2.2.3.

The translation of common knowledge $\|C_\Delta\psi\|_{\mathcal{F}}$ deserves some explanation: It is crucial that Λ is not a syntactic operator on plain formulas, because then it would not have a fixpoint — the formula would just become more and more complex. However, the lattice of boolean formulas modulo equivalence $\mathcal{L}_B(V) \,/\!\equiv$ is finite, because there are only finitely many boolean functions for the finite vocabulary V. Moreover, we can check that Λ is monotone. Hence its greatest fixpoint can be computed by starting with $\Lambda(\top)$ and then iterating Λ until we reach the first and thereby smallest k such that $\Lambda^k(\top) \equiv \Lambda^{k+1}(\top)$. Formally, the result of our translation needs to be a formula again and not an equivalence class thereof, hence we let $\mathsf{gfp}\Lambda$ be the representative of $\Lambda^k(\top)$ obtained by reading Λ as a syntactic operator. Any other representative would work as well. In practice, i.e. in Chapter 3, all computations will be done on Binary Decision Diagrams instead of boolean formulas.

2.2.7. EXAMPLE. Using the structure $\mathcal{F}$ from Example 2.2.2 we have:

$$\begin{array}{rcl}\|K_2(p \vee q)\|_{\mathcal{F}} & = & \forall (V \setminus O_2)(\theta \rightarrow \|p \vee q\|_{\mathcal{F}}) \\ & = & \forall p((p \rightarrow q) \rightarrow (p \vee q)) \\ & = & ((\top \rightarrow q) \rightarrow (\top \vee q)) \wedge ((\bot \rightarrow q) \rightarrow (\bot \vee q)) \\ & \equiv & (q \rightarrow \top) \wedge (\top \rightarrow q) \\ & \equiv & q\end{array}$$

One can check that indeed the formulas $K_2(p \vee q)$ and q are true at the same states of $\mathcal{F}$, namely $\{p, q\}$ and $\{q\}$. Note that we consider equivalent boolean formulas to be identical, so in particular we can ignore succinctness of DEL formulas and their translations, in line with the implementation in Chapter 3.

The next section contains more complex examples of this translation. Here it remains to show that the boolean translations are indeed locally equivalent.

2.2.8. THEOREM. *Definition 2.2.6 preserves and reflects truth. That is, for any formula φ and any scene $(\mathcal{F}, s)$ we have that $(\mathcal{F}, s) \vDash \varphi$ iff $s \vDash \|\varphi\|_{\mathcal{F}}$.*

Proof:
By induction on φ. The base case for atomic propositions is immediate. In the induction step, negation and conjunction are standard.

For the case of knowledge, so $\varphi = K_i\psi$, remember how we defined boolean quantification in Definition 1.0.3 and note the following equivalences:

$$\begin{array}{lll} & (\mathcal{F}, s) \vDash K_i\psi & \\ \iff & \forall t \text{ of } \mathcal{F} \text{ s.t. } s \cap O_i = t \cap O_i : (\mathcal{F}, t) \vDash \psi & \text{by Definition 2.2.3} \\ \iff & \forall t \text{ s.t. } t \vDash \theta \text{ and } s \cap O_i = t \cap O_i : (\mathcal{F}, t) \vDash \psi & \text{by Definition 2.2.1} \\ \iff & \forall t \text{ s.t. } s \cap O_i = t \cap O_i \text{ and } t \vDash \theta : t \vDash \|\psi\|_{\mathcal{F}} & \text{by induction hypothesis} \\ \iff & \forall t \text{ s.t. } s \cap O_i = t \cap O_i : t \vDash \theta \rightarrow \|\psi\|_{\mathcal{F}} & \\ \iff & s \vDash \forall (V \setminus O_i)(\theta \rightarrow \|\psi\|_{\mathcal{F}}) & \end{array}$$

For the common knowledge case $\varphi = C_\Delta\psi$, let Λ be the operator defined in as in Definition 2.2.6. Also let $\Lambda^0(\alpha) := \alpha$ and $\Lambda^{k+1}(\alpha) := \Lambda(\Lambda^k(\alpha))$.

For left to right, suppose $(\mathcal{F}, s) \vDash C_\Delta\psi$. Note that Λ is monotone but there are only finitely many boolean functions over V. Hence there is some m such that $\mathsf{gfp}\Lambda = \Lambda^m(\top)$. Therefore we can show $s \vDash \mathsf{gfp}\Lambda$ by proving $s \vDash \Lambda^m(\top)$ for all m. Suppose not, i.e. there is an m such that $s \nvDash \Lambda^m(\top)$. Then $s \nvDash \|\psi\|_{\mathcal{F}}$ or $s \nvDash \bigwedge_{i \in \Delta} \forall (V \setminus O_i)(\theta \rightarrow \Lambda^{m-1}(\top))$. The first is excluded by the induction hypothesis applied to $(\mathcal{F}, s) \vDash \psi$ which follows from $(\mathcal{F}, s) \vDash C_\Delta\psi$ by reflexivity. Hence there must be some $i \in \Delta$ and an assignment s_2 such that $s \cap O_i = s_2 \cap O_i$ and $s_2 \nvDash \theta \rightarrow \Lambda^{m-1}(\top)$. Then $s_2 \vDash \theta$, so s_2 is a state of $\mathcal{F}$, and $s_2 \nvDash \Lambda^{m-1}(\top)$. Spelling this out we have $s_2 \nvDash \|\psi\|_{\mathcal{F}}$ or $s_2 \nvDash \bigwedge_{i \in \Delta} \forall (V \setminus O_i)(\theta \rightarrow \Lambda^{m-2}(\top))$. Again the first case cannot be: s_2 is a state of $\mathcal{F}$ and by $s_1 \cap O_i = s_2 \cap O_i$ we have $(s, s_2) \in \mathcal{E}_\Delta$. Thus $(\mathcal{F}, s) \vDash C_\Delta\psi$ implies $(\mathcal{F}, s_2) \vDash \psi$ which by induction hypothesis gives $s_2 \vDash \|\psi\|_\mathcal{F}$. Iterating this we get an $\mathcal{E}_\Delta$-chain $s = s_1, \ldots, s_m$ such that $s_{1+k} \vDash \|\psi\|_\mathcal{F}$ and $s_{1+k} \nvDash \Lambda^{m-k}(\top)$ for all $k \in \{1, \ldots, m-1\}$. In particular $s_m \nvDash \Lambda(\top)$ and because $s_m \vDash \|\psi\|_\mathcal{F}$ we get $s_m \nvDash \top$. Contradiction! Hence $s \vDash \Lambda^m(\top)$ must hold for all m.

For right to left, suppose $s \vDash \mathsf{gfp}\Lambda$. Note that $\mathsf{gfp}\Lambda \to \Lambda^k(\top)$ is valid and thus we have $s \vDash \Lambda^k(\top)$ for any k. Fix any state t of $\mathcal{F}$ such that $(s, t) \in \mathcal{E}^*_\Delta$. We have to show $(\mathcal{F}, t) \vDash \psi$. By definition of $\mathcal{E}^*_\Delta$ there is a chain $s = s_1, \ldots, s_m = t$ and there are agents $i_1, \ldots, i_{m-1} \in \Delta$ such that for all $k \in \{1, \ldots, m-1\}$ we have $s_k \cap O_{i_k} = s_{k+1} \cap O_{i_k}$. Note that $s = s_1$ and $s_1 \vDash \Lambda^m(\top)$, i.e. $s_1 \vDash \|\psi\|_\mathcal{F} \wedge \bigwedge_{i\in\Delta} \forall(V \setminus O_i)(\theta \to \Lambda^{m-1}(\top))$. This implies $s_1 \vDash \forall(V \setminus O_{i_1})(\theta \to \Lambda^{m-1}(\top))$. By $s_1 \cap O_{i_1} = s_2 \cap O_{i_1}$ we get $s_2 \vDash \theta \to \Lambda^{m-1}(\top)$. Because s_2 is a state of $\mathcal{F}$ we have $s_2 \vDash \theta$ and therefore $s_2 \vDash \Lambda^{m-1}(\top)$. Iterating this, we get $s_{1+k} \vDash \Lambda^{m-k}(\top)$ for all $k \in \{1, \ldots, m-1\}$. In particular $s_m \vDash \Lambda(\top)$ which implies $s_m \vDash \|\psi\|_\mathcal{F}$. By $s_m = t$ and the induction hypothesis, this shows $(\mathcal{F}, t) \vDash \psi$.

For public announcements $\varphi = [!\psi]\xi$ note the following equivalences:

$$\begin{array}{lll} & (\mathcal{F}, s) \vDash [!\psi]\xi & \\ \iff & (\mathcal{F}, s) \vDash \psi \text{ implies } (\mathcal{F}^\psi, s) \vDash \xi & \text{by Definition 2.2.3} \\ \iff & s \vDash \|\psi\|_\mathcal{F} \text{ implies } s \vDash \|\xi\|_{\mathcal{F}^\psi} & \text{by induction hypothesis} \\ \iff & s \vDash \|\psi\|_\mathcal{F} \to \|\xi\|_{\mathcal{F}^\psi} & \end{array}$$

□

We can now explain the semantics for public announcements given in Definition 2.2.3. Note that public announcements only modify the state law of the knowledge structure. Moreover, the new state law is always a conjunction containing the previous one. Hence the set of states is restricted, just like public announcements on Kripke models can only restrict and never enlarge the set of possible worlds.

An announcement uses the local boolean equivalent of the announced formula with respect to the original structure $\mathcal{F}$, just like in Kripke semantics the condition for copying worlds or cutting edges is about the original model $\mathcal{M}$ and not the model $\mathcal{M}^\psi$ after the announcement. Hence a well-known consequence of this definition also holds for our knowledge structures: Truthful announcements can be unsuccessful in the sense that after something is announced, it might not be true anymore. Famous examples are Moore sentences of the form "It is snowing in Amsterdam and you don't know it".

Theorem 2.2.8 is somewhat surprising because it "explains away" knowledge and announcement operators. For dynamic operators like $[!\varphi]$ this is also possible on Kripke models, using well-known reduction axioms that might lead to an exponentially larger formula [Lut06]. In contrast, removing static modalities like K_i is impossible on Kripke models. It can be done on our structures only because the implicit valuation function is injective.

All this does not make DEL any less expressive. Rather we can think of the original formulas as universally usable — they capture an intended meaning across different models or structures. Their local boolean equivalents given by Definition 2.2.6 still do so across states, but only within a specific structure.

Common knowledge is the trickiest part in the definitions and proof above. One might think that, given the representation of epistemic relations as sets of observable propositions, there would be an easier way. How about using the set $O_\Delta := \cap_{i\in\Delta} O_i$ and the induced relation $R_\Delta xy :\iff (x \cap O_\Delta = y \cap O_\Delta)$ instead of the harder to compute transitive closure of $\mathcal{E}_\Delta$ above? Intuitively, this would define common knowledge as those observations which all agents in a group make. However, the following example shows that in general these two definitions do not yield the same relation. Hence O_Δ does not give us a shortcut in computing common knowledge of a group on knowledge structures.

2.2.9. Example. Consider the knowledge structure

$$(V = \{p, q\},\ \theta = (p \leftrightarrow q),\ O_a = \{p\},\ O_b = \{q\})$$

which has the two states $\{p, q\}$ and $\varnothing$. Note that on this set of states (i) $\mathcal{E}_{\{a,b\}}$ is *not* the total relation but the identity and (ii) $\cap_{i\in\Delta} O_i = \varnothing$ and therefore R_Δ would be total.

2.3 Example: Muddy Children

What do knowledge structures look like in practice? To give an answer, we consider probably the most famous example in the epistemic agency literature. The *Muddy Children* story illustrates how announcements, both of propositional and of epistemic facts, work on knowledge structures.

An early version of this puzzle are the three ladies on a train:

> "Three ladies, A, B, C in a railway carriage all have dirty faces and are all laughing. It suddenly flashes on A: why doesn't B realize C is laughing at her? — Heavens! *I* must be laughable. (Formally: if I, A, am not laughable, B will be arguing: if I, B, am not laughable, C has nothing to laugh at. Since B does not so argue, I, A, must be laughable.)" [Lit53]

Isomorphic to this is the story about muddy children:

> "Imagine n children playing together. The mother of these children has told them that if they get dirty there will be severe consequences. So, of course, each child wants to keep clean, but each would love to see the others get dirty. Now it happens during their play that some

> of the children, say k of them, get mud on their foreheads. Each can see the mud on others but not on his own forehead. So, of course, no one says a thing. Along comes the father, who says, "At least one of you has mud on your forehead," thus expressing a fact known to each of them before he spoke (if $k > 1$). The father then asks the following question, over and over: "Does any of you know whether you have mud on your own forehead?" Assuming that all the children are perceptive, intelligent, truthful, and that they answer simultaneously, what will happen?" [Fag+95, p. 4]

For a standard analysis with Kripke models, see [Fag+95, p. 24-30] or [DHK07, p. 93-96].

Let p_i stand for "child i is muddy". We consider the case of three children $I = \{1, 2, 3\}$ who are all muddy, i.e. the actual state is $\{p_1, p_2, p_3\}$. At the beginning the children do not have any further information, hence the initial knowledge structure $\mathcal{F}_0$ in Figure 2.1 has the state law $\theta_0 = \top$ and the set of states is the full powerset of the vocabulary, i.e. $\mathcal{P}(\{p_1, p_2, p_3\})$. All children can observe whether the others are muddy but do not see their own face. This is represented with observational variables: Agent 1 observes p_2 and p_3, etc.

$$\mathcal{F}_0 = \left(V = \{p_1, p_2, p_3\}, \theta_0 = \top, \begin{array}{r} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

$$\Downarrow [!(p_1 \lor p_2 \lor p_3)]$$

$$\mathcal{F}_1 = \left(V = \{p_1, p_2, p_3\}, \theta_1 = (p_1 \lor p_2 \lor p_3), \begin{array}{r} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array} \right)$$

Figure 2.1: Knowledge structures before and after the first announcement.

Now the father says "At least one of you is muddy." which we model as a public announcement of $p_1 \lor p_2 \lor p_3$. This limits the set of states by adding this statement to the state law. Note that it already is a purely boolean statement, hence the formula is added as it is, leading to the new knowledge structure $\mathcal{F}_1$ in Figure 2.1.

The father now asks "Do you know if you are muddy?" but none of the children does. As it is common in the literature, we understand this as a public announcement of "Nobody knows their own state.":

$$\bigwedge_{i \in I} (\neg(K_i p_i \lor K_i \neg p_i))$$

This is not a purely boolean formula, hence the public announcement is slightly more complicated: Using Definition 2.2.6 and Theorem 2.2.8 we first find a boolean formula which on the current knowledge structure $\mathcal{F}_1$ is equivalent to the announced formula. Then this boolean equivalent is added to θ.

We have

$$\begin{aligned}
\|K_1 p_1\|_{\mathcal{F}_1} &= \forall (V \setminus O_1)(\theta_1 \to \|p_1\|_{\mathcal{F}_1}) \\
&\equiv \forall p_1((p_1 \lor p_2 \lor p_3) \to p_1) \\
&\equiv ((\top \lor p_2 \lor p_3) \to \top) \land ((\bot \lor p_2 \lor p_3) \to \bot) \\
&\equiv \neg (p_2 \lor p_3)
\end{aligned}$$

$$\begin{aligned}
\|K_1 \neg p_1\|_{\mathcal{F}_1} &= \forall (V \setminus O_1)(\theta_1 \to \|\neg p_1\|_{\mathcal{F}_1}) \\
&\equiv \forall p_1((p_1 \lor p_2 \lor p_3) \to \neg p_1) \\
&\equiv ((\top \lor p_2 \lor p_3) \to \neg\top) \land ((\bot \lor p_2 \lor p_3) \to \neg\bot) \\
&\equiv \bot
\end{aligned}$$

and analogous for $K_2 p_2$, $K_2 \neg p_2$, $K_3 p_3$ and $K_3 \neg p_3$. These results make intuitive sense: In $\mathcal{F}_1$ it is common knowledge that at least one child is muddy. Hence a child knows it is muddy if and only if it sees that the other two children are clean, but it can never know that it is clean itself.

The announced formula becomes

$$\begin{aligned}
\| \bigwedge_{i \in I} (\neg (K_i p_i \lor K_i \neg p_i)) \|_{\mathcal{F}_1} &= \bigwedge_{i \in I} \| \neg (K_i p_i \lor K_i \neg p_i) \|_{\mathcal{F}_1} \\
&\equiv \neg(\neg(p_2 \lor p_3)) \land \neg(\neg(p_1 \lor p_3)) \land \neg(\neg(p_1 \lor p_2)) \\
&\equiv (p_2 \lor p_3) \land (p_1 \lor p_3) \land (p_1 \lor p_2)
\end{aligned}$$

The announcement essentially says that at least two children are muddy. We get a knowledge structure $\mathcal{F}_2$ with the following more restrictive state law θ_2. The vocabulary and the observational variables do not change, so we do not repeat them.

$$\theta_2 = (p_1 \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (p_1 \lor p_3) \land (p_1 \lor p_2))$$

Now the same announcement is made again: "Nobody knows their own state." It is important that we again start with the epistemic formula $\bigwedge_{i \in I} (\neg (K_i p_i \lor K_i \neg p_i))$ and compute a new boolean equivalent, now with respect to $\mathcal{F}_2$.

By further boolean reasoning we have that

$$\begin{aligned}
\|K_1 p_1\|_{\mathcal{F}_2} &= \forall (V \setminus O_1)(\theta_2 \to \|p_1\|_{\mathcal{F}_2}) \\
&\equiv \forall p_1((p_1 \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (p_1 \lor p_3) \land (p_1 \lor p_2)) \to p_1) \\
&\equiv ((\top \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (\top \lor p_3) \land (\top \lor p_2)) \to \top) \\
&\quad \land ((\bot \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (\bot \lor p_3) \land (\bot \lor p_2)) \to \bot) \\
&\equiv \top \land ((p_2 \lor p_3) \land ((p_2 \lor p_3) \land p_3 \land p_2) \to \bot) \\
&\equiv \neg((p_2 \lor p_3) \land ((p_2 \lor p_3) \land p_3 \land p_2)) \\
&\equiv \neg(p_2 \land p_3)
\end{aligned}$$

and

$$\begin{array}{rcl}
\|K_1\neg p_1\|_{\mathcal{F}_2} & = & \forall(V \setminus O_1)(\theta_2 \to \|\neg p_1\|_{\mathcal{F}_2}) \\
& \equiv & \forall p_1(\theta_2 \to \neg p_1) \\
& \equiv & \forall p_1((p_1 \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (p_1 \lor p_3) \land (p_1 \lor p_2)) \to \neg p_1) \\
& \equiv & ((\top \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (\top \lor p_3) \land (\top \lor p_2)) \to \neg\top) \\
& & \land((\bot \lor p_2 \lor p_3) \land ((p_2 \lor p_3) \land (\bot \lor p_3) \land (\bot \lor p_2)) \to \neg\bot) \\
& \equiv & (\top \land ((p_2 \lor p_3) \land \top \land \top) \to \bot) \\
& & \land((p_2 \lor p_3) \land ((p_2 \lor p_3) \land (p_3) \land (p_2)) \to \top) \\
& \equiv & (p_2 \lor p_3 \to \bot) \land \top \\
& \equiv & \neg(p_2 \lor p_3)
\end{array}$$

which together gives us:

$$\begin{array}{rcl}
\|\neg(K_1 p_1 \lor K_1\neg p_1)\|_{\mathcal{F}_2} & = & \neg(\|K_1 p_1\|_{\mathcal{F}_2} \lor \|K_1\neg p_1\|_{\mathcal{F}_2}) \\
& \equiv & \neg(\neg(p_2 \land p_3) \lor \neg(p_2 \lor p_3)) \\
& \equiv & (p_2 \land p_3) \land (p_2 \lor p_3) \\
& \equiv & p_2 \land p_3
\end{array}$$

We have analogous formulas for children 2 and 3. Note that this admittedly tedious calculation brings to light a detail of the puzzle: It would suffice to announce "I do not know *that* I am muddy", in contrast to "I do not know *whether* I am muddy" which in general is more informative but not in this specific situation.

Finally, with respect to $\mathcal{F}_2$, we get the following boolean equivalent of the announcement, essentially saying that everyone is muddy.

$$\begin{array}{rcl}
\|\bigwedge_{i\in I}(\neg(K_i p_i \lor K_i\neg p_i))\|_{\mathcal{F}_2} & \equiv & (p_3 \land p_2) \land (p_3 \land p_1) \land (p_2 \land p_1) \\
& \equiv & p_1 \land p_2 \land p_3
\end{array}$$

The resulting knowledge structure thus has the state law $\theta_3 = \theta_2 \land (p_1 \land p_2 \land p_3)$ which is equivalent to $p_1 \land p_2 \land p_3$ and marks the end of the story: The only state left is the situation in which all three children are muddy. Moreover, this is common knowledge among them because the only state is also the only state reachable via $\mathcal{E}_I^*$ in Definition 2.2.3. Alternatively, note that the fixed point mentioned in Definition 2.2.6 in this case will be the same as θ_3.

2.4 Equivalence Proof for S5-PAL

We now look more deeply into the foundations of what we have been doing. For a start, we show that knowledge structures and standard models for DEL are equivalent from a semantic point of view. Lemma 2.4.1 gives us a canonical way to show that a knowledge structure and an S5 Kripke model satisfy the same formulas. Theorems 2.4.3 and 2.4.6 say that such equivalent models and structures can always be found.

2.4.1. Lemma. *Suppose we have a knowledge structure $\mathcal{F} = (V, \theta, O_1, \ldots, O_n)$ and a finite S5 Kripke model $M = (W, \pi, R)$ with a set of primitive propositions $U \subseteq V$. Furthermore, suppose we have a function $g \colon W \to \mathcal{P}(V)$ such that*

C1 For all $w_1, w_2 \in W$, and all i such that $1 \leq i \leq n$, we have that $g(w_1) \cap O_i = g(w_2) \cap O_i$ iff $R_i w_1 w_2$.

C2 For all $w \in W$ and $p \in U$, we have that $p \in g(w)$ iff $p \in \pi(w)$.

C3 For every $s \subseteq V$, s is a state of $\mathcal{F}$ iff $s = g(w)$ for some $w \in W$.

Then, for every $\mathcal{L}(U)$-formula φ we have $(\mathcal{F}, g(w)) \vDash \varphi$ iff $(\mathcal{M}, w) \vDash \varphi$.

Before we dive into the proof, let us step back a bit to see that conditions C1 to C3 describe a special case of something well-known, namely a surjective p-morphism between the model $\mathcal{M}$ and a model encoded by the structure $\mathcal{F}$ which uses a subset of $\mathcal{P}(V)$ as its set of worlds. We will make this precise in Definition 2.4.2 below. The mathematical reader might thus already be convinced by general invariance results [BRV01, §2.1] and skip the following induction.

Proof:
We proceed by induction on φ. First consider the base case when φ is a primitive proposition, say p. Then, by condition C2, we have that $(\mathcal{F}, g(w)) \vDash p$ iff $p \in g(w)$ iff $p \in \pi(w)$ iff $(\mathcal{M}, w) \vDash p$.

Now suppose that φ is not a primitive proposition, and as an induction hypothesis the claim holds for every formula of lower complexity than φ. We distinguish four cases:

1. φ is of the form $\neg\psi$ or $\psi \wedge \xi$. Definitions 1.1.3 and 2.2.3 do the same recursion for negations and conjunctions, hence this case follows from the induction hypothesis.

2. φ is of the form $K_i\psi$. By Definition 2.2.3, we have $(\mathcal{F}, g(w)) \vDash K_i\psi$ iff $(\mathcal{F}, s) \vDash \psi$ for all states s of $\mathcal{F}$ with $g(w) \cap O_i = s \cap O_i$. By C3 this is equivalent to having $(\mathcal{F}, g(w')) \vDash \psi$ for all $w' \in W$ with $g(w) \cap O_i = g(w') \cap O_i$, which by C1 is equivalent to $(\mathcal{F}, g(w')) \vDash \psi$ for all $w' \in W$ with $R_i w w'$. Now by the induction hypothesis, this is equivalent to $(\mathcal{M}, w') \vDash \psi$ for all $w' \in W$ with $R_i w w'$, which is exactly $(\mathcal{M}, w) \vDash K_i\psi$ by Definition 1.1.3.

3. φ is of the form $C_\Delta\psi$. Recall that for states s and t of $\mathcal{F}$, $(s, t) \in \mathcal{E}_\Delta$ iff there exists an $i \in \Delta$ with $s \cap O_i = t \cap O_i$. By C1 we have, for all $w_1, w_2 \in W$:

$$(g(w_1), g(w_2)) \in \mathcal{E}_\Delta \text{ iff } (w_1, w_2) \in \bigcup_{i \in \Delta} R_i$$

As $\mathcal{E}^*_\Delta$ is the transitive closure of $\mathcal{E}_\Delta$ and R_Δ is that of $\bigcup_{i \in \Delta} R_i$, by C3 we have for all $w_1, w_2 \in W$ that

$$(g(w_1), g(w_2)) \in \mathcal{E}^*_\Delta \text{ iff } (w_1, w_2) \in R^M_\Delta$$

We now claim that $(\mathcal{F}, g(w)) \vDash C_\Delta \psi$ iff $(\mathcal{M}, w) \vDash C_\Delta \psi$. On the one hand, we have $(\mathcal{F}, g(w)) \vDash C_\Delta \psi$ iff for all states s of $\mathcal{F}$ with $(g(w), s) \in \mathcal{E}_\Delta^*$ we have $(\mathcal{F}, s) \vDash \psi$. By C3 this is the case iff for all $w' \in W$ with $(g(w), g(w')) \in \mathcal{E}_\Delta^*$ we have $(\mathcal{F}, g(w')) \vDash \psi$. On the other hand, $(\mathcal{M}, w) \vDash C_\Delta \psi$ iff for all $w' \in W$ with $(w, w') \in R_\Delta$ we have $\mathcal{M}, w' \vDash \psi$. Hence our claim follows by the above connection between the two transitive closures and the induction hypothesis applied to ψ.

4. φ is of the form $[!\psi]\xi$. By Definition 1.2.2, we have that $(\mathcal{M}, w) \vDash [!\psi]\xi$ iff $(\mathcal{M}, w) \vDash \psi$ implies $(\mathcal{M}^\psi, w) \vDash \xi$, and by Definition 2.2.3 we have that $(\mathcal{F}, g(w)) \vDash [!\psi]\xi$, iff $(\mathcal{F}, g(w)) \vDash \psi$ implies $(\mathcal{F}^\psi, g(w)) \vDash \xi$. As $(\mathcal{M}, w) \vDash \psi$ iff $(\mathcal{F}, g(w)) \vDash \psi$ by the induction hypothesis, it suffices to prove that $(\mathcal{M}^\psi, w) \vDash \xi$ iff $(\mathcal{F}^\psi, g(w)) \vDash \xi$. Let g' be the restriction of g to $W^{\mathcal{M}^\psi} = \{w \in W \mid (\mathcal{M}, w) \vDash \psi\}$. Note that because g fulfills the universal conditions C1 and C2, they must also hold for g' with respect to the restricted set $W^{\mathcal{M}^\psi}$. To show C3 for g', for left to right suppose $s \subseteq V$ is a state of $\mathcal{F}^\psi$. Then s is also a state of $\mathcal{F}$ and by condition C3 for g, there is a $w \in W$ such that $s = g(w)$. Moreover, $\mathcal{F}, s \vDash \psi$ and therefore by the induction hypothesis $(\mathcal{M}, w) \vDash \psi$. Hence $w \in W^{\mathcal{M}^\psi}$ and we also have $g'(w) = s$. For right to left suppose $g'(w) = s$ for some $w \in W^{\mathcal{M}^\psi}$ and some $s \subseteq V$. Then $(\mathcal{M}, w) \vDash \psi$ and s is a state of $\mathcal{F}$ because $g(w) = g'(w) = s$. Therefore by the induction hypothesis $\mathcal{F}, s \vDash \psi$. Hence $s \vDash \|\psi\|_\mathcal{F}$ which implies that s is also a state of $\mathcal{F}^\psi$. Together, g' fulfills all three conditions and by the induction hypothesis we get that $(\mathcal{M}^\psi, w) \vDash \xi$ iff $(\mathcal{F}^\psi, g(w)) \vDash \xi$. □

The following definition and theorem show that for every knowledge structure there is an equivalent Kripke model. This is the easy direction which follows directly from Lemma 2.4.1.

2.4.2. Definition. For any knowledge structure $\mathcal{F} = (V, \theta, O)$, we define the Kripke model $\mathcal{M}(\mathcal{F}) := (W, \pi, R)$ as follows:

1. $W := \{s \subseteq V \mid s \vDash \theta\}$ is the set of all states of $\mathcal{F}$
2. for each $w \in W$, let the assignment be w itself: $\pi(w) := w$
3. for each agent i and all $v, w \in W$, let $R_i vw$ iff $v \cap O_i = w \cap O_i$

2.4.3. Theorem. *The function $\mathcal{M}(\cdot)$ from Definition 2.4.2 preserves truth: For any knowledge structure $\mathcal{F}$, any state s of $\mathcal{F}$ and any formula $\varphi \in \mathcal{L}(V)$, we have $(\mathcal{F}, s) \vDash \varphi$ iff $(\mathcal{M}(\mathcal{F}), s) \vDash \varphi$.*

Proof:
By Lemma 2.4.1 using the identity function for g. □

2.4.4. Example. We can apply Definition 2.4.2 to $\mathcal{F} = (\{p, q\}, p \to q, \{p\}, \{q\})$ from Example 2.2.2. The result is the equivalent Kripke model shown in Figure 2.2: The set of worlds is $W = \{s \subseteq \{p, q\} \mid s \vDash p \to q\} = \{\varnothing, \{q\}, \{p, q\}\}$ and the valuation function is the identity. To illustrate point 3 of Definition 2.4.2, for example, we have an edge for agent 1 between $\varnothing$ and $\{q\}$ because $\varnothing \cap O_1 = \varnothing \cap \{p\} = \varnothing = \{q\} \cap \{p\} = \{q\} \cap O_1$.

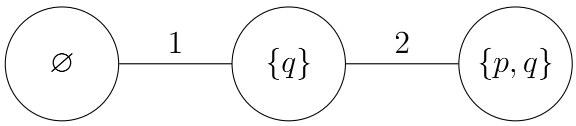


Figure 2.2: Kripke model $\mathcal{M}(\mathcal{F})$ equivalent to $\mathcal{F}$ from Example 2.2.2.

Vice versa, for any $\mathsf{S5}$ Kripke model we can find an equivalent knowledge structure. This is the both more interesting and more difficult direction. The essential idea is to add propositions as observational variables to encode the relations of each agent. To obtain a simple knowledge structure we should add as few propositions as possible. The method below adds $\sum_{i \in I} \lceil \log_2 k_i \rceil$ propositions, where k_i is the number of R_i-equivalence classes. This could be further improved if one were to find a general way of using the propositions already present in the Kripke model as observational variables directly.

2.4.5. Definition. For any S5 model $\mathcal{M} = (W, \pi, R)$ with vocabulary U we define a knowledge structure $\mathcal{F}(\mathcal{M})$ as follows. For each agent i, write $\gamma_{i,1}, \dots, \gamma_{i,k_i}$ for the equivalence classes given by R_i and let $l_i := \lceil \log_2 k_i \rceil$. Let O_i be a set of l_i many fresh atomic propositions. This yields sets of observational variables $O_1, \dots, O_n$, all disjoint to each other. If agent i has a total relation, i.e. only one equivalence class, then we have $O_i = \varnothing$. Enumerate k_i many subsets of O_i as $O_{\gamma_{i,1}}, \dots, O_{\gamma_{i,k_i}}$ and define $g_i \colon W \to \mathcal{P}(O_i)$ by $g_i(w) := O_{\gamma_i(w)}$, where $\gamma_i(w)$ is the R_i-equivalence class of w. Let $V := U \cup \bigcup_{0<i\leq n} O_i$ and define $g \colon W \to \mathcal{P}(V)$ by

$$g(w) := \{v \in U \mid v \in \pi(w)\} \cup \bigcup_{0<i\leq n} g_i(w)$$

Finally, let $\mathcal{F}(\mathcal{M}) := (V, \theta_M, O_1, \dots, O_n)$ using

$$\theta_M := \bigvee \{g(w) \sqsubseteq V \mid w \in W\}$$

where $\sqsubseteq$ is the abbreviation from Definition 1.0.1 saying that out of the propositions in the second set exactly those in the first set are true.

Note that the idea here is to represent the state law θ_M as a BDD and not as a complex formula. Thereby we obtain a compact representation for many Kripke models, especially for situations with a lot of symmetry like the muddy children story. However, in the worst case a BDD can have exponential size in the number of variables [Bry86]. Hence $\mathsf{Bdd}(\theta_M)$ might be of size exponential in $|V|$.

We also implemented these translations between Kripke models and knowledge structures and will discuss them again in Chapter 3. For now it remains to prove that the translation is correct.

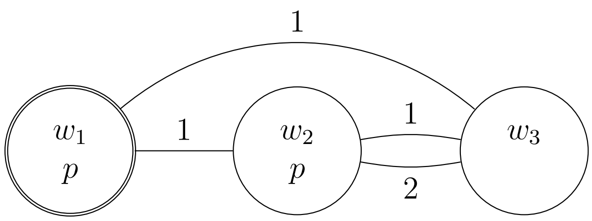
2.4.6. THEOREM. *The function $\mathcal{F}(\cdot)$ from Definition 2.4.5 preserves truth: For any finite pointed S5 Kripke model $(\mathcal{M}, w)$ and every formula φ, we have $(\mathcal{M}, w) \vDash \varphi$ iff $(\mathcal{F}(\mathcal{M}), g(w)) \vDash \varphi$.*

Proof:
We have to check that Lemma 2.4.1 applies to Definition 2.4.5. To show C1, take any $w_1, w_2 \in W$ and $i \in \{1, \dots, n\}$. Note that $g(w_1) \cap O_i = g_i(w_1) \cap O_i$ and $g(w_2) \cap O_i = g_i(w_2) \cap O_i$ because the observational variables introduced in Definition 2.4.5 are disjoint sets of fresh propositions. By definition of g_i, we have that $g_i(w_1)$ and $g_i(w_2)$ are the same subset of O_i iff w_1 and w_2 are in the same R_i-equivalence class. This shows that $g(w_1) \cap O_i = g(w_2) \cap O_i$ iff $R_i w_1 w_2$.

For C2, take any $w \in W$ and any $v \in U$. Note that U is the original set of atomic propositions and therefore does not contain observational variables. Hence by definition of g we have $v \in g(w)$ iff $v \in \pi(w)$.

For the right-to-left part of C3: If $s = g(w)$ for some $w \in W$, then by definition of θ_M we have $g(w) \vDash \theta_M$, hence $g(w)$ is a state of $\mathcal{F}(\mathcal{M})$. For the left-to-right part, suppose s is a state of $\mathcal{F}(\mathcal{M})$. Then $s \vDash \theta_M$, hence it must satisfy one of the disjuncts and there must be a $w \in W$ such that $s \vDash g(w) \sqsubseteq V$. Then by definition of $\sqsubseteq$ we have $s = g(w)$. Now the theorem follows from Lemma 2.4.1. $\square$

2.4.7. EXAMPLE. Consider the pointed Kripke model $(\mathcal{M}, w_1)$ in Figure 2.3. Agent 2 knows that p, agent 1 does not know that p. Moreover, agent 1 does not even know whether agent 2 knows whether p.



$\mathcal{M}, w_1 \vDash K_2 p \wedge \neg K_1 p$
$\mathcal{M}, w_1 \vDash \neg K_1(K_2 p \vee K_2 \neg p)$
$\mathcal{M}, w_1 \vDash \neg K_1 \neg(K_2 p \vee K_2 \neg p)$

Figure 2.3: Pointed Kripke model $(\mathcal{M}, w_1)$ and some facts about it.

Now let us see how this knowledge and meta-knowledge get encoded symbolically. This direction is more difficult than going from a knowledge structure to a Kripke model as in Example 2.4.4, because here the worlds are not uniquely identified by valuations: $\pi(w_1) = \pi(w_2)$. Applying Definition 2.4.5 therefore means that we add one fresh proposition $O_2 := \{q\}$ to distinguish the two epistemic equivalence classes $\{w_1\}$ and $\{w_2, w_3\}$ of agent 2. For example, let $g_2(w_1) := \{q\}$ and $g_2(w_2) = g_2(w_3) := \varnothing$. Then we have $g(w_1) = \{p, q\}$, $g(w_2) = \{p\}$ and $g(w_3) = \varnothing$. Now we can compute the state law, a boolean formula over the vocabulary $V = \{p, q\}$, as follows:

$$\begin{aligned}\theta_M &= (g(w_1) \sqsubseteq V) \lor (g(w_2) \sqsubseteq V) \lor (g(w_3) \sqsubseteq V) \\ &= (\{p,q\} \sqsubseteq \{p,q\}) \lor (\{p\} \sqsubseteq \{p,q\}) \lor (\varnothing \sqsubseteq \{p,q\}) \\ &= (p \land q) \lor (p \land \neg q) \lor (\neg p \land \neg q) \\ &= q \rightarrow p\end{aligned}$$

The equivalent knowledge structure is thus

$$\mathcal{F}(\mathcal{M}) = (V' = \{p, q\}, \theta_M = q \rightarrow p, O_1 = \varnothing, O_2 = \{q\})$$

and the scene $(\mathcal{F}(\mathcal{M}), \{p, q\})$ is equivalent to $(\mathcal{M}, w_1)$.

2.5 Knowledge Transformers

We have seen how the two ways of interpreting DEL, though computationally different, are semantically equivalent. This leads us to consider how their interplay will work in more complex settings. The obvious direction to probe this is the area where DEL unleashes its full power: action models and the product update. We now show how our structures can be accompanied with *transformers* to model more complex events.

The following knowledge transformers are to knowledge structures what S5 action models without factual change are to S5 Kripke models.

2.5.1. Definition. A *knowledge transformer* for a given vocabulary V and set of agents $I = \{1, \dots, n\}$ is a tuple $\mathcal{X} = (V^+, \theta^+, O_1, \dots, O_n)$, where V^+ is a set of atomic propositions such that $V \cap V^+ = \varnothing$, θ^+ is a possibly epistemic formula from $\mathcal{L}(V \cup V^+)$ called the *event law* and $O_i \subseteq V^+$ for all agents i. An *event* is a knowledge transformer together with a subset $x \subseteq V^+$, written as $(\mathcal{X}, x)$.

The *knowledge transformation* of a knowledge structure $\mathcal{F} = (V, \theta, O_1, \dots, O_n)$ with a knowledge transformer $\mathcal{X} = (V^+, \theta^+, O_1^+, \dots, O_n^+)$ for V is defined by:

$$\mathcal{F} \times \mathcal{X} := (V \cup V^+, \theta \land \|\theta^+\|_{\mathcal{F}}, O_1 \cup O_1^+, \dots, O_n \cup O_n^+)$$

Given a scene $(\mathcal{F}, s)$ and an event $(\mathcal{X}, x)$, we define $(\mathcal{F}, s) \times (\mathcal{X}, x) := (\mathcal{F} \times \mathcal{X}, s \cup x)$.

To illustrate the definition of knowledge transformers, we show how the public and semi-private announcements from above fit into this new symbolic framework.

2.5.2. Example. The public announcement of φ is the event

$$(\mathcal{X} = (V^+ = \varnothing,\ \theta^+ = \varphi,\ O_1 = \varnothing,\ \ldots,\ O_n = \varnothing), x = \varnothing)$$

and the semi-private announcement of φ to a group Δ is given by

$$((\{p_\varphi\}, p_\varphi \leftrightarrow \varphi, O_1^+, \ldots, O_n^+), \{p_\varphi\})$$

where $O_i^+ = \{p_\varphi\}$ if $i \in \Delta$ and $O_i^+ = \varnothing$ otherwise.

The event law θ^+ is not restricted to be a boolean formula, just like preconditions of action models can be arbitrary formulas. Still, applying a knowledge transformer to a knowledge structure should again yield a knowledge structure with a boolean formula as the new state law. Hence, in Definition 2.5.1 we do not directly take the conjunction of θ and θ^+, but first localize θ^+ to the boolean equivalent $\|\theta^+\|_{\mathcal{F}}$. This formula will be equivalent to θ^+ on the previous structure $\mathcal{F}$, but not necessarily on the new structure $\mathcal{F} \times \mathcal{X}$.

For example, if the announced formula contains a K_i operator, then we rewrite it by quantifying over $V \setminus O_i$, not over $V \cup V^+ \setminus O_i \cup O_i^+$ as one might first think. The latter would yield boolean equivalents with respect to $\mathcal{F} \times \mathcal{X}$ whereas the former is with respect to $\mathcal{F}$. Compare this to the product update in Definition 1.3.1 where the preconditions are also evaluated on the model *before* the update.

The alert reader might still be worried about the language of θ^+. In an action model we have one formula $\mathsf{pre}(a) \in \mathcal{L}_V$ for each possible action a. In a knowledge transformer, the single formula $\theta^+ \in \mathcal{L}(V \cup V^+)$ encodes preconditions for all events at once. Within that formula we use the propositional atoms from V^+ to distinguish different events.

In Example 2.5.2 above, the fresh variable p_φ is used to distinguish two events, where the announcement is positive or negative. The event law $p_\psi \leftrightarrow \psi$ means that $[p_\varphi \mapsto \top]\theta^+ \equiv \psi$ is the precondition for one event and $[p_\varphi \mapsto \bot]\theta^+ \equiv \neg\psi$ for the other. In particular, ψ and thereby θ^+ may contain modal operators, for example if we announce $K_a p$. Intuitively though, in the scope of these modal operators there should only be atoms from V and not from V^+ because atoms from V^+ describe *which* event happens and not *when*. That is, θ^+ should be a boolean combination of formulas from $\mathcal{L}(V)$ and atoms from V^+. Somewhat abusing notation, this is the language $\mathcal{L}_B(\mathcal{L}(V) \cup V')$ and in practice we will only use event laws of that form. Still, we can keep the simpler definition with $\theta^+ \in \mathcal{L}(V \cup V^+)$, because the additional expressivity does not actually matter. The following example illustrates that K operators in front of an atom $q \in V^+$ simply disappear once we apply the transformer.

2.5.3. Example. Consider the structure $(V = \{p\}, \theta = \top, O_a = \{p\}, O_b = \varnothing)$ and the following, somewhat unusual but well-defined, knowledge transformer:

$$(V^+ = \{q\},\ \theta^+ = (K_a q \vee K_b p),\ O_a^+ = \varnothing,\ O_b^+ = \{q\})$$

To apply the former to the latter, we calculate the new state law

$$\theta \land \|\theta^+\|_{\mathcal{F}} \;\equiv\; \top \land \|K_a q \lor K_b p\|_{\mathcal{F}} \;\equiv\; (\forall \varnothing q) \lor (\forall p p) \;\equiv\; q \lor \bot \;\equiv\; q$$

and then get the result $\mathcal{F} \times \mathcal{X} = (V = \{p, q\},\ \theta = q,\ O_a = \{p\},\ O_b = \{q\})$. Note that the event law of $\mathcal{X}$ implies $q \lor K_b p$ and on $\mathcal{F}$ the condition $K_b p$ is always false. Hence only $\{q\}$ and not $\varnothing$ can happen. This is reflected in the new state law which makes q common knowledge among all agents. It does not matter whether q was prefixed with K_a or K_b or nothing at all.

An obvious question about knowledge transformers is how they relate to action models, i.e. whether they describe the same class of events. The answer is the same as for the relation between Kripke models and knowledge structures: For any $\mathsf{S5}$ action model there is an equivalent transformer and vice versa. We make this precise as follows, using the same ideas as for Definitions 2.4.2 and 2.4.5 and then using Lemma 2.4.1.

2.5.4. DEFINITION. For any knowledge transformer $\mathcal{X} = (V^+, \theta^+, O_1^+, \dots, O_n^+)$ we define an $\mathsf{S5}$ action model $\mathsf{Act}_{\mathsf{S5}}(\mathcal{X})$ as follows. First, let the set of actions be $A := \mathcal{P}(V^+)$. Second, for any two actions $\alpha, \beta \in A$, let $\alpha R_i \beta$ iff $\alpha \cap O_i^+ = \beta \cap O_i^+$. Third, for any α, let $\mathsf{pre}(\alpha) := [\alpha \sqsubseteq V^+]\,\theta^+$. Finally, let $\mathsf{Act}_{\mathsf{S5}}(\mathcal{X}) := (A, (R_i)_{i \in I}, \mathsf{pre})$.

2.5.5. DEFINITION. Suppose we have an $\mathsf{S5}$ action model $\mathcal{A} = (A, (R_i)_{i \in I}, \mathsf{pre})$. The function $\mathsf{Trf}_{\mathsf{S5}}$ maps it to a knowledge transformer as follows. Let P be a finite set of fresh propositions such that there is an injective labeling function $\ell \colon A \to \mathcal{P}(P)$ and let

$$\Phi := \bigwedge \{(\ell(a) \sqsubseteq P) \to \mathsf{pre}(a) \mid a \in A\}$$

where $\sqsubseteq$ is the "out of" abbreviation from Definition 1.0.1. Now, do the following for each i: Write A/R_i for the set of equivalence classes induced by R_i. Let O_i^+ be a finite set of fresh propositions such that there is an injective labeling function $g_i \colon A/R_i \to \mathcal{P}(O_i^+)$ and let

$$\Phi_i := \bigwedge \left\{ (g_i(\alpha) \sqsubseteq O_i) \to \left(\bigvee_{a \in \alpha} (\ell(a) \sqsubseteq P) \right) \,\middle|\, \alpha \in A/R_i \right\}$$

Finally, define $\mathsf{Trf}_{\mathsf{S5}}(\mathcal{A}) := (V^+, \theta^+, O_1^+, \dots, O_n^+)$ where $V^+ := P \cup \bigcup_{i \in I} O_i^+$ and $\theta^+ := \Phi \land \bigwedge_{i \in I} \Phi_i$.

In contrast to the translation in Definition 2.4.5, where θ_M could be represented as a BDD, here we cannot do so with θ^+ as it might contain non-boolean operators in Φ. Still, before taking the outer conjunction in θ^+ we can compute a smaller equivalent of the purely boolean part $\bigwedge_{i \in I} \Phi_i$.

2.5.6. Example. We translate the product update from our letter story in Example 1.3.3 to a knowledge transformation as follows. First note that in $\mathcal{M}$ both agents have a total relation, hence we do not have to add observational variables. The equivalent knowledge structure is just $\mathcal{F}(\mathcal{M}) = (\{p\}, \top, \varnothing, \varnothing)$. Now we use Definition 2.5.5 to obtain $\mathsf{Trf}_{\mathsf{S5}}(\mathcal{A})$. Choose the set $P = \{q\}$ where q is fresh, and label the events of $\mathcal{A}$ by $\ell(\alpha) := \{q\}$ and $\ell(\beta) := \varnothing$. We then get $\Phi := (q \to p) \wedge (\neg q \to \neg p) = q \leftrightarrow p$. Bob has a total relation in $\mathcal{A}$, so we can choose $O_b^+ = \varnothing$ and $g_b(\alpha) := g_b(\beta) := \varnothing$. Note that $\varnothing \sqsubseteq \varnothing = \top$. Hence $\Phi_b = (\top \to (q \vee \neg q)) = \top$. For Alice we need two labels, so let $O_a^+ := \{r\}$ where r is fresh, $g_a(\alpha) := \{r\}$ and $g_a(\beta) := \varnothing$. Then we get $\Phi_a = (r \to q) \wedge (\neg r \to \neg q) = r \leftrightarrow q$. Putting it all together we get $\theta^+ = (q \leftrightarrow p) \wedge (r \leftrightarrow q)$ and thereby this transformer:

$$\mathsf{Trf}_{\mathsf{S5}}(\mathcal{A}) = (V^+ = \{q, r\},\ \theta^+ = ((q \leftrightarrow p) \wedge (r \leftrightarrow q)),\ O_a^+ = \{r\},\ O_b^+ = \varnothing)$$

Finally, we can calculate the knowledge transformation $\mathcal{F}(\mathcal{M}) \times \mathsf{Trf}_{\mathsf{S5}}(\mathcal{A})$:

$$\begin{array}{rl} & (\{p\}, \top, \varnothing, \varnothing) \\ \times & (\{q, r\}, ((q \leftrightarrow p) \wedge (r \leftrightarrow q)), \{r\}, \varnothing) \\ = & (\{p, q, r\}, ((q \leftrightarrow p) \wedge (r \leftrightarrow q)), \{r\}, \varnothing) \end{array}$$

Observe that θ^+ makes q and r equivalent which makes this transformer redundant. As mentioned in Example 2.5.2, such a semi-private announcement can be done with a simpler transformer using only one proposition, in this case $(V^+ = \{q\}, \theta^+ = (q \leftrightarrow p), O_a^+ = \{q\},\ O_b^+ = \varnothing)$. In general however, the distinction between those propositions linked to preconditions and those describing the observation is needed to translate more complex action models to knowledge transformers.

It remains to show that our translations to go back and forth between $\mathsf{S5}$ action models and knowledge transformers are truthful in general. The following theorem says that we have an *action emulation*, as discussed in [ERS12], between the explicit and the symbolic representation of updates.

2.5.7. Theorem.

(i) The function Act *from Definition 2.5.4 preserves truth: For any scene $(\mathcal{F}, s)$, any event $(\mathcal{X}, x)$ and any formula φ over the vocabulary of $\mathcal{F}$ we have*

$$(\mathcal{F}, s) \times (\mathcal{X}, x) \vDash \varphi \iff (\mathcal{M}(\mathcal{F}) \times \mathsf{Act}_{\mathsf{S5}}(\mathcal{X})), (s, x) \vDash \varphi$$

(ii) The function Trf *from Definition 2.5.5 preserves truth: For any pointed* $\mathsf{S5}$ *Kripke model $(\mathcal{M}, w)$, any pointed* $\mathsf{S5}$ *action model $(\mathcal{A}, \alpha)$ and any formula φ over the vocabulary of $\mathcal{M}$ we have*

$$\mathcal{M} \times \mathcal{A}, (w, \alpha) \vDash \varphi \iff \mathcal{F}(\mathcal{M}) \times \mathsf{Trf}_{\mathsf{S5}}(\mathcal{A}), (g_{\mathcal{M}}(w) \cup g_{\mathcal{A}}(\alpha)) \vDash \varphi$$

where $g_{\mathcal{M}}$ is as in the construction of $\mathcal{F}(\mathcal{M})$ in Definition 2.4.2 and $g_{\mathcal{A}}$ is as in the construction of $\mathsf{Trf}_{\mathsf{S5}}(\mathcal{A})$ *in Definition 2.5.5.*

Proof:
We use Lemma 2.4.1. For the first part, g needs to map worlds of $\mathcal{M}(\mathcal{F}) \times \mathsf{Act}_{\mathsf{S5}}(\mathcal{X})$ to states of $\mathcal{F} \times \mathcal{X}$. The former are pairs $(s, x) \in \mathcal{P}(V) \times \mathcal{P}(V^+)$, hence we define $g(s, x) := s \cup x$. For the second part, g should map worlds of $\mathcal{M} \times \mathcal{A}$ to states of $\mathcal{F}(\mathcal{M}) \times \mathsf{Trf}_{\mathsf{S5}}(\mathcal{A})$. Hence let $g(w, \alpha) := g_{\mathcal{M}}(w) \cup g_{\mathcal{A}}(\alpha)$ where $g_{\mathcal{M}}$ and $g_{\mathcal{A}}$ are from Definitions 2.4.5 and 2.5.5 respectively. It is straightforward to check C1 to C3 for both functions. $\square$

Similar to the language $\mathcal{L}_D$, which contains dynamic operators for action models, we can also define a language with dynamic operators for knowledge transformers. Just like $\mathcal{L}_D$, the language we obtain will be more succinct, but not more expressive than $\mathcal{L}$. Analogously to Fact 1.3.8 we again have reduction axioms, which provide a globally truthful translation back to pure $\mathcal{L}$.

With respect to a given structure however, we can do even better: Formulas with dynamic operators for knowledge transformers have local boolean equivalents as well. We only need to add a clause like this to Definition 2.2.6:

$$\|[\mathcal{X}, x]\varphi\|_{\mathcal{F}} := \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \rightarrow [x \sqsubseteq V^+]\|\varphi\|_{\mathcal{F}\times\mathcal{X}}$$

Note that $[x \sqsubseteq V^+]$ plays two different roles here: Its first use reduces θ^+ to the actual precondition. The second use simulates that the actual event happens, i.e. that we evaluate at the state $s \cup x$ of the new structure $\mathcal{F} \times \mathcal{X}$ instead of any given state s of $\mathcal{F}$.

We postpone a complete definition of the language including dynamic operators for transformers and reduction axioms until Section 2.10, where we discuss the most general case of transformers. The next sections lead there step-by-step: from knowledge structures to belief structures, from knowledge transformers to belief transformers, and finally to transformers which also model factual change.

2.6 Belief Structures

The previous methods only allow us to represent Kripke models based on equivalence relations. But in Section 1.8 we already discussed a method to represent arbitrary relations over sets of states symbolically. Indeed this can also be used to generalize knowledge structures to what we call belief structures, employing Definition 1.8.8 as follows.

2.6.1. DEFINITION. A *belief structure* is a tuple $\mathcal{F} = (V, \theta, \Omega)$ where V is a finite set of propositional variables called the *vocabulary*, $\theta \in \mathcal{L}_B(V)$ is a boolean formula over V called the *state law* and Ω is a set of formulas indexed by agents such that for each agent i, $\Omega_i \in \mathcal{L}_B(V \cup V')$ is a boolean formula over the double vocabulary. We call this formula the *observation law* of i.

Any $s \subseteq V$ such that $s \vDash \theta$ is called a *state* of $\mathcal{F}$. A pair $(\mathcal{F}, s)$ where s is a state of $\mathcal{F}$ is called a *scene*. All terms and notational conventions for knowledge structures in Definition 2.2.1 also apply to belief structures.

The relations encoded in these structures do not have to be (but still may be) equivalence relations. We highlight this by calling them *belief* instead of *knowledge* structures. However, they are neither meant to only model belief, nor do we claim that they yield a semantics which is always appropriate to model beliefs. In fact, will show that they are equivalent to the standard general Kripke models — with all their features and problems. In particular, our belief structures are *not* meant to directly model or replace more complex definitions for *belief revision* as in the famous AGM framework from [AGM85], which has also been modeled as a part of DEL in [BS08].

2.6.2. DEFINITION. We define semantics for $\mathcal{L}_P$ on belief structures in the same way as in Definition 2.2.3 for knowledge structures, only changing the two clauses for K_i and C_i:

1. For belief we now use Ω_i instead of O_i:

$$(\mathcal{F}, s) \vDash \Box_i \varphi \text{ iff for all states } t \text{ of } \mathcal{F} : s \cup t' \vDash \Omega_i \text{ implies } (\mathcal{F}, t) \vDash \varphi$$

2. For common belief, let $\mathcal{E}_\Delta$ be the relation defined by $\mathcal{E}st :\iff \exists i \in \Delta : s \cup t' \vDash \Omega_i$ and denote its transitive closure by $\mathcal{E}^*_\Delta$. Then let

$$(\mathcal{F}, s) \vDash C_\Delta \varphi \text{ iff for all states } t \text{ of } \mathcal{F} : (s,t) \in \mathcal{E}^*_\Delta \text{ implies } (\mathcal{F}, t) \vDash \varphi$$

Note that in Definition 2.6.2 we did not change the semantics of public announcements, hence we implicitly use $\| \cdot \|_\mathcal{F}$ where $\mathcal{F}$ is now a belief structure. As on knowledge structures, all formulas have boolean equivalents with respect to a given belief structure.

2.6.3. DEFINITION. Given a belief structure $\mathcal{F}$, we redefine the translation from $\mathcal{L}_P$ to $\mathcal{L}_B$ from Definition 2.2.6 for the general modality $\Box_i$ and common belief C_Δ.

- For belief, let
$$\|\Box_i \psi\|_\mathcal{F} := \forall V'(\theta' \to (\Omega_i \to (\|\psi\|_\mathcal{F})'))$$

- For common belief, let $\|C_\Delta \psi\|_\mathcal{F} := \mathsf{gfp}\Lambda$, where Λ is the following operator on boolean formulas modulo equivalence and $\mathsf{gfp}\Lambda$ denotes a representative of its greatest fixed point:

$$\Lambda(\alpha) := \forall V' \left(\theta' \to \left(\bigvee_{i \in \Delta} \Omega_i \to (\|\psi\|_\mathcal{F} \wedge \alpha)' \right) \right)$$

As before with common knowledge, the translation of common belief is the most difficult part — see page 41 where we discuss the type of $\mathsf{gfp}\Lambda$ and how it can be computed.

It is crucial to use the primed formula $(\|\psi\|_{\mathcal{F}} \wedge \alpha)'$ in this translation for common belief. We check ψ at all reachable states, but it does not have to hold at the starting state: Common *belief* might be wrong. Hence we cannot use the same translation as for common *knowledge* in Definition 2.2.6 which assumes reflexivity, i.e. factivity. If we are interested in common *true* belief however, then an operator like the one in Definition 2.2.6 should be used because it is more efficient.

The quantification over V' removes all primed propositions. Hence Λ and the whole translation function are both of type $\| \cdot \| \colon \mathcal{L}_P(V) \to \mathcal{L}_B(V)$. It remains to state and prove that this translation is indeed correct.

2.6.4. Theorem. *Definition 2.6.3 preserves and reflects truth. That is, for any formula φ and any scene $(\mathcal{F}, s)$ where $\mathcal{F}$ is a belief structure, we have that $(\mathcal{F}, s) \vDash \varphi$ iff $s \vDash \|\varphi\|_{\mathcal{F}}$.*

Proof:
By induction on φ, as in the proof of Theorem 2.2.8. We only consider the two changed cases of belief and common belief. First, for $\Box_i$, note the following chain of equivalences. Recall from Definition 1.8.8 that we write $st' \vDash \varphi$ for $s \cup t' \vDash \varphi$.

$$\begin{array}{rcl}
\mathcal{F}, s \vDash \Box_i \varphi & \iff & \text{For all } t \in \mathcal{F} : \text{If } st' \vDash \Omega_i \text{ then } (\mathcal{F}, t) \vDash \varphi \\
& \iff & \text{For all } t \in \mathcal{F} : \text{If } st' \vDash \Omega_i \text{ then } t \vDash \|\varphi\|_{\mathcal{F}} \\
& \iff & \text{For all } t : \text{If } t \in \mathcal{F} \text{ and } st' \vDash \Omega_i \text{ then } t \vDash \|\varphi\|_{\mathcal{F}} \\
& \iff & \text{For all } t : \text{If } t \vDash \theta \text{ and } st' \vDash \Omega_i \text{ then } t \vDash \|\varphi\|_{\mathcal{F}} \\
& \iff & \text{For all } t : \text{If } t' \vDash \theta' \text{ and } st' \vDash \Omega_i \text{ then } t' \vDash (\|\varphi\|_{\mathcal{F}})' \\
& \iff & \text{For all } t : \text{If } st' \vDash \theta' \text{ and } st' \vDash \Omega_i \text{ then } st' \vDash (\|\varphi\|_{\mathcal{F}})' \\
& \iff & \text{For all } t : st' \vDash \theta' \to (\Omega_i \to (\|\varphi\|_{\mathcal{F}})') \\
& \iff & s \vDash \forall V' : (\theta' \to (\Omega_i \to (\|\varphi\|_{\mathcal{F}})'))
\end{array}$$

Second, for the case $\varphi = C_\Delta \psi$, let Λ be the operator defined in as in Definition 2.6.3. Also let $s_1 := s$, $\Lambda^0(\alpha) := \alpha$ and $\Lambda^{k+1}(\alpha) := \Lambda(\Lambda^k(\alpha))$.

For left to right, suppose $(\mathcal{F}, s_1) \vDash C_\Delta \psi$. As in the proof of Theorem 2.2.8 we show $s_1 \vDash \mathsf{gfp}\Lambda$ by proving $s_1 \vDash \Lambda^m(\top)$ for any m. Suppose not, i.e. there is an m such that $s_1 \nvDash \Lambda^m(\top)$. Then we have

$$s_1 \nvDash \forall V'(\theta' \to (\bigvee_{i \in \Delta} \Omega_i \to (\|\psi\|_{\mathcal{F}} \wedge \Lambda^{m-1}(\top))')).$$

Hence there must be some assignment $s_2' \subseteq V'$ such that $s_2' \vDash \theta'$, and an agent $i \in \Delta$ such that $s_1 s_2' \vDash \Omega_i$ and $s_2' \nvDash (\|\psi\|_{\mathcal{F}} \wedge \Lambda^{m-1}(\top))'$. By removing the primes we get that $s_2 \vDash \theta$, so s_2 is a state of $\mathcal{F}$, and $s_2 \nvDash \|\psi\|_{\mathcal{F}} \wedge \Lambda^{m-1}(\top)$. By boolean semantics we have $s_2 \nvDash \|\psi\|_{\mathcal{F}}$ or $s_2 \nvDash \Lambda^{m-1}(\top)$. But the first cannot be: s_2 is a state of $\mathcal{F}$ and by $s_1 s_2' \vDash \Omega_i$ we have $(s_1, s_2) \in \mathcal{E}_\Delta$. Thus $(\mathcal{F}, s_1) \vDash C_\Delta \psi$ implies $(\mathcal{F}, s_2) \vDash \psi$ which by induction hypothesis gives $s_2 \vDash \|\psi\|_{\mathcal{F}}$. Hence the second must hold. Spelling it out we get

$$s_2 \nvDash \forall V'(\theta' \to (\bigvee_{i\in\Delta} \Omega_i \to (\|\psi\|_{\mathcal{F}} \land \Lambda^{m-2}(\top))')).$$

But this means there has to be a state s_3 to continue. Iterating the reasoning m times we get an $\mathcal{E}_\Delta$-chain of states $s_1, \ldots, s_m$ such that $s_{1+k} \vDash \|\psi\|_{\mathcal{F}}$ and $s_{1+k} \nvDash \Lambda^{m-k}(\top)$ for all $k \in \{1, \ldots, m-1\}$. At the end of the chain with $k = m-1$ we have $s_m \vDash \|\psi\|_{\mathcal{F}}$ and $s_m \nvDash \Lambda(\top)$. But then $s_m \nvDash \top$. Contradiction! Hence $s_1 \vDash \Lambda^m(\top)$ must hold for all m.

For right to left, suppose $s_1 \vDash \mathsf{gfp}\Lambda$. Note that $\mathsf{gfp}\Lambda \to \Lambda^k(\top)$ is valid and thus we have $s_1 \vDash \Lambda^k(\top)$ for any k. To show $(\mathcal{F}, s) \vDash C_\Delta\varphi$, fix any state t of $\mathcal{F}$ such that $(s,t) \in \mathcal{E}_\Delta^*$. We have to show $(\mathcal{F}, t) \vDash \psi$. By definition of $\mathcal{E}_\Delta^*$ there is a chain $s_1, \ldots, s_m = t$ of states of $\mathcal{F}$ and there are agents $i_1, \ldots, i_{m-1} \in \Delta$ such that for all $k \in \{1, \ldots, m-1\}$ we have $s_k s_{k+1}' \vDash \Omega_{i_k}$. Note that $s_1 \vDash \Lambda^m(\top)$, i.e. $s_1 \vDash \forall V'(\theta' \to (\bigvee_{i\in\Delta} \Omega_i \to (\|\psi\|_{\mathcal{F}} \land \Lambda^{m-1}(\top))'))$. This implies $s_1 \vDash \forall V'(\theta' \to (\Omega_{i_1} \to \Lambda^{m-1}(\top)))$. Because s_2 is a state of $\mathcal{F}$ we have $s_2' \vDash \theta'$. Together with $s_1 s_2' \vDash \Omega_{i_1}$ we thus get $s_2 \vDash \Lambda^{m-1}(\top)$. Iterating this, we follow the chain to get $s_{1+k} \vDash \Lambda^{m-k}(\top)$ for all $k \in \{1, \ldots, m-1\}$. In particular $s_m \vDash \Lambda(\top)$ which implies $s_m \vDash \|\psi\|_{\mathcal{F}}$. By $s_m = t$ and the induction hypothesis this shows $(\mathcal{F}, t) \vDash \psi$. □

2.6.5. Fact. Belief structures are a generalization of knowledge structures: Any set of observational variables O can also be encoded using the BDD of the boolean formula $\Omega(O) := \bigwedge_{p\in O}(p \leftrightarrow p')$. This describes the same relation as O, because for any two states s and t we have $s \cup t' \vDash \Omega(O)$ iff $s \cap O = t \cap O$.

2.6.6. Example. We can also model Example 1.3.4 as a private announcement on belief structures. The initial structure is

$$\mathcal{F} = (V = \{p\}, \theta = \top, \Omega_{\text{Alice}} = \top, \Omega_{\text{Bob}} = \top)$$

and after the update we have

$$\mathcal{F}_p^{\text{Alice}} = (V = \{p, p_p\}, \theta = (p_p \to p), \Omega_{\text{Alice}} = (p_p \leftrightarrow p_p'), \Omega_{\text{Bob}} = \neg p_p')$$

We can see how this corresponds to the second Kripke model in Figure 1.3: First note that the state law θ is satisfied by the three states $\varnothing$, $\{p\}$ and $\{p, p_p\}$ which we can identify respectively with the worlds in the top left, top right and bottom. The observation law Ω_{Alice} then says that the upper and the lower part of the model are disconnected for Alice, whereas Bob almost has a total relation encoded in Ω_{Bob} up to the lower world being unreachable.

As the reader will already expect, such a correspondence between general Kripke models and belief structures can also be made precise. The following generalizes Lemma 2.4.1. The only difference is in condition C1, which now deals with observation laws instead of observational variables.

2.6.7. LEMMA. *Suppose we have a belief structure $\mathcal{F} = (V, \theta, \Omega)$ and a finite Kripke model $M = (W, \pi, R)$ with a set of primitive propositions $U \subseteq V$. Furthermore, suppose we have a function $g \colon W \to \mathcal{P}(V)$ such that*

C1 For all $w_1, w_2 \in W$ and $i \in I$, we have that $g(w_1)(g(w_2)') \vDash \Omega_i$ iff $R_i w_1 w_2$.

C2 For all $w \in W$ and $p \in U$, we have that $p \in g(w)$ iff $p \in \pi(w)$.

C3 For every $s \subseteq V$, s is a state of $\mathcal{F}$ iff $s = g(w)$ for some $w \in W$.

Then, for every $\mathcal{L}(U)$-formula φ we have $(\mathcal{F}, g(w)) \vDash \varphi$ iff $(\mathcal{M}, w) \vDash \varphi$.

Proof:
By induction on φ, the same as for Lemma 2.4.1 up to the following cases:

1. Belief: If φ is of the form $\Box_i \psi$, then by Definition 2.6.2, we have $(\mathcal{F}, g(w)) \vDash \Box_i \psi$ iff $(\mathcal{F}, s) \vDash \psi$ for all states s of $\mathcal{F}$ with $g(w)s' \vDash \Omega_i$. By C3 this is equivalent to having $(\mathcal{F}, g(w')) \vDash \psi$ for all $w' \in W$ with $g(w)g(w')' \vDash \Omega_i$, which by C1 is equivalent to $(\mathcal{F}, g(w')) \vDash \psi$ for all $w' \in W$ with $R_i w w'$. Now by the induction hypothesis, this is equivalent to $(\mathcal{M}, w') \vDash \psi$ for all $w' \in W$ with $R_i w w'$ which is exactly $(\mathcal{M}, w) \vDash \Box_i \psi$ by Definition 1.1.3.

2. Common Belief: Suppose φ is of the form $C_\Delta \varphi$. Recall that for arbitrary states s and t of $\mathcal{F}$, $(s, t) \in \mathcal{E}_\Delta$ iff there exists an $i \in \Delta$ with $st' \vDash \Omega_i$. By C1 we have, for all $w_1, w_2 \in W$:

$$(g(w_1), g(w_2)) \in \mathcal{E}_\Delta \text{ iff } (w_1, w_2) \in \bigcup_{i \in \Delta} R_i$$

   Now the exact same reasoning as in the proof of Lemma 2.4.1 applies: This iff-statement still holds if we take the transitive closure on both sides. Then use C3 and the induction hypothesis for ψ to get $(\mathcal{F}, g(w)) \vDash C_\Delta \psi$ iff $(\mathcal{M}, w) \vDash C_\Delta \psi$.

3. Public announcements: Suppose φ is of the form $[!\psi]\xi$. Just like in the proof for Lemma 2.4.1 it suffices to show that $(\mathcal{M}^\psi, W) \vDash \xi$ iff $(\mathcal{F}^\psi, g(w)) \vDash \xi$. To do so, let g' be the restriction of g to $W^{\mathcal{M}^\psi} = \{w \in W \mid (\mathcal{M}, w) \vDash \psi\}$. It remains to show that g' fulfills C1 to C3. The new C1 is again a universal condition and holds for g on W^M, hence it must also hold for g' with respect to the restricted set $W^{\mathcal{M}^\psi} \subseteq W^M$. Conditions C2 and C3 are unchanged, hence the proof for Lemma 2.4.1 still applies. Together, g' fulfills all three conditions and by the induction hypothesis we get that $(\mathcal{M}^\psi, W) \vDash \xi$ iff $(\mathcal{F}^\psi, g(w)) \vDash \xi$. $\Box$

We now also generalize the translation methods from Definitions 2.4.2 and 2.4.5 to belief structures. The new Lemma 2.6.7 then allows us to show the correctness of our translations and get generalized versions of Theorems 2.4.3 and 2.4.6: For every belief structure there is an equivalent Kripke model and vice versa.

2.6.8. DEFINITION. For any belief structure $\mathcal{F} = (V, \theta, \Omega)$, we define the Kripke model $\mathcal{M}(\mathcal{F}) := (W, \pi, R)$ as follows:

1. W is the set of all states of $\mathcal{F}$.

2. For each $w \in W$, let the assignment $\pi(w)$ be w itself.

3. For each agent i and all $w, w' \in W$, let $R_i ww'$ iff $ww' \vDash \Omega_i$.

2.6.9. DEFINITION. For any finite Kripke model $\mathcal{M} = (W, \pi, R)$ we define a belief structure $\mathcal{F}(\mathcal{M})$ as follows. Without loss of generality we assume unique valuations, i.e. that for all $w, w' \in W$ we have $\pi(w) \neq \pi(w')$. If this is not the case, we can add propositions to V and extend π in such a way that $\pi(w) \neq \pi(w')$ for all $w, w' \in W$. The maximum number of propositions we might have to add is $\lceil \log_2 |W| \rceil$. Let $\mathcal{F}(\mathcal{M}) := (V, \theta_M, \Omega)$ where

1. V is the vocabulary of $\mathcal{M}$, including extra propositions to make π injective,

2. $\theta_M := \bigvee\{s \sqsubseteq V \mid \exists w \in W : \pi(w) = s\}$ using $\sqsubseteq$ from Definition 1.0.1.

3. For each i the boolean formula $\Omega_i := \Phi(R_i)$ represents the relation R_i on $\mathcal{P}(V)$ given by $R_i st$ iff $\exists v, w \in W : \pi(v) = s \wedge \pi(w) = t \wedge R_i vw$.

Different from Definition 2.4.5, in Definition 2.6.9 we do not have to add propositions to distinguish all equivalence classes of all agents. This is because the Ω_i can carry more information than the simple sets of observed variables O_i.

2.6.10. THEOREM. *For any belief structure $\mathcal{F}$, any state s of $\mathcal{F}$ and any formula φ, we have $(\mathcal{F}, s) \vDash \varphi$ iff $(\mathcal{M}(\mathcal{F}), s) \vDash \varphi$.*

Proof:
By Lemma 2.6.7 using the identity function for g. □

2.6.11. THEOREM. *For any finite pointed Kripke model $(\mathcal{M}, w)$ and every formula φ, we have that $(\mathcal{M}, w) \vDash \varphi$ iff $(\mathcal{F}(\mathcal{M}), g(w)) \vDash \varphi$.*

Proof:
We have to check that Lemma 2.6.7 applies to Definition 2.6.9. As we already assume unique valuations in $\mathcal{M}$, the appropriate injective function $g \colon W \to \mathcal{P}(V)$ is the same as the valuation function $g(w) := \{p \in V \mid p \in \pi(w)\}$.

To show C1, take any $w_1, w_2 \in W$ and $i \in \{1, \dots, n\}$ and note that we have $g(w_1)g(w_2)' \vDash \Omega_i$ iff $\pi(w_1)\pi(w_2)' \vDash \Phi(R_i)$ iff $R_i w_1 w_2$.

For C2, take any $w \in W$ and any $v \in U$. By definition of g we have $v \in g(w)$ iff $v \in \pi(w)$.

For the right-to-left part of C3: If $s = g(w)$ for some $w \in W$, then by the definition of θ_M, we have that $g(w) \vDash \theta_M$ and hence $g(w)$ is a state of $\mathcal{F}(\mathcal{M})$. For the left-to-right part, suppose s is a state of $\mathcal{F}(\mathcal{M})$. Then $s \vDash \theta_M$, hence it must satisfy one of the disjuncts and there must be a $w \in W$ such that $s \vDash g(w) \sqsubseteq V$. Now by definition of $\sqsubseteq$ we have $s = g(w) = \pi(w)$.

Now the theorem follows from Lemma 2.6.7. □

Given this symbolic representation of Kripke models with arbitrary relations, one might wonder whether graph properties characterized by modal formulas from Table 1.1 correspond to properties of boolean formulas. The answer is positive.

2.6.12. Example. The frame properties for a relation R are related to properties of its symbolic encoding $\Phi(R) \in \mathcal{L}_B(V \cup V')$ as follows.

- The total relation is given by $\Phi(R) \equiv \top$ and the empty relation by $\Phi(R) \equiv \bot$.
- To compute the inverse $\Phi(R^{-1})$, simultaneously substitute primed for unprimed variables and vice versa in $\Phi(R)$.
- The relation R is symmetric iff $\Phi(R) \equiv \Phi(R^{-1})$. To get the symmetric closure, take $\Phi(R) \vee \Phi(R^{-1})$.
- Similarly, R is reflexive iff $\bigwedge_i (p_i \leftrightarrow p_i') \rightarrow \Phi(R)$ is a tautology and the reflexive closure is given by $\Phi(R) \vee \bigwedge_{p \in V} (p \leftrightarrow p')$.
- To check for transitivity we need to talk about three states at the same time which means we need a third copy of variables. Denote this third copy by V'', then we have that R is transitive iff the formula $\Phi(R) \wedge \Phi(R)' \rightarrow [V' \mapsto V'']\Phi(R)$ is a tautology. Note that this is a formula in $\mathcal{L}_B(V \cup V' \cup V'')$.
- Similarly, to compose two relations R_1 and R_2, we can take
$$\Phi(R_2 \circ R_1) := [V'' \mapsto V'](\exists V' : \Phi(R_1) \wedge \Phi(R_2)')$$
- Iterating this composition gives us the transitive closure: $\Phi(R^*)$ is given by $\mathsf{lfp}\Lambda$ where $\Lambda \colon \mathcal{L}_B(V \cup V') \rightarrow \mathcal{L}_B(V \cup V')$ is the operator
$$\Lambda(\alpha) := \Phi(R) \vee [V'' \mapsto V'] \, (\exists V' : \Phi(R) \wedge \alpha')$$
and lfp denotes the least fixpoint with respect to semantic equivalence.

We note that these conditions are similar to those obtained from relation algebra or matrix representations of Kripke models, see for example the analysis of bisimulations as linear functions in [Fit03], and the representation of Kripke semantics and communication between agents as matrix multiplication in [HST15].

2.7 Belief Transformers

The generalization from knowledge to belief structures is compatible with the one from public announcements to knowledge transformers: We will now define *belief transformers* in the same style as knowledge transformers in Definition 2.5.1, replacing the additional observational propositions O_i^+ with boolean formulas Ω_i^+ encoding a relation on $\mathcal{P}(V^+)$. In an implementation, those boolean formulas are of course again meant to be replaced by BDDs. Thus we obtain a symbolic representation of events where the epistemic relation between different actions need not be an equivalence relation, for example if someone is being deceived.

2.7.1. DEFINITION. A *belief transformer* for V is a tuple $\mathcal{X} = (V^+, \theta^+, \Omega^+)$ where V^+ is a set of atomic propositions such that $V \cap V^+ = \varnothing$, $\theta^+ \in \mathcal{L}(V \cup V^+)$ is a possibly epistemic formula and $\Omega_i^+ \in \mathcal{L}_B(V \cup V^+)$ is a boolean formula for each $i \in I$. A *belief event* is a belief transformer together with a subset $x \subseteq V^+$, written as $(\mathcal{X}, x)$.

The *belief transformation* of a belief structure $\mathcal{F} = (V, \theta, \Omega)$ with $\mathcal{X}$ is defined by $\mathcal{F} \times \mathcal{X} := (V \cup V^+, \theta \wedge \|\theta^+\|_{\mathcal{F}}, \{\Omega_i \wedge \Omega_i^+\}_{i \in I})$. Given a scene $(\mathcal{F}, s)$ and a belief event $(\mathcal{X}, x)$, let $(\mathcal{F}, s) \times (\mathcal{X}, x) := (\mathcal{F} \times \mathcal{X}, s \cup x)$.

The resulting observations are boolean formulas over a new double vocabulary

$$(V \cup V') \cup (V^+ \cup V^{+\prime}) \;=\; (V \cup V^+) \cup (V \cup V^+)'$$

describing a relation between the new states which are subsets of $V \cup V^+$.

Belief transformers share both the features and the problems of non-$\mathsf{S5}$ action models. As [Eij14b] says, "update of belief models with belief action models has a glitch": The result of updating a KD45 Kripke model (in which the relations are serial, transitive and Euclidean, but not necessarily reflexive) with a KD45 action model does not have to be KD45. Still, we consider it a feature of our symbolic methods that they agree with the standard explicit semantics.

We further generalize from belief transformers to transformers with factual change in the next section. Therefore we omit the definitions and theorems here to connect belief transformers and non-$\mathsf{S5}$ action models without factual change.

2.8 Symbolic Factual Change

Our knowledge and belief transformers so far only change what agents know and not what is actually the case — they do not provide a symbolic equivalent of postconditions for factual change as introduced in [BEK06] and included in our Definition 1.3.1.

Possible worlds in a Kripke model get their meaning via a valuation function, but not their identity. In particular, we can assign the same atomic truths to different possible worlds. In contrast, all states of our structures satisfy different atomic propositions and can thus be identified with their valuation. This is what makes structures symbolic and efficient to implement, but it complicates the idea of changing facts, as the following minimal example shows.

2.8.1. Example. Consider the coin flip from Example 1.3.5. It is easy to find the following structures that are equivalent to the initial and the resulting model, but how can we symbolically describe the update which transforms one into the other?

$$\begin{array}{rc} & (V = \{p\},\ \theta = p,\ O_a = \{p\},\ O_b = \{p\}) \\ \times & ??? \\ = & (V = \{p\},\ \theta = \top,\ O_a = \varnothing\ ,\ O_b = \{p\}) \end{array}$$

In product updates of Kripke and action models, the name of a resulting world (w, a_1) makes clear that it "comes from" w. In contrast, a state of a knowledge structure like $\varnothing$ does not reveal its history or any relation to the previous state $\{p\}$.

For purely epistemic actions this was not a problem — we only *add* propositions from V^+ to the state to distinguish different epistemic events. But for factual change, propositions from V have to be modified and we need a way to *remove* them from states.

Our solution is to *copy propositions*: We store the old value of p in a fresh variable p°. Then we rewrite the state law and observations using substitutions.

We now define transformation with factual change, adding the components V_- and θ_- to describe which propositions are changed and how. Note that the belief transformers without factual change as discussed in the previous section are exactly those transformers where $V_- = \varnothing$.

2.8.2. Definition. A belief transformer with factual change, also just called *transformer*, for the vocabulary V is a tuple $\mathcal{X} = (V^+, \theta^+, V_-, \theta_-, \Omega^+)$ where

- V^+ is a set of fresh atomic propositions such that $V \cap V^+ = \varnothing$,
- θ^+ is a possibly epistemic formula from $\mathcal{L}(V \cup V^+)$ called the *event law*,
- $V_- \subseteq V$ is a subset of the original vocabulary called the *modified subset*,
- $\theta_- \colon V_- \to \mathcal{L}_B(V \cup V^+)$ is a map from modified propositions to boolean formulas called the *change law*,
- $\Omega_i^+ \in \mathcal{L}_B(V^+ \cup V^{+\prime})$ is a boolean formula for each agent $i \in I$ called the *event observation law*.

To transform a belief structure $\mathcal{F} = (V, \theta, \Omega_i)$ with $\mathcal{X}$, we define a new belief structure $\mathcal{F} \times \mathcal{X} := (V^{\text{new}}, \theta^{\text{new}}, \Omega_i^{\text{new}})$ where

1. $V^{\text{new}} \ := \ V \cup V^+ \cup V_-^\circ$
2. $\theta^{\text{new}} \ := \ [V_- \mapsto V_-^\circ] \, (\theta \land \|\theta^+\|_{\mathcal{F}}) \ \land \ \bigwedge_{q \in V^-} \big(q \leftrightarrow [V_- \mapsto V_-^\circ](\theta_-(q))\big)$
3. $\Omega_i^{\text{new}} \ := \ \big([V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i\big) \land \Omega_i^+$

An *event* is a pair $(\mathcal{X}, x)$ where $x \subseteq V^+$. Given a scene $(\mathcal{F}, s)$ and an event $(\mathcal{X}, x)$, let $(\mathcal{F}, s) \times (\mathcal{X}, x) := (\mathcal{F} \times \mathcal{X}, s^x)$ where the new actual state is given by:

$$s^x \ := \ (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- \mid s \cup x \vDash \theta_-(p)\}$$

To explain this definition, let us consider the components one by one.

First, the new vocabulary V^{new} besides V and V^+ now also contains $V_-^\circ = \{p^\circ \mid p \in V_-\}$. These are fresh copies of the modified subset. We use them to keep track of the old valuation.

Second, the new state law θ^{new}: A state in the resulting structure needs to satisfy the old state law and the event law encoding the preconditions. For modified propositions the old values have to be used, hence we apply a substitution to both laws in the left conjunct. Modified propositions are then overwritten in the right conjunct, using θ_- which encodes postconditions. As in Definition 1.3.1, postconditions are evaluated in the old model, hence we also substitute here.

Third, to define the new observations Ω_i^{new} we replace modified variables by their copies. Two substitutions are needed because Ω_i is in a double vocabulary. Old observations induce new ones via the state law. For example, if q was flipped publicly, then $q \leftrightarrow \neg q^\circ$ is part of the new state law and observing whether q is equivalent to observing whether $\neg q^\circ$, i.e. having observed q in the original structure.

Finally, the new actual state s^x is the union of four sets: propositions in the old state that have not been modified $(s \setminus V_-)$, copies of the modified propositions that were in the old state $(s \cap V_-)^\circ$, those propositions labeling the actual event x and the modified propositions whose precondition was true in the old state $\{p \in V_- \mid s \cup x \vDash \theta_-(p)\}$.

Note that we do not make it part of the definition of transformation that the encoded precondition has to hold: $(\mathcal{F}, s) \times (\mathcal{X}, x)$ is well-defined even if we do not have $\mathcal{F}, s \vDash [a \sqsubseteq V^+] \, \theta^+$. But then it yields a tuple where the second element, the resulting actual state, is not a state of the first element, the resulting structure. In practice and in Definition 2.10.2 below we check the encoded precondition before applying a transformer and only ever apply possible events — similar to how action models are used in Definition 1.3.7 above.

2.8.3. Example. We can now model the coin flip from Example 1.3.5 as follows. Because we use the more general belief (instead of knowledge) structures, the initial structure now has observational laws Ω_i instead of sets of observational variables O_i:

$$(V = \{p\},\ \theta = p,\ \Omega_a = p \leftrightarrow p',\ \Omega_b = p \leftrightarrow p')$$

The following transformer models the coin flip visible to b but not to a. We use q to label the two different events, representing different outcomes of the coin flip.

$$(V^+ = \{q\},\ \theta^+ = \top,\ V_- = \{p\},\ \theta_-(p) := q,\ \Omega_a^+ = \top,\ \Omega_b^+ = q \leftrightarrow q')$$

The result of applying the latter to the former is this:

$$(V = \{p, q, p^\circ\},\ \theta = p^\circ \wedge (p \leftrightarrow q),\ \Omega_a = p^\circ \leftrightarrow p^{\circ\prime},\ \Omega_b = (p^\circ \leftrightarrow p^{\circ\prime}) \wedge (q \leftrightarrow q'))$$

This is not syntactically identical but still equivalent to the resulting structure we gave in Example 2.8.1, as we will discuss in Sections 2.11 and 2.12.

2.8.4. Example. In general, a publicly observable change $p := \varphi$ which sets the truth value of p to the current truth value of a propositional formula φ can be modeled by this transformer:

$$(V^+ = \varnothing,\ \theta^+ = \top,\ V_- = \{p\},\ \theta_-(p) := \varphi,\ \Omega_i^+ = \top)$$

To conclude this section, we note that factual change can be added to knowledge transformers instead of belief transformers in the same way. The following definition shows what needs to be changed: We apply the substitution $[V_- \mapsto V_-^\circ]$ to the observational variables O_i instead of the observation laws Ω_i.

2.8.5. Definition. A *knowledge transformer with factual change* for the vocabulary V is a tuple $\mathcal{X} = (V^+, \theta^+, V_-, \theta_-, O^+)$, where V^+, θ^+, V_- and θ_- are the same as for transformers in Definition 2.8.2 and $O_i^+ \subseteq V^+$ is a subset of V^+ for each agent $i \in I$, called the *event observation*.

To transform a knowledge structure $\mathcal{F} = (V, \theta, O_i)$ with a knowledge transformer with factual change $\mathcal{X}$, let $\mathcal{F} \times \mathcal{X} := (V^{\text{new}}, \theta^{\text{new}}, O_i^{\text{new}})$ where V^{new} and θ^{new} are the same as in Definition 2.8.2 and $O_i^{\text{new}} := \big([V_- \mapsto V_-^\circ]O_i\big) \cup O_i^+$.

To transform a scene $(\mathcal{F}, s)$ where $\mathcal{F}$ is a knowledge structure, with an event $(\mathcal{X}, x)$ where $\mathcal{X}$ is a knowledge transformer with factual change, the actual state of the result is the same as in Definition 2.8.2.

The implementation which we introduce in Chapter 3 implements both knowledge and belief transformers with factual change, because using observational variables instead of laws, when applicable, always uses less memory.

2.9 Equivalence Proof for the General Case

We now show that transformers describe exactly the same class of updates as action models. The main ingredients for the proof are Lemma 2.6.7, to show that a Kripke model and a belief structure are equivalent, and two definitions, to go from transformers to action models and vice versa.

This is the most general setting we consider in this thesis. The following two definitions generalize the previous translations for $\mathsf{S5}$ updates in Definitions 2.5.4 and 2.5.5, respectively.

2.9.1. DEFINITION. The function Act maps transformers to action models as follows. Given an event $(\mathcal{X} = (V^+, \theta^+, V_-, \theta_-, \Omega^+), x)$, we define an action $(\mathsf{Act}(\mathcal{X}) := (A, \mathsf{pre}, \mathsf{post}, R), x)$ by

- $A := \mathcal{P}(V^+)$
- $\mathsf{pre}(a) := [a \sqsubseteq V^+]\, \theta^+$
- $\mathsf{post}_a(p) := \begin{cases} [a \sqsubseteq V^+]\, (\theta_-(p)) & \text{if } p \in V_- \\ p & \text{otherwise} \end{cases}$
- $R_i := \{(a, b) \mid a \cup b' \vDash \Omega_i^+\}$

where b' denotes a copy of b as in Definition 1.8.8.

2.9.2. DEFINITION. We define the function Trf mapping action models to transformers. Consider an action $(\mathcal{A} = (A, \mathsf{pre}, \mathsf{post}, R), a_0)$. Let $n := \lceil \log_2 |A| \rceil$ and let $\ell\colon A \to \mathcal{P}(\{q_1, \ldots, q_n\})$ be an injective labeling function using fresh atomic variables q_k.

Then let $(\mathsf{Trf}(\mathcal{A}) := (V^+, \theta^+, V_-, \theta_-, \Omega^+), \ell(a_0))$ be the event defined by

- $V^+ := \{q_1, \ldots, q_n\}$
- $\theta^+ := \bigvee_{a \in A} (\mathsf{pre}(a) \wedge \ell(a) \sqsubseteq V^+)$
- $V_- := \{p \in V \mid \exists a : \mathsf{post}_a(p) \neq p\}$
- $\theta_-(p) := \bigvee_{a \in A} (\ell(a) \sqsubseteq V^+ \wedge \mathsf{post}_a(p))$
- $\Omega_i^+ := \bigvee_{(a,b) \in R_i} (\ell(a) \sqsubseteq V^+ \wedge (\ell(b) \sqsubseteq V^+)')$

Besides these translations for the dynamic parts, in the following we will also use the translations $\mathcal{M}(\cdot)$ and $\mathcal{F}(\cdot)$ from belief structures to Kripke models and vice versa, as given in Definitions 2.6.8 and 2.6.9. Now everything is in place to state and prove the main result of this section. The following generalizes Theorem 2.5.7.

2.9.3. Theorem.

(i) Act *from Definition 2.9.1 is truth-preserving: For any scene* $(\mathcal{F}, s)$, *any event* $(\mathcal{X}, x)$ *and any formula* φ *over the vocabulary of* $\mathcal{F}$ *we have:*

$$(\mathcal{F}, s) \times (\mathcal{X}, x) \vDash \varphi \iff (\mathcal{M}(\mathcal{F}), s) \times (\mathsf{Act}(\mathcal{X}), x) \vDash \varphi$$

(ii) Trf *from Definition 2.9.2 is truth-preserving: For any pointed model* $(\mathcal{M}, w)$, *any action* $(\mathcal{A}, a)$ *and any formula* φ *over the vocabulary of* $\mathcal{M}$ *we have:*

$$(\mathcal{M} \times \mathcal{A}, (w, a)) \vDash \varphi \iff (\mathcal{F}(\mathcal{M}), g_{\mathcal{M}}(w)) \times (\mathsf{Trf}(\mathcal{A}), \ell(a)) \vDash \varphi$$

where $g_{\mathcal{M}}$ *is the possibly extended valuation* π *from* $\mathcal{F}(\mathcal{M})$ *in Definition 2.6.9.*

Proof:
By Lemma 2.6.7. We first need appropriate functions g. For part (i), g needs to map worlds of $\mathcal{M}(\mathcal{F}) \times \mathsf{Act}(\mathcal{X})$, i.e. pairs $(s, x) \in \mathcal{P}(V) \times \mathcal{P}(V^+)$, to states of $\mathcal{F} \times \mathcal{X}$, i.e. subsets of $V \cup V^+ \cup V_-^\circ$. Let

$$g(s, x) := (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- \mid s \cup x \vDash \theta_-(p)\}$$

which is exactly s^x from Definition 2.8.2 above. We prove the conditions C1 to C3 from Lemma 2.6.7 for this g.

For C1, take any two worlds (s, x) and (t, y). We need to show $g(s, x)(g(t, y))' \vDash \Omega_i^{\text{new}}$ iff $R_i^{\text{new}}(s, x)(t, y)$. For this, start on the left side and note the following equivalences. We have $g(s, x)(g(t, y))' \vDash \Omega_i^{\text{new}}$ iff

$$\begin{array}{ll} & (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- \mid s \cup x \vDash \theta_-(p)\} \\ \cup & ((t \setminus V_-) \cup (t \cap V_-)^\circ \cup y \cup \{p \in V_- \mid t \cup y \vDash \theta_-(p)\})' \\ \vDash & [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i \wedge \Omega_i^+ \end{array}$$

Here V_- and V_-' do not occur in the formula, as old epistemic relations do not depend on new values of modified propositions. Hence we can drop the subsets of V_- and V_-' to obtain the equivalent condition

$$(s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup (t \setminus V_-)' \cup (t^\circ \cap V_-^\circ)' \cup y' \vDash [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i \wedge \Omega_i^+$$

in which we can split both sides into separate vocabularies:

$$(s \setminus V_-) \cup (s \cap V_-)^\circ \cup (t \setminus V_-)' \cup (t^\circ \cap V_-^\circ)' \vDash [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i$$

$$\text{and } x \cup y' \vDash \Omega_i^+$$

Now undo the $\circ$-substitution on both sides in the first conjunct to see that it is equivalent to $s \cup t' \vDash \Omega_i$. Hence the whole condition is equivalent to $R_i^{\mathcal{M}} st$ and $R_i^{\mathcal{A}} xy$, which is exactly $R_i^{\text{new}}(s, x)(t, y)$ by definition of $\mathcal{M}(\cdot)$ and Definition 2.9.1.

To show C2, take any (s, x) and any $p \in V$. We have to show that $p \in g(s, x)$ iff $p \in \pi^{\text{new}}(s, x) = \{p \in V \mid \mathcal{M}, s \vDash \mathsf{post}_x(p)\}$. There are two cases. First, if $p \notin V_-$, then $\mathsf{post}_x(p) = p$ by Definition 2.9.1 and we directly have $p \in g(s, x)$ iff $p \in s$ iff $\mathcal{M}, s \vDash p$ iff $p \in \pi^{\text{new}}(s, x)$. Second, if $p \in V_-$, then $p \in g(s, x)$ iff $s \cup x \vDash \theta_-(p)$ by definition of g and $\mathsf{post}_x(p) = [x \sqsubseteq V^+]\theta_-(p)$ by Definition 2.9.1. Hence we have a chain of equivalences: $p \in g(s, x)$ iff $s \cup x \vDash \theta_-(p)$ iff $s \vDash [x \sqsubseteq V^+]\theta_-(p)$ iff $\mathcal{M}, s \vDash [x \sqsubseteq V^+]\theta_-(p)$ iff $p \in \pi^{\text{new}}(s, x)$.

For C3, take any $s^{\text{new}} \subseteq V \cup V^+ \cup V_-^\circ$. We want to show that $s^{\text{new}} \vDash \theta^{\text{new}}$ iff there is a world (s, x) such that $g(s, x) = s^{\text{new}}$. For left-to-right, suppose $s^{\text{new}} \vDash \theta^{\text{new}}$, i.e.:

$$s^{\text{new}} \vDash [V_- \mapsto V_-^\circ]\left(\theta \wedge \|\theta^+\|_{\mathcal{F}}\right) \wedge \bigwedge_{q \in V^-} \left(q \leftrightarrow [V_- \mapsto V_-^\circ](\theta_-(q))\right) \tag{2.1}$$

Let $s := (s^{\text{new}} \cap (V \setminus V_-)) \cup \{p \in V_- \mid p^\circ \in s^{\text{new}}\}$. From 2.1 we then get $s \vDash \theta$, which means that s is a state of $\mathcal{F}$ and thus also a world of $\mathcal{M}(\mathcal{F})$. Second, let $x := s^{\text{new}} \cap V^+$. Now by definition of g we have $g(s, x) = s^x = s^{\text{new}}$.

For right-to-left, suppose we have a world (s, x) such that $g(s, x) = s^x = s^{\text{new}}$. We now have to show 2.1 above for $s^{\text{new}} = (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- \mid s \cup x \vDash \theta_-(p)\}$.

First note that s is a world of $\mathcal{M}(\mathcal{F})$ and thus a state of $\mathcal{F}$, i.e. we have $s \vDash \theta$. Second, (s, x) is a world of $\mathcal{M}(\mathcal{F}) \times \mathsf{Act}(\mathcal{X})$, hence we have $s \cup x \vDash \|\theta^+\|_{\mathcal{F}}$. Third, we have by definition of g that for all $q \in V_-$ on the one hand $q^\circ \in s^{\text{new}}$ iff $q \in s$ and on the other hand $q \in s^{\text{new}}$ iff $s \vDash \theta_-(q)$. All three together imply 2.1.

For part (ii), g should map worlds of the model $\mathcal{M} \times \mathcal{A}$ to states of the structure $\mathcal{F}(\mathcal{M}) \times \mathsf{Trf}(\mathcal{A})$. Again we use s^x from above, but s and x are now given by the propositional encodings $g_{\mathcal{M}}(w)$ and $\ell(a)$. Let g be defined by this:

$$g(w, a) := (g_{\mathcal{M}}(w) \setminus V_-) \cup (g_{\mathcal{M}}(w) \cap V_-)^\circ \cup \ell(a) \cup \{p \in V_- \mid g_{\mathcal{M}}(w) \cup \ell(a) \vDash \theta_-(p)\}$$

We now have to check C1 to C3 again for this g. The proofs follow the same pattern as in part (i), with $g_{\mathcal{M}}(w)$ and $\ell(a)$ taking the role of s and x respectively.

For C1, take any two worlds (w_1, a_1) and (w_2, a_2) in the updated model. We need to show that $g(w_1, a_1)(g(w_2, a_2))' \vDash \Omega_i^{\text{new}}$ iff $R_i^{\text{new}}(s, x)(t, y)$. For this, note the following equivalences. We have $g(w_1, a_1)(g(w_2, a_2))' \vDash \Omega_i^{\text{new}}$ iff

$$\begin{aligned} &(g_{\mathcal{M}}(w_1) \setminus V_-) \cup (g_{\mathcal{M}}(w_1) \cap V_-)^\circ \cup \ell(a_1) \cup \{p \in V_- \mid g_{\mathcal{M}}(w_1) \cup \ell(a_1) \vDash \theta_-(p)\} \cup \\ &((g_{\mathcal{M}}(w_2) \setminus V_-) \cup (g_{\mathcal{M}}(w_2) \cap V_-)^\circ \cup \ell(a_2) \cup \{p \in V_- \mid g_{\mathcal{M}}(w_2) \cup \ell(a_2) \vDash \theta_-(p)\})' \\ &\vDash [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i \wedge \Omega_i^+ \end{aligned}$$

Again V_- and V_-' do not occur in the formula, so we can drop the subsets of V_- and V_-' to obtain the equivalent condition

$$\begin{array}{ll} & (g_{\mathcal{M}}(w_1) \setminus V_-) \cup (g_{\mathcal{M}}(w_1) \cap V_-)^\circ \cup \ell(a_1) \\ \cup & (g_{\mathcal{M}}(w_2) \setminus V_-)' \cup (g_{\mathcal{M}}(w_2)^\circ \cap V_-^\circ)' \cup \ell(a_2)' \\ \vDash & [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i \wedge \Omega_i^+ \end{array}$$

which we can split into separate vocabularies:

$$\begin{array}{ll} & (g_{\mathcal{M}}(w_1) \setminus V_-) \cup (g_{\mathcal{M}}(w_1) \cap V_-)^\circ \\ \cup & (g_{\mathcal{M}}(w_2) \setminus V_-)' \cup (g_{\mathcal{M}}(w_2)^\circ \cap V_-^\circ)' \\ \vDash & [V_- \mapsto V_-^\circ][(V_-)' \mapsto (V_-^\circ)']\Omega_i \end{array}$$

$$\text{and } \ell(a_1) \cup \ell(a_2)' \vDash \Omega_i^+$$

Now we can undo the $\circ$-substitution on both sides in the first conjunct to see that it is equivalent to $g_{\mathcal{M}}(w_1) \cup g_{\mathcal{M}}(w_2)' \vDash \Omega_i$. In the proof of Theorem 2.6.8 we have shown that the conditions of Lemma 2.6.7 also apply to $g_{\mathcal{M}}$ in the construction of $\mathcal{F}(\mathcal{M})$. In particular, by condition C1 we have $g_{\mathcal{M}}(w_1) \cup g_{\mathcal{M}}(w_2)' \vDash \Omega_i$ iff $R_i^{\mathcal{M}} w_1 w_2$. Moreover, by Definition 2.9.2 we have $\ell(a_1) \cup \ell(a_2)' \vDash \Omega_i^+$ iff $R_i^{\mathcal{A}} a_1 a_2$. Hence the combined condition is equivalent to $R_i^{\mathcal{M}} w_1 w_2$ and $R_i^{\mathcal{A}} a_1 a_2$. By Definition 1.3.1 for the product update, this is exactly $R_i^{\text{new}}(w_1, a_1)(w_2, a_2)$.

To show C2, take any world (w, a) of $\mathcal{M} \times \mathcal{A}$ and any $p \in V$. We have to show that $p \in g(w, a)$ iff $p \in \pi^{\text{new}}(w, a) = \{p \in V \mid \mathcal{M}, s \vDash \mathsf{post}_a(p)\}$. Consider the set $V_- := \{p \in V \mid \exists a : \mathsf{post}_a(p) \neq p\}$ from Definition 2.9.1 and distinguish two cases.

First, if $p \notin V_-$, then $\mathsf{post}_a(p) = p$ and we directly have $p \in g(w, a)$ iff $p \in g_{\mathcal{M}}(w)$ iff $\mathcal{M}, w \vDash p$ iff $p \in \pi^{\text{new}}(w, a)$.

Second, if $p \in V_-$, then $p \in g(w, a)$ iff $g_{\mathcal{M}}(w) \cup \ell(a) \vDash \theta_-(p)$ by definition of g. Note that $\theta_-(p)$ in Definition 2.9.2 is a disjunction over all actions in $\mathcal{A}$ with mutually exclusive disjuncts, because only one action can actually take place. Hence we have the following chain of equivalences: $p \in g(w, a)$ iff $g_{\mathcal{M}}(w) \cup \ell(a) \vDash \theta_-(p)$ iff $g_{\mathcal{M}}(w) \cup \ell(a) \vDash \bigvee_{a \in A} (\ell(a) \sqsubseteq V^+ \wedge \mathsf{post}_a(p))$ iff $g_{\mathcal{M}}(w) \vDash \mathsf{post}_a(p)$ iff $p \in \pi^{\text{new}}(s, x)$.

For C3, take any $s^{\text{new}} \subseteq V \cup V^+ \cup V_-^\circ$. We want to show that $s^{\text{new}} \vDash \theta^{\text{new}}$ iff there is a world (w, a) such that $g(w, a) = s^{\text{new}}$. To prepare both directions, let us "take apart" s^{new} into $s := (s^{\text{new}} \cap (V \setminus V_-)) \cup \{p \in V_- \mid p^\circ \in s^{\text{new}}\}$ and $x := s^{\text{new}} \cap V^+$.

For left-to-right, suppose $s^{\text{new}} \vDash \theta^{\text{new}}$, i.e.:

$$s^{\text{new}} \vDash [V_- \mapsto V_-^\circ] \left(\theta \wedge \|\theta^+\|_{\mathcal{F}}\right) \ \wedge \ \bigwedge_{q \in V^-} \left(q \leftrightarrow [V_- \mapsto V_-^\circ](\theta_-(q))\right) \tag{2.2}$$

From 2.2 we then get $s \vDash \theta$, which means that s is a state of $\mathcal{F}(\mathcal{M})$. By condition C3 there must be a world w of $\mathcal{M}$ such that $g_{\mathcal{M}}(w) = s$. Remember that we defined $x = s^{\text{new}} \cap V^+$. From 2.2 we have $s \cup x \vDash \|\theta^+\|_{\mathcal{F}}$. By Definition 2.9.2 we have $\theta^+ := \bigvee_{a \in A} (\mathsf{pre}(a) \wedge \ell(a) \sqsubseteq V^+)$. Hence there must be an action $a \in A$ such that $\ell(a) = x$ and $s \cup x \vDash \|\mathsf{pre}(a)\|_{\mathcal{F}}$, which implies $\mathcal{M}, w \vDash \mathsf{pre}(a)$. Hence (w, a) is a world of $\mathcal{M} \times \mathcal{A}$. Moreover, from 2.2 we get for all $q \in V_-$ that $q \in s^{\text{new}}$ iff $s^{\text{new}} \vDash \mathsf{post}_a(q)$. Hence by definition of g we have $g(w, a) = s^{\text{new}}$.

For right-to-left, suppose we have a world (w, a) in $\mathcal{M} \times \mathcal{A}$ such that $g(w, a) = s^{\text{new}}$. We now have to show $s^{\text{new}} \vDash \theta^{\text{new}}$, i.e. 2.2 above. By definitions of s and

x we have $g_{\mathcal{M}}(w) = s$ and $\ell(a) = x$. First note that w is a world of $\mathcal{M}$ and thus $s = g_{\mathcal{M}}(w)$ is a state of $\mathcal{F}$, i.e. we have $s \vDash \theta$. Second, (w, a) is a world of $\mathcal{M} \times \mathcal{A}$, hence $\mathcal{M}, w \vDash \mathsf{pre}(a)$. With $g_{\mathcal{M}}(w) \cup \ell(a) = s \cup x$ therefore, we have $s \cup c \vDash \mathsf{pre}(a) \wedge \ell(a) \sqsubseteq V^+$ and thus $s \cup x \vDash \|\theta^+\|_{\mathcal{F}}$ by Definition 2.9.2. Third, we have by definition of g that for all $q \in V_-$ (i) $q^\circ \in s^{\text{new}}$ iff $q \in s$ and (ii) $q \in s^{\text{new}}$ iff $s \vDash \theta_-(q)$. All three together imply 2.2. □

Given the translations from explicit action models to symbolic transformers, and a proof of their correctness, we immediately obtain another connection, between Arrow Update Logic from Section 1.4 and our symbolic version of DEL with transformers.

2.9.4. THEOREM. *For every arrow update there is an equivalent transformer. More formally, for every arrow update U as defined in Definition 1.4.1 there is an event $(\mathcal{X}, x)$ such that*

$$\mathcal{M} * U, w \vDash \varphi \quad \textit{iff} \quad (\mathcal{F}(\mathcal{M}), g_{\mathcal{M}}(w)) \times (\mathcal{X}, x) \vDash \varphi$$

where $\mathcal{F}(M)$ and $g_{\mathcal{M}}(w)$ are as in Definition 2.6.9.

Proof:
It was shown in [KR11b] that for every arrow update U there is an equivalent action $(\mathcal{A}, a)$. This action model can be translated to a transformer $\mathcal{X}$ by Definition 2.9.2, which will be equivalent to $\mathcal{A}$ by Theorem 2.9.3.

Together, we have:

$$\mathcal{M} * U, w \vDash \varphi \text{ iff } \mathcal{M}, w \times (\mathcal{A}, a) \vDash \varphi \text{ iff } (\mathcal{F}(\mathcal{M}), g_{\mathcal{M}}(w)) \times (\mathsf{Trf}(\mathcal{A}), \ell(a)) \vDash \varphi$$

□

The translation procedure given by this proof is not efficient: Going from an arrow update U to an equivalent action model $\mathcal{A}$ can already lead to an exponential blow-up. Moreover, then going from $\mathcal{A}$ to $\mathsf{Trf}(\mathcal{A})$ using Definition 2.9.2 means we add $\lceil \log_2 |A| \rceil$ atomic propositions. At least the latter can be avoided by going directly from arrow updates to transformers, using the following definition.

2.9.5. DEFINITION. Given an arrow update U, we define a new set of fresh variables $V^+ := \{p_\varphi \mid \exists(\varphi, i, \chi) \in U \text{ or } \exists(\psi, i, \varphi) \in U\}$ with an element for each formula occurring in an arrow in U. Then we define a belief transformer $\mathsf{Trf}(U) := (V^+, \theta^+, \Omega^+)$, where $\theta^+ := \bigwedge_{p_\varphi \in V^+}(p_\varphi \leftrightarrow \varphi)$ and $\Omega_i^+ := \bigvee_{(\psi,i,\chi)\in U}\{p_\psi \wedge p'_\chi\}$.

2.9.6. THEOREM. *The function $\mathsf{Trf}(U)$ from Definition 2.9.5 is truth-preserving: For any Kripke model $\mathcal{M}$ and any arrow update U we have*

$$\mathcal{M} * U, w_0 \vDash \varphi \quad \textit{iff} \quad (\mathcal{F}(\mathcal{M}), g_{\mathcal{M}}(w_0)) \times (\mathsf{Trf}(U), x_0) \vDash \varphi$$

where $\mathcal{F}(M)$ and $g_{\mathcal{M}}(w)$ are as in Definition 2.6.9 and the actual event is given by $x_0 := \{p_\varphi \in V^+ \mid \mathcal{M}, w_0 \vDash \varphi\}$.

Proof:
We use Lemma 2.6.7, mapping worlds of the model $\mathcal{M} * U$ to states of the structure $(\mathcal{F}(\mathcal{M}), g_{\mathcal{M}}(w_0)) \times (\mathsf{Trf}(U), x_0)$ by $g(w) := g_{\mathcal{M}}(w) \cup \{p_\varphi \in V^+ \mid \mathcal{M}, w \vDash \varphi\}$.

To show C1, fix any agent i and any two worlds w_1 and w_2 of $\mathcal{M} * U$. We have the following chain of equivalences: $R_i^{\mathcal{M}*U} w_1 w_2$ holds iff we have

$$R_i^{\mathcal{M}} w_1 w_2 \text{ and there are } \psi,\ \chi \text{ s.t. } (\psi, i, \chi) \in U \text{ and } \mathcal{M}, w_1 \vDash \psi \text{ and } \mathcal{M}, w_2 \vDash \chi$$

iff

$$\begin{gathered} g_{\mathcal{M}}(w_1) \cup g_{\mathcal{M}}(w_2)' \vDash \Omega_i \text{ and} \\ \{p_\varphi \in V^+ \mid \mathcal{M}, w_1 \vDash \varphi\} \cup \{p'_\varphi \in V^+ \mid \mathcal{M}, w_2 \vDash \varphi\} \vDash \bigvee_{(\psi,i,\chi)\in U} \{p_\psi \wedge p'_\chi\} \end{gathered}$$

iff

$$\begin{array}{rl} & g_{\mathcal{M}}(w_1) \cup \{p_\varphi \in V^+ \mid \mathcal{M}, w_1 \vDash \varphi\} \\ \cup & g_{\mathcal{M}}(w_2)' \cup \{p'_\varphi \in V^+ \mid \mathcal{M}, w_2 \vDash \varphi\} \end{array} \vDash \Omega_i \wedge \bigvee_{(\psi,i,\chi)\in U} \{p_\psi \wedge p'_\chi\}$$

iff

$$\begin{array}{rl} & g_{\mathcal{M}}(w_1) \cup \{p_\varphi \in V^+ \mid \mathcal{M}, w_1 \vDash \varphi\} \\ \cup & g_{\mathcal{M}}(w_2)' \cup \{p'_\varphi \in V^+ \mid \mathcal{M}, w_2 \vDash \varphi\} \end{array} \vDash \Omega_i \wedge \Omega_i^+$$

iff

$$g(w_1) \cup g(w_2)' \vDash \Omega_i \wedge \Omega_i^+$$

For C2, take any world w of $\mathcal{M} * U$ and any $p \in U$ where U is the original vocabulary of $\mathcal{M}$. Arrow updates never modify the valuation, so we immediately have $p \in g(w)$ iff $p \in g_{\mathcal{M}}(W)$ iff $p \in \pi^{\mathcal{M}}(w)$.

For C3, take any $t \subseteq V \cup V^+$ where V is the vocabulary of $\mathcal{F}(\mathcal{M})$, possibly extending that of $\mathcal{M}$. We show only left-to-right, the other direction is similar.

Suppose t is a state of the resulting structure: $t \vDash \theta \wedge \|\theta^+\|_{\mathcal{F}(\mathcal{M})}$. Then $t \cap V \vDash \theta$ and thus $t \cap V$ is a state of $\mathcal{F}(\mathcal{M})$. In particular, there is a world w of $\mathcal{M}$ such that $g_{\mathcal{M}}(w) = t \cap V$. Arrow updates never delete worlds, so w is also a world of $\mathcal{M} * U$.

Spelling out θ^+, we have $t \vDash \| \bigwedge_{p_\varphi \in V^+} (p_\varphi \leftrightarrow \varphi) \|_{\mathcal{F}(\mathcal{M})}$, which means that for all $p_\varphi \in V^+$ we have $p_\varphi \in t$ iff $t \vDash \|\varphi\|_{\mathcal{F}(\mathcal{M})}$. But note that $\|\varphi\|_{\mathcal{F}(\mathcal{M})} \in \mathcal{L}_B(V)$ and recall $g_{\mathcal{M}}(w) = t \cap V$. Therefore $p_\varphi \in t$ iff $g_{\mathcal{M}}(w) \cap V \vDash \|\varphi\|_{\mathcal{F}(\mathcal{M})}$ iff $\mathcal{M}, w \vDash \varphi$.

Hence we have $t \cap V^+ = \{p_\varphi \in V^+ \mid \mathcal{M}, w \vDash \varphi\}$, which is exactly the second part of our definition for g. Together, we have a world w in $\mathcal{M} * U$ such that $t = g(w)$. □

2.9.7. Example. Consider the arrow update $U = \{(p, a, p), (\neg p, a, \neg p), (\top, b, \top)\}$ from Example 1.4.2, with a and b referring to Alice and Bob, respectively. Definition 2.9.5 gives us the following equivalent belief transformer:

$$\left(V^+ = \{p_p, p_{\neg p}, p_\top\},\ \theta^+ = \begin{array}{l} (p_p \leftrightarrow p) \wedge p_\top \\ \wedge (p_{\neg p} \leftrightarrow \neg p) \end{array},\ \begin{array}{l} \Omega_a = (p_p \wedge p'_p) \vee (p_{\neg p} \wedge p'_{\neg p}) \\ \Omega_b = \top \end{array} \right)$$

Here θ^+ implies $p_p \leftrightarrow p_{\neg p}$ and p. Hence we can remove $p_{\neg p}$ and $p_\top$ from V^+ to get the shorter equivalent $(V^+ = \{p_p\},\ \theta^+ = (p_p \leftrightarrow p),\ \Omega_a = (p_p \leftrightarrow p'_p),\ \Omega_b = \top)$ which in turn is equivalent to $(V^+ = \{q\}, \theta^+ = (q \leftrightarrow p), O^+_a = \{q\},\ O^+_b = \varnothing)$ from Example 2.5.6.

The attentive reader will notice that Definition 2.9.5 still encodes an exponential blow-up: If there are n different formulas occurring in the arrows of U, then we also use $|V^+| = n$ new propositional variables to define $\mathsf{Trf}(U)$. This means that the transformer in principle talks about 2^n possible events, just like the action model translation given in [KR11b, Definition 4.6].

We can define a better translation with less propositions by not labeling each formula occurring in the arrows with a new proposition, but with a subset of a large enough set of fresh propositions — similar to the labeling of actions in Definition 2.9.2. This yields an equivalent transformer such that $|V^+| = \lceil \log_2 n \rceil$ where n is the number of different formulas occurring in the original arrow update. The downside of this encoding is that the connection between individual arrows and the observation laws becomes much less intuitive.

Translating arrow updates to transformers also sheds new light on a restriction we made in the definition of transformers and transformations, namely the strict separation between V and V^+. In particular, we demanded that observational laws Ω^+_i are from the boolean language $\mathcal{L}_B(V^+)$.

Suppose we would allow observational laws to come from the language $\mathcal{L}_B(V \cup V^+)$ including the original vocabulary V or even the epistemic language $\mathcal{L}(V \cup V^+)$. For this, we would have to adapt the definition of transformation to first translate observational laws to local boolean equivalents. The result would be a symbolic representation for updates that allows for a direct translation of arrows: for each (ψ, i, χ), add the disjunct $\psi \wedge \chi'$ to Ω_i.

In fact, arrow updates then correspond to those knowledge transformers where $V^+ = \varnothing$, reflecting the fact that they can only refine models and never increase the number of worlds. We leave it as future work to define and study the details of such "symbolic arrow updates", and return to transformers for the next section.

2.10 Symbolic Language and Reduction Axioms

Analogous to the action model language from Definition 1.3.6, we can also add transformers to our language as operators.

2.10.1. DEFINITION. Given a vocabulary V, the symbolic language $\mathcal{L}_S(V)$ of Dynamic Epistemic Logic with dynamic operators for transformers extends $\mathcal{L}(V)$ and is given by

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi \mid C_\Delta\varphi \mid [\mathcal{X}, x]\varphi$$

where $p \in V$, $i \in I$, $\Delta \subseteq I$ and (X, x) is an event as in Definition 2.8.2.

This language with dynamic operators can be interpreted on belief structures.

2.10.2. Definition. Suppose we have a transformer $\mathcal{X} = (V^+, \theta^+, V_-, \theta_-, \Omega^+)$. The corresponding dynamic operator in $\mathcal{L}_S(V)$ is interpreted as follows:

$$(\mathcal{F}, s) \vDash [\mathcal{X}, x]\varphi \quad \text{iff} \quad (\mathcal{F}, s) \vDash [x \sqsubseteq V^+]\theta^+ \text{ implies } (\mathcal{F} \times \mathcal{X}, s^x) \vDash \varphi$$

where s^x is the new actual state as in Definition 2.8.2.

We can see how $[x \sqsubseteq V^+]\theta^+$ takes over the role of $\mathsf{pre}(a)$. These semantics yield the following globally valid reduction axioms, similar to those for action models in Fact 1.3.8.

2.10.3. Fact. The following $\mathcal{L}_S$ formulas called *reduction axioms* are valid.

- $[\mathcal{X}, x]p \leftrightarrow ([x \sqsubseteq V^+]\theta^+ \to ([x \sqsubseteq V^+]\theta_-(p)))$
- $[\mathcal{X}, x]\neg\psi \leftrightarrow ([x \sqsubseteq V^+]\theta^+ \to \neg[\mathcal{X}, x]\psi)$
- $[\mathcal{X}, x](\psi_1 \wedge \psi_2) \leftrightarrow ([\mathcal{X}, x]\psi_1 \wedge [\mathcal{X}, x]\psi_2)$
- $[\mathcal{X}, x]K_i\psi \leftrightarrow ([x \sqsubseteq V^+]\theta^+ \to \bigwedge\{K_i[\mathcal{X}, y]\psi \mid x \cup y' \vDash \Omega^+\})$

Hence for every formula in $\mathcal{L}_S$ without C there is an equivalent formula in $\mathcal{L}$.

Formulas from $\mathcal{L}_S$ can also be evaluated symbolically by translating them to boolean equivalents. In contrast to the reduction axioms, this translation is with respect to a specific belief structure. For a belief transformer $(\mathcal{X}, x)$ without factual change, i.e. where we have $V_- = \varnothing$, we can use the same reduction as for knowledge transformers mentioned on page 56:

$$\|[\mathcal{X}, x]\varphi\|_{\mathcal{F}} := \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \to [x \sqsubseteq V^+]\|\varphi\|_{\mathcal{F}\times\mathcal{X}}$$

For transformers with factual change the boolean translation becomes more complex, because we also need to substitute postconditions for variables and restore the old values. The following definition and theorem give all details.

2.10.4. Definition. Given a belief structure $\mathcal{F}$ we can translate from $\mathcal{L}(V)$ to $\mathcal{L}_B(V)$ as described in Definition 2.6.3. We extend this translation to $\mathcal{L}_S(V)$ with the following case.

$$\|[\mathcal{X}, x]\varphi\|_{\mathcal{F}} := \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \to [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto \theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}}$$

Admittedly, this chain of substitutions deserves some explanation. We can read the consequent of this formula from inside out, i.e. from right to left, as follows.

1. $\|\varphi\|_{\mathcal{F}\times\mathcal{X}}$ is the boolean equivalent of φ with respect to the new structure $\mathcal{F}\times\mathcal{X}$ after the transformation.

2. The operator $[V_- \mapsto \theta_-(V_-)]$ replaces all variables in V_- with their postconditions, slightly abusing notation: It denotes the *simultaneous* substitution of $\theta_-(q)$ for each $q \in V_-$.

   In principle we would have to apply an additional substitution $[V_- \mapsto V_-^\circ]$ to all $\theta_-(q)$, to evaluate postconditions with the old values of changed propositions. But this would be undone by $[V_-^\circ \mapsto V_-]$ in step 4 anyway, so it makes no difference whether we use $\theta_-(q)$ or $[V_- \mapsto V_-^\circ]\theta_-(q)$ here.

   In the proof of Theorem 2.10.5 however, the nested substitution is needed as shown in line (5) below.

3. The next operator $[x \sqsubseteq V^+]$ simulates the actual event x.

4. Finally, $[V_-^\circ \mapsto V_-]$ moves the copies of modified propositions back to the original variables.

2.10.5. THEOREM. *The translation given in Definition 2.10.4 is truthful: for any belief structure $\mathcal{F}$, any state s, any event $(\mathcal{X}, x)$ and any formula φ we have:*

$$(\mathcal{F}, s) \vDash [\mathcal{X}, x]\varphi \iff s \vDash \|[\mathcal{X}, x]\varphi\|_{\mathcal{F}}$$

Proof:
The interesting case is when the precondition holds, so we first assume that $(\mathcal{F}, s) \vDash [x \sqsubseteq V^+]\theta^+$. Then we have the following chain of equivalences.

$$\begin{aligned}
& s^x \vDash \|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (1)\\
\iff\ & (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \cup \{p \in V_- \mid s \cup x \vDash \theta_-(p)\} \vDash \|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (2)\\
\iff\ & (s \setminus V_-) \cup (s \cap V_-)^\circ \cup x \vDash [V_- \mapsto [V_- \mapsto V_-^\circ]\theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (3)\\
\iff\ & (s \setminus V_-) \cup (s \cap V_-)^\circ \vDash [x \sqsubseteq V^+][V_- \mapsto [V_- \mapsto V_-^\circ]\theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (4)\\
\iff\ & s \vDash [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto [V_- \mapsto V_-^\circ]\theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (5)\\
\iff\ & s \vDash [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto \theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} && (6)
\end{aligned}$$

To clarify what is happening here, note that lines (1) and (2) take place in the language $\mathcal{L}_B(V \cup V^+ \cup V_-^\circ)$, line (3) in $\mathcal{L}_B((V \setminus V_-) \cup V^+ \cup V_-^\circ)$, line (4) in $\mathcal{L}_B((V \setminus V_-) \cup V_-^\circ)$, and lines (5) and (6) in $\mathcal{L}_B(V)$.

Together with the semantics for $[\mathcal{X}, x]$, we can now finish the proof:

$$(\mathcal{F}, s) \vDash [\mathcal{X}, x]\varphi$$

$$\iff (\mathcal{F}, s) \vDash [x \sqsubseteq V^+]\theta^+ \text{ impl. } (\mathcal{F} \times \mathcal{X}, s^x) \vDash \varphi \qquad \text{by Definition 2.10.2}$$

$$\iff s \vDash \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \text{ impl. } s^x \vDash \|\varphi\|_{\mathcal{F}\times\mathcal{X}} \qquad \text{by Theorem 2.6.4}$$

$$\iff s \vDash \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \text{ impl. } s \vDash [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto \theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} \qquad \text{by (1)} \iff \text{(6) above}$$

$$\iff s \vDash \|[x \sqsubseteq V^+]\theta^+\|_{\mathcal{F}} \ \rightarrow\ [V_-^\circ \mapsto V_-][x \sqsubseteq V^+][V_- \mapsto \theta_-(V_-)]\|\varphi\|_{\mathcal{F}\times\mathcal{X}} \qquad \text{by Definition 1.0.2}$$

$$\iff s \vDash \|[\mathcal{X}, x]\varphi\|_{\mathcal{F}} \qquad \text{by Definition 2.10.4}$$

□

2.11 Symbolic Bisimulations

For Kripke models the notion of bisimulation characterizes the situation when two models are equivalent — see Definition 1.1.5 and Theorems 1.1.6 and 1.1.8 in the previous chapter. We now investigate how bisimulations can be defined for our symbolic structures.

When are two knowledge or belief structures equivalent? This question comes with a hidden parameter, namely the vocabulary for which we want them to be equivalent. If the structures have disjoint vocabularies, then there are no non-trivial formulas which can be interpreted on both. So we will assume that their vocabularies at least overlap. They do not have to be same, though. For example, the two structures can use different auxiliary variables that encode epistemic relations. Or they might use the same variables, but still only be equivalent with respect to a subset of the vocabulary.

The following definition describes a symbolic equivalent of bisimulations for knowledge structures. We use a boolean formula over a double vocabulary to encode the relation between the two models represented by the structures, similar to how we encoded relations in belief structures.

2.11.1. Definition. Suppose we have two knowledge structures $\mathcal{F}_1 = (V_1, \theta_1, O_1)$ and $\mathcal{F}_2 = (V_2, \theta_2, O_2)$. Consider a subset of their shared vocabulary $V \subseteq V_1 \cap V_2$. To separate any remaining shared vocabulary not in V, let $\overline{V}$ be the *disjoint* union of V, V_1 without V and V_2 without V, i.e. $\overline{V} := V \cup ((V_1 \setminus V) \uplus (V_2 \setminus V))$.

Similar to our notation with primes and $\circ$, for any set X, let X^* denote a fresh copy of all variables in X.

A boolean formula $\beta \in \mathcal{L}_B(\overline{V} \cup \overline{V}^*)$ is called a *symbolic bisimulation* for V between $\mathcal{F}_1$ and $\mathcal{F}_2$ iff for any states s_1 of $\mathcal{F}_1$ and s_2 of $\mathcal{F}_2$ such that $s_1 \cup s_2^* \vDash \beta$, we have:

1. Propositional agreement: For all $p \in V$ we have $s_1 \vDash p$ iff $s_2 \vDash p$.
2. Forth: For any agent i and any state t_1 of $\mathcal{F}_1$ such that $O_1^i \cap s_1 = O_1^i \cap t_1$, there is a state t_2 of $\mathcal{F}_2$ such that $t_1 \cup t_2^* \vDash \beta$ and $O_2^i \cap s_2 = O_2^i \cap t_2$.
3. Back: For any agent i and any state t_2 of $\mathcal{F}_2$ such that $O_2^i \cap s_2 = O_2^i \cap t_2$, there is a state t_1 of $\mathcal{F}_1$ such that $t_1 \cup t_2^* \vDash \beta$ and $O_1^i \cap s_1 = O_1^i \cap t_1$.

Similar to Kripke models, we call two scenes $(\mathcal{F}_1, s_1)$ and $(\mathcal{F}_2, s_2)$ *bisimilar* iff there is a symbolic bisimulation β such that $s_1 \cup s_2^* \vDash \beta$.

The conditions for a symbolic bisimulation encode the usual definition of bisimulation: Connected worlds first need to agree on the atomic propositions, in our case only those in the shared vocabulary. Then we have the "forth" and "back" conditions which say that any reachable state on one side must be connected to a reachable state on the other side.

An interesting feature of symbolic bisimulations is that all three conditions about β can themselves be expressed as boolean formulas and therefore be checked easily. In particular, the following lemma means that we do not have to generate the encoded explicit Kripke models to check a bisimulation between two knowledge structures.

2.11.2. LEMMA. *Suppose we have two knowledge structures $\mathcal{F}_1$ and $\mathcal{F}_2$. The following are equivalent:*

- *β is a symbolic bisimulation for V between $\mathcal{F}_1$ and $\mathcal{F}_2$;*
- *β encodes a bisimulation for the vocabulary V between the two encoded S5 Kripke models $\mathcal{M}(\mathcal{F}_1)$ and $\mathcal{M}(\mathcal{F}_2)$;*
- *the following three boolean formulas are tautologies, i.e. their BDDs and the single BDD of their conjunction are equal to $\top$:*

$$(\theta_1 \land \theta_2^* \land \beta) \to \bigwedge_{p \in V} (p \to p^*)$$

$$(\theta_1 \land \theta_2^* \land \beta) \to \bigwedge_i \left(\forall (V \setminus O_1^i) \left(\theta_1 \to \exists (V \setminus O_2^i)^* \left(\theta_2^* \land \beta'\right)\right)\right)$$

$$(\theta_1 \land \theta_2^* \land \beta) \to \bigwedge_i \left(\forall (V \setminus O_2^i)^* \left(\theta_2^* \to \exists (V \setminus O_1^i) \left(\theta_1 \land \beta'\right)\right)\right)$$

2.11.3. THEOREM. *Symbolic bisimulation implies semantic equivalence: If β is a symbolic bisimulation for V between the knowledge structures $\mathcal{F}_1$ and $\mathcal{F}_2$ such that $s_1 \cup s_2^* \vDash \beta$, then for all $\varphi \in \mathcal{L}(V)$ we have that $(\mathcal{F}_1, s_1) \vDash \varphi$ iff $(\mathcal{F}_2, s_2) \vDash \varphi$.*

Proof:
This follows directly from Lemma 2.11.2 and Theorem 1.1.6. An alternative proof is by induction on φ and proceeds analogous to a proof of Theorem 1.1.6. □

Given this symbolic analogue of Theorem 1.1.6, we are naturally also interested in the other direction: If two knowledge structures satisfy the same formulas, must there be a symbolic bisimulation between them? The answer is yes and we have the following symbolic version of the Hennessy-Milner Theorem 1.1.8.

2.11.4. THEOREM. *Semantic equivalence implies symbolic bisimilarity: Suppose we have two knowledge structures $\mathcal{F}_1$ and $\mathcal{F}_2$ such that for all $\varphi \in \mathcal{L}(V)$ we have that $(\mathcal{F}_1, s_1) \vDash \varphi$ iff $(\mathcal{F}_2, s_2) \vDash \varphi$. Then there is a symbolic bisimulation β for V between $\mathcal{F}_1$ and $\mathcal{F}_2$ such that $s_1 \cup s_2^* \vDash \beta$.*

Proof:
Suppose $(\mathcal{F}_1, s_1)$ is semantically equivalent to $(\mathcal{F}_2, s_2)$. Then we also have that $(\mathcal{M}(\mathcal{F}_1), s_1)$ is semantically equivalent to $(\mathcal{M}(\mathcal{F}_2), s_2)$. By Theorem 1.1.8 there must a bisimulation Z linking s_1 and s_2. Now let $\beta := \bigvee\{t_1 \sqsubseteq V_1 \land (t_2 \sqsubseteq V_2)^* \mid (t_1, t_2) \in Z\}$. Note that this encodes Z in the sense that we have $t_1 \cup t_2^* \vDash \beta$ iff Zt_1t_2. Hence by Lemma 2.11.2 it is also a symbolic bisimulation. □

In this proof we made a detour via Kripke models. Instead, one can also try to imitate the proof for explicit bisimulations: To prove the Hennessy-Milner Theorem it is usually shown that $\equiv$, the relation of satisfying the same formulas, is itself already a bisimulation (see [BRV01, page 69]). This is also true for our symbolic structures, but we have to be precise about the vocabularies: Semantic equivalence for the modal language $\mathcal{L}(V)$ is not always expressible in the boolean language $\mathcal{L}_B(V \cup V^*)$, because the epistemic relations in the structures can be encoded using different propositions outside of V. In particular, the condition $\bigwedge_{p \in V}(p \leftrightarrow p^*)$ in $\mathcal{L}_B(V \cup V^*)$ is not a symbolic bisimulation for V in general and this is why we defined the larger vocabulary $\overline{V}$ in Definition 2.11.1. Given these complications, we are content with the detour in the proof above and leave it as future work to spell out a more direct proof.

We can generalize symbolic bisimulations to *belief* structures, where they also correspond to the standard notion of bisimulation for explicit Kripke models with non-$\mathsf{S5}$ relations. For brevity we only state the definition and omit the direct analogues of Lemma 2.11.2, Theorem 2.11.4 and Theorem 2.11.3.

2.11.5. Definition. Suppose we have two belief structures $\mathcal{F}_1 = (V_1, \theta_1, \Omega_1)$, $\mathcal{F}_2 = (V_2, \theta_2, \Omega_2)$ and let V and $\overline{V}$ be as in Definition 2.11.1.

A boolean formula $\beta \in \mathcal{L}_B(\overline{V} \cup \overline{V}^*)$ is called a *symbolic bisimulation* for V between $\mathcal{F}_1$ and $\mathcal{F}_2$ iff for any states s_1 of $\mathcal{F}_1$ and s_2 of $\mathcal{F}_2$ such that $s_1 \cup s_2^* \vDash \beta$ we have:

1. Propositional agreement: For all $p \in V$ we have $s_1 \vDash p$ iff $s_2 \vDash p$.
2. Forth: For any agent i and any state t_1 of $\mathcal{F}_1$ such that $s_1 \cup t_1' \vDash \Omega_1^i$, there is a state t_2 of $\mathcal{F}_2$ such that $t_1 \cup t_2^* \vDash \beta$ and $s_2 \cup t_2' \vDash \Omega_2^i$.
3. Back: For any agent i and any state t_2 of $\mathcal{F}_2$ such that $s_2 \cup t_2' \vDash \Omega_2^i$, there is a state t_1 of $\mathcal{F}_1$ such that $t_1 \cup t_2^* \vDash \beta$ and $s_1 \cup t_1' \vDash \Omega_1^i$.

Again, this can also be expressed as a boolean formula. However, we now use *four* copies of our vocabulary, corresponding to the four corners of the usual bisimulation diagram.

The "forth" and "back" conditions translate to:

$$(\theta_1 \wedge \theta_2^* \wedge \beta) \to \bigwedge_i \left(\forall V' \left((\theta_1' \wedge \Omega_1^i) \to \exists V'^* \left({\theta_2'}^* \wedge \beta' \wedge (\Omega_2^i)^*\right)\right)\right)$$

$$(\theta_1 \wedge \theta_2^* \wedge \beta) \to \bigwedge_i \left(\forall V'^* \left(({\theta_2'}^* \wedge (\Omega_2^i)^*) \to \exists V' \left((\theta_1' \wedge \beta' \wedge \Omega_1^i)\right)\right)\right)$$

We could also state this condition with only three copies of our vocabulary, as the original variables in V do not occur after the $\exists$ quantifier. Hence we can overwrite V instead of introducing the fourth copy V'^* and adapt $(\Omega_2^i)^*$ accordingly. This observation can also be made about the standard definition of bisimulation for explicit Kripke models: Bisimulation for standard modal logic is usually described with four variables, but it is already expressible in the three-variable fragment of first-order logic.

If we implement symbolic bisimulation checking with BDDs, unused variables do not matter much — they just do not occur in the BDD of the existentially qualified expression. Hence it is just as fine to use four copies of variables above. Moreover, it is more natural and efficient to sort our variables in this way: Both the BDDs Ω_i describing the agents' relations and a BDD encoding a symbolic bisimulation β will first ask for the variables encoding the starting point and after that for those encoding the ending point.

Finally, we mention but do not further investigate another connection between our symbolic bisimulations and first-order logic fragments: k-variable fragments of first-order logic share many desirable properties of modal logic, including polynomial model checking complexity [Var95]. Similar to our definition for basic modal logic above, the bisimulation notion for a k-variable fragment of first order logic could be encoded using k many copies of the vocabulary, then consisting of predicate symbols instead of atomic propositions.

2.12 Redundancy and Optimization

DEL does not have temporal operators and agents never know the past explicitly. Therefore, after dynamic updates with factual change any old valuation that got overwritten becomes irrelevant. The original explicit product update on Kripke models does this "garbage collection" better than our symbolic transformation: In the coin flip Example 1.3.5, the result is a model which no longer contains any information about the old state of the coin. In contrast, the resulting structure in Example 2.8.3, where we modeled the same coin flip as a transformer, still has the old value. But we have no way in the language to refer to it, so this information is indeed garbage. Fortunately, we can often eliminate such left-over propositions outside the original vocabulary V.

2.12.1. EXAMPLE. The result from Example 2.8.3 was this belief structure:

$$(V = \{p, q, p^\circ\},\ \theta = p^\circ \wedge (p \leftrightarrow q),\ \Omega_a = p^\circ \leftrightarrow p^{\circ\prime},\ \Omega_b = (p^\circ \leftrightarrow p^{\circ\prime}) \wedge (q \leftrightarrow q'))$$

This structure is $\equiv_{\{p,q\}}$ equivalent to

$$(V = \{p, q\},\ \theta = p \leftrightarrow q,\ \Omega_a = \top,\ \Omega_b = q \leftrightarrow q')$$

In general, we can always eliminate an old copy — or any other proposition we no longer care about — from a structure if it is determined by the state law.

2.12.2. LEMMA. *Suppose a structure $\mathcal{F}$ uses the vocabulary $V \cup \{p\}$ and $p \notin V$ is determined by the state law, i.e. $\theta \to p$ or $\theta \to \neg p$ is a tautology. Then there is a smaller structure $\mathcal{F}'$ using only the vocabulary V, such that $(\mathcal{F}, s) \equiv_V (\mathcal{F}', s \setminus \{p\})$.*

Proof:
We obtain $\mathcal{F}'$ by removing p from the vocabulary, replacing θ with $\exists p \theta$, and replacing each Ω_i with $\exists p \exists p' \Omega_i$. A symbolic bisimulation between $\mathcal{F}$ and F' is $\beta := \bigwedge \{q \leftrightarrow q^* \mid q \in V\}$. □

Another form of redundancy can occur between the state law and the encoded epistemic relations, be it observational variables or observation laws. A state law already determines which states we consider at all. The epistemic part of our structures however might repeat this information, in the sense that the set of accessible states will always be a subset of the set of all states determined by θ. We consider the following two toy examples to illustrate this.

2.12.3. EXAMPLE. The following knowledge structures are equivalent:

$$(V = \{p, q\},\ \theta = (p \leftrightarrow q),\ O_a = \{p, q\})$$

$$(V = \{p, q\},\ \theta = (p \leftrightarrow q),\ O_a = \{p\})$$

Similarly, these belief structures are equivalent:

$$(V = \{p, q\},\ \theta = (p \leftrightarrow q),\ \Omega_a = (p \rightarrow q) \wedge (p' \leftrightarrow q') \wedge p' \wedge q')$$
$$(V = \{p, q\},\ \theta = (p \leftrightarrow q),\ \Omega_a = p')$$

In these structures the components O_a and Ω_a repeat (part of) the restriction imposed by the state law θ. But we do not have to repeat θ in O_i or Ω_i because the state semantics are restricted to states of $\mathcal{F}$ anyway. Importantly, the boolean translations in Definitions 2.2.6 and 2.6.3 also explicitly repeat θ (and θ') to restrict the set of accessible states. Our observation from Example 2.12.3 can thus be generalized as follows.

2.12.4. LEMMA. *Suppose we have a belief structure $\mathcal{F} = (V, \theta, \Omega)$. Moreover, suppose that for each i we have a formula $\Omega_i^{\equiv}$ such that $(\theta \wedge \theta') \rightarrow (\Omega_i \leftrightarrow \Omega_i^{\equiv})$ is a boolean tautology. Then $\mathcal{F}$ is equivalent to $(V, \theta, \Omega^{\equiv})$.*

In the implementation we can use Lemma 2.12.4 to optimize our structures. And we are in for a treat: Most BDD packages provide a `restrictLaw` function which does exactly the kind of minimization we need here. For a detailed example, see Section 3.7.

We now end this section with a note how the above compares to optimization techniques for explicit methods. On Kripke models a well known and efficient optimization is to use a generated submodel: Given a pointed model $(\mathcal{M}, w)$ we start with the set $\{w\}$ and close it under the relations of all agents, iterating until a fixpoint is reached. The set of worlds can then be restricted to this reachable subset and we obtain a model $\mathcal{M}'$ such that $(\mathcal{M}, w)$ and $(\mathcal{M}', w)$ satisfy the same formulas — a bisimulation is given by the identity on the worlds we kept.

Symbolically, the analogue of a generated submodel would be this: Start with an actual state s and close it under the encoded relations to get a set of reachable states S. Now change the state law from θ to $\theta \wedge \bigvee\{s \sqsubseteq V \mid s \in S\}$, i.e. a conjunction of the original state law and a big disjunction saying that only those reachable states exist. The resulting structure will be equivalent and will satisfy the same formulas. However, this procedure is not necessarily an optimization: Because we are taking a conjunction, the BDD of the new state law can become much larger than before, incorporating all the relations.

In contrast, the optimization enabled by Lemma 2.12.4 above is safe in the sense that BDDs of the structure will not grow, because we only restrict them with the state law, but do not include it into them. We can now see that our optimization method is actually dual to generated submodels: We restrict reachability encoded in the Ω_i, using the set of states given by θ, not vice versa. Going full circle, the explicit analogue of our optimization would be to add, remove or simply ignore epistemic edges outside the set of possible worlds W.

In conclusion, the switch from an explicit to a symbolic representation implies that we have to rethink what sort of redundancy we should avoid and which methods of optimization perform well.

2.13 Other Similarity Types, Beyond Normality

Before we end this chapter and move on to the details of implementing knowledge and belief structures, we consider some theoretical questions on the generality of our approach. Within the field of Modal Logic we only covered a small specific case: all the modalities we studied are unary and normal. While these are the most common modalities, especially in epistemic logic, it is natural to ask whether our methods can be extended to n-ary and non-normal modalities. For the case of n-ary modalities we can give a positive answer.

2.13.1. EXAMPLE. Consider a ternary relation $R \subseteq (W \times W \times W)$ and a symbolic encoding $\theta \in \mathcal{L}(V)$ of W as in Definition 1.8.1. Then we can define a boolean formula in a triple vocabulary $\Omega(R) \in \mathcal{L}(V \cup V' \cup V'')$ by

$$\Omega(R) := \bigvee_{(x,y,z)\in R} (x \sqsubseteq V \land y \sqsubseteq V' \land z \sqsubseteq V'')$$

to get this equivalence:

$$\forall xyz : Rxyz \iff x \cup y' \cup z'' \vDash \Omega(R)$$

For example, the binary modality $\circ$ from [Kur+95] with the standard semantics

$$\mathcal{M}, x \vDash \varphi \circ \psi \iff \exists y \in W \text{ s.t. } \exists z \in W \text{ s.t. } Rxyz \text{ and } \mathcal{M}, y \vDash \varphi \text{ and } \mathcal{M}, z \vDash \psi$$

can then be translated to boolean equivalents:

$$\|\varphi \circ \psi\| := \exists V' \, (\theta' \land \exists V'' \, (\theta'' \land \Omega(R) \land \|\varphi\|' \land \|\psi\|''))$$

If we want to change the quantifiers in the semantics of $\circ$, we can simply make the same changes in the boolean translation to preserve the correspondence.

In general, for n-ary modalities, we can encode their $(n+1)$-ary relation in a boolean formula $\Omega_R \in \mathcal{L}(V^0 \cup V^1 \cup \cdots \cup V^n)$ with $n+1$ copies of the original vocabulary by defining

$$\Omega_R := \bigvee \{(s_0 \sqsubseteq V^0) \land (s_1 \sqsubseteq V^1) \land \cdots \land (s_n \sqsubseteq V^n) \mid (s_0, \ldots, s_n) \in R\}$$

which gives us:

$$Rs_0 \ldots s_n \iff s_0 \cup s_1' \cup \cdots \cup s_n^{\prime\ldots\prime} \vDash \Omega_R$$

Finding symbolic methods for non-normal modal logics though seems hard. Preferences and plausibility orders are often used as an alternative to the (somewhat controversial) KD45 Kripke models. In principle, such orders are still relations and can be implemented using BDDs, as already shown in [GR02], which also covers belief revision. However, it is not clear whether there is a computational advantage over explicit models.

Another widely used non-normal semantics are neighborhood models where a world can reach multiple sets of worlds called neighborhoods. Relations in those models are of type $R \subseteq W \times \mathcal{P}(W)$ or equivalently $R\colon W \mapsto \mathcal{P}(\mathcal{P}(W))$. In recent work they are used to model *evidence* available to an agent and the knowledge based on it [BFP14].

To our knowledge it is an open question how to symbolically represent neighborhood models. Assuming an injective valuation, the challenge is to characterize *sets of sets of worlds* with boolean formulas or functions. If the number of neighborhoods of each world has a finite bound, this could be done with enough copies of the vocabulary, but this method will not scale well. We conjecture that different representations might be useful for different classes of neighborhood models with different closure conditions — similar to how partitions and observational variables provide compact representations for $\mathsf{S5}$ Kripke models.

Chapter 3

Implementing Symbolic DEL with BDDs

> Informally, though, safe languages can be defined as ones that make it impossible to shoot yourself in the foot while programming.
>
> Benjamin C. Pierce: *Types and Programming Languages*

The previous chapter provides a symbolic framework for Dynamic Epistemic Logic (DEL). We now present an implementation of this framework, resulting in *SMCDEL*, a symbolic model checker for DEL based on Binary Decision Diagrams (BDDs). From an outside perspective, SMCDEL mainly solves the following task: Given a scene $(\mathcal{F}, s)$ and a DEL formula φ, decide whether $\mathcal{F}, s \vDash \varphi$ holds. Separate implementations are given for the case where $\mathcal{F}$ is a knowledge or a belief structure.

Besides this main model checking task, we implement many helper functions and other operations on models and structures. This includes functions to convert back and forth between explicit Kripke models and symbolic structures, i.e. implementations of the translations from Definitions 2.4.5 and 2.6.9.

Our model checker is implemented in Haskell and can be used like DEMO-S5, both in the interactive compiler `ghci` and compiled as a library. Additionally we provide a command-line and a web interface for the most common tasks, working with knowledge structures.

We do not explain all parts of the implementation and not include the complete source code in this thesis: It would waste a lot of paper and as we plan to further develop the code in the future any printed version would quickly become outdated. Instead, we only quote some of the main functions here. The complete code with a documentation in literate programming style [Knu84] can be found here:

`https://github.com/jrclogic/SMCDEL`

The simple web interface is available at:

`https://w4eg.de/malvin/illc/smcdelweb/`

This chapter is structured as follows. We first give an overview of existing software for epistemic model checking in Section 3.1. Section 3.2 explains our choice of Haskell and illustrates its advantages with data types for formulas. Section 3.3 shows how we use BDDs to implement knowledge structures. We give two complete examples in Section 3.4 to show what the input and output of SMCDEL looks like. In Section 3.5 we then give a type-safe implementation of BDDs with different vocabularies. We use this in Section 3.6 to implement belief structures with BDDs and show how they can be optimized in Section 3.7. To model symbolic updates including factual change, we implement transformers in Section 3.8. We end the chapter with a list of modules in Section 3.9, automated testing in Section 3.10 and further ideas for development in Section 3.11.

3.1 Existing Epistemic Model Checkers

Most existing software for model checking was made for temporal logics. The first implementation of symbolic model checking was *SMV* from [McM93] which is also described in [CGP99, Section 8.1]. Since then it has been reimplemented as *NuSMV 2* [Cim+02], which also includes methods for bounded model checking using SAT solvers instead of BDDs. NuSMV and its variants are probably the most widely used model checkers to date. However, NuSMV uses plain temporal logics as input languages and does not cover K or other epistemic operators.

One of the first model checkers for knowledge is *MCK* [GM04]. It still uses LTL and CTL as a temporal base, but on top one can choose between different knowledge semantics to interpret K for different kinds of agents: observational, clock or synchronous perfect recall. MCK is written in Haskell, and internally the original MCK uses BDDs to symbolically represent temporal Kripke models. Recent versions also offer bounded semantics via SAT solving. MCK 1.1.0 was released in August 2014. Unfortunately, only earlier versions of MCK were placed under an open source license. As of March 2018, not even binaries are available on the website of the project at `https://cgi.cse.unsw.edu.au/~mck/pmck/`.

Another model checker for temporal logics with knowledge is *MCTK*, first presented in [SSL07]. It is based on NuSMV 2.1 and employs the same translation of K to $\forall(V \setminus O_i)$ as the $\mathsf{S5}$ version of our implementation (see Definition 2.2.6). MCTK is open source and released under the LGPL. The newest version 1.0.2 was released in January 2016 and can be downloaded from `https://sites.google.com/site/cnxyluo/MCTK/`. Another mirror of the project website is `http://kailesu.net/MCTK/`.

A third model checker for epistemic temporal logics is *MCMAS* which was first released in 2006 [LR06] and has since been under heavy development. The most recent presentation and a comparison to MCK and MCTK is in [LQR15]. The latest version 1.3.0 from September 2017 can be downloaded at `http://vas.doc.ic.ac.uk/software/mcmas/`.

All three model checkers we mentioned so far are for temporal logics. For Dynamic Epistemic Logic, the standard implementations are the two explicit model checkers by Jan van Eijck: *DEMO* [Eij07] and the successor *DEMO-S5* [Eij14a] which is optimized for `S5` logics, using partitions instead of list of pairs to represent relations.

DEMO and DEMO-S5 are written in Haskell and have been adapted in various ways, for example to deal with probabilistic belief [Eij13], actions with factual change [Eij11], knowledge of numbers in register models [Gat14] and most recently, public announcement logic with awareness [GT17].

Another explicit model checker for DEL is the *VisualDEL* tool written in Java by Maduka Attamah, introduced in [Att12]. Unfortunately, this tools was initially not released publicly and we were unable to include it in our comparisons and benchmarks. It also has not been used as widely as DEMO [GT17; Dit+12; VR07; Dit+06]. Since July 2017 VisualDEL is freely available under the MIT license at `https://github.com/mdk333/VisualDEL`, so while it was not within the scope of this thesis, we hope that a better comparison can be done in the future.

The big advantage of explicit implementations like DEMO is in their usability. The user can simply work with the same kind of Kripke models as they are used to drawing on paper. Moreover, we can manipulate models at a single possible world and easily visualize them using tools like *graphviz* [Ell+04].

Additionally, DEMO uses the power of Haskell's *type variables* to gain extra flexibility: possible worlds in Kripke models do not have to be mere indices but can be of almost any type `a`, thereby carrying information in their names, eliminating the need for a valuation function. The DEL language is then extended with a construct `Info` of type `a -> Form` and a formula `Info x` is true at world `w` iff `w == x`. For example, the Muddy Children can be represented with worlds of type `[Bool]` and a formula saying that all three are muddy is simply `Info [True,True,True]`.

3.1.1. Example. Consider the Muddy Children example from Section 2.3. Figure 3.1 shows how we can define the Muddy Children Kripke model for DEMO-S5. The function `mudDemoKrpInit` takes parameters n and m and returns the initial situation of n children out of which m are muddy. It makes use of `bTables`, which generates all possible boolean assignments for a set of propositions. Instead of using a valuation function, the states themselves are lists of boolean values that indicate which agents are muddy. The equivalence relations for each agent are then defined as partitions. The output for $n = m = 3$ is shown in Figure 3.2 and a graph of the model can be seen in Figure 3.3.

At the same time, explicit representation is also the biggest disadvantage of tools like DEMO, because it means that models have to be quite small to be manageable and fit in the memory — the well known state explosion problem already mentioned in Section 1.7.

```
mudDemoKrpInit :: Int -> Int -> DEMO_S5.EpistM [Bool]
mudDemoKrpInit n m = (DEMO_S5.Mo states agents [] rels points) where
  states = DEMO_S5.bTables n
  agents = map DEMO_S5.Ag [1..n]
  rels   = [(DEMO_S5.Ag i, [[tab1++[True]++tab2,tab1++[False]++tab2] |
                  tab1 <- DEMO_S5.bTables (i-1),
                  tab2 <- DEMO_S5.bTables (n-i) ]) | i <- [1..n] ]
  points = [replicate (n-m) False ++ replicate m True]
```

Figure 3.1: DEMO-S5 definition and for Muddy Children.

```
λ> mudDemoKrpInit 3 3
Mo  [ [True ,True ,True ], [True ,True ,False] -- 8 possible worlds
    , [True ,False,True ], [True ,False,False] -- of type [Bool]
    , [False,True ,True ], [False,True ,False]
    , [False,False,True ], [False,False,False] ]
    [Ag 1,Ag 2,Ag 3] -- three agents
    [] -- no valuation function
    [ (Ag 1,[ [[True ,True ,True ],[False,True ,True ]] -- relation as
            , [[True ,True ,False],[False,True ,False]] -- partition
            , [[True ,False,True ],[False,False,True ]] -- for agent 1
            , [[True ,False,False],[False,False,False]] ])
    , ... -- similar for agent 2 and 3 (omitted)
    , ... ]
    [[True,True,True]] -- actual world: all three are muddy
```

Figure 3.2: DEMO-S5 output for three muddy children.

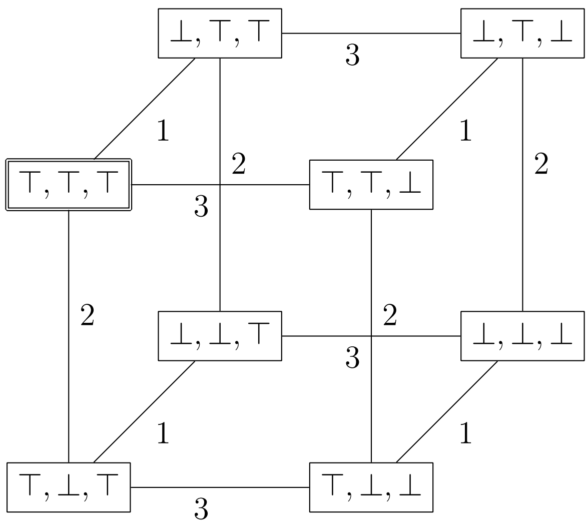


Figure 3.3: Visualized DEMO-S5 model for three muddy children.

With our new implementation *SMCDEL* we combine the best of two worlds: efficient symbolic model checking on one side and intuitive modeling in DEL on the other side. While SMCDEL is not directly based on any of the above model checkers, it uses many ideas from the existing tools. As already mentioned, for the `S5` case we use the same translation for K as in MCTK. Moreover, to make it easy to compare and run benchmarks, we include a full copy of DEMO-S5 in the module `SMCDEL.Explicit.DEMO_S5`.

3.2 From Mathematics to Haskell

Our implementation is written in *Haskell*, a modern purely functional programming language which has several advantages over other languages that make it particularly suitable for our task.

First, Haskell is functional, which fits nicely to mathematical style. For example the list syntax, pattern matching and point-free function composition allow us to write code that resembles the original notation. We especially encourage any reader unfamiliar with Haskell to read the code examples to see how close they are to the original formal mathematical definition.

Second, Haskell is statically typed. This gives us safety guarantees — many mistakes one could easily make in other languages are already noticed at compile time. As an easy example, in our implementation it is impossible to represent a formula that is not well-formed. A more involved usage of the type system is discussed in Section 3.5, where we move the management of different vocabularies to the type level to make sure we do not construct wrong or meaningless BDDs.

Third, Haskell is lazy, i.e. it only evaluates expressions in our program when they are needed. This means that we can work with infinite structures we are used to, such as the list of natural numbers `[0..]` or an infinite supply of atomic propositional variables — as long as we make sure that, once we actually run it, our program will only use a finite part. Laziness can also speed up our program for finite objects: If we only need parts of a model or structure later, the rest does not have to be computed.

A nice cheat sheet for the basic syntax of Haskell is [Bai13]. Proper and systematic introductions to Haskell can be found in the fun and colorful [Lip11], the serious and mature [OSG08], and the opinionated and forthcoming [AM18]. An alternative introduction to Haskell, Mathematics and Logic at the same time is [DE12].

When translating mathematics to Haskell we have to be precise and careful. It often happens that differences which we did not care about when defining something, suddenly become important when we want to implement it. For example, we usually identify a propositional variable $p \in V$ with the same variable used as a formula $p \in \mathcal{L}_B(V)$. To implement our ideas in a typed language such as Haskell, the difference has to be spelled out. In SMCDEL, `P 0` is the atomic proposition and `PrpF (P 0)` is the corresponding formula.

Similarly, all representations of relations (see Section 1.8) are isomorphic and when proving something about a relation R, we do not worry whether it is a subset of $A \times B$, a function $A \to \mathcal{P}(B)$ or a function $A \to B \to \{\mathsf{True}, \mathsf{False}\}$. For Haskell though, lists of pairs `[(a,b)]`, unary functions to lists `a -> [b]` and binary functions to booleans `a -> b -> Bool` are all different types.

Even when there is a way to repeat a mathematical simplification in code, this might not be the best idea for performance reasons. In the definition of boolean languages we only introduce $\wedge$ and $\neg$ as primitive operators and then define $\vee$ and $\to$ as abbreviations. But if we evaluate $p \vee q$ by first spelling it out as $\neg(p \wedge q)$ it will take longer and use more memory than if we make $\vee$ a primitive. While this seems irrelevant for small examples, the effect does matter for more complex boolean functions. Hence in our implementation, all boolean connectives are primitives which get interpreted with their usual semantics.

Figure 3.4 shows the data types for propositional variables and formulas used in SMCDEL. Note the similarity between a recursive BNF, which we use to give mathematical definitions of formal languages, and the definition of a data type in Haskell.

```haskell
newtype Prp = P Int deriving (Eq,Ord,Show)

data Form
  = Top                             -- ^ True Constant
  | Bot                             -- ^ False Constant
  | PrpF Prp                        -- ^ Atomic Proposition
  | Neg Form                        -- ^ Negation
  | Conj [Form]                     -- ^ Conjunction
  | Disj [Form]                     -- ^ Disjunction
  | Xor [Form]                      -- ^ n-ary X-OR
  | Impl Form Form                  -- ^ Implication
  | Equi Form Form                  -- ^ Bi-Implication
  | Forall [Prp] Form               -- ^ Boolean Universal Quantification
  | Exists [Prp] Form               -- ^ Boolean Existential Quantification
  | K Agent Form                    -- ^ Knowing that
  | Ck [Agent] Form                 -- ^ Common knowing that
  | Kw Agent Form                   -- ^ Knowing whether
  | Ckw [Agent] Form                -- ^ Common knowing whether
  | PubAnnounce Form Form           -- ^ Public announcement that
  | PubAnnounceW Form Form          -- ^ Public announcement whether
  | Announce [Agent] Form Form      -- ^ (Semi-)Private announcement that
  | AnnounceW [Agent] Form Form     -- ^ (Semi-)Private announcement whether
  deriving (Eq,Ord,Show)
```

Figure 3.4: Definition of formulas in `SMCDEL.Language`.

Our `Form` type has many more cases, i.e. primitives, than the formal language from Definition 1.3.6, because it is more convenient and efficient to implement them directly and not as abbreviations.

On the other hand, just like DEMO and DEMO-S5, we do not include all dynamic operators into our language as done in Definition 1.3.6. This is to interpret the same language on explicit and symbolic structures: If our language contained action models or transformers, those formulas could only be interpreted via translations, to make sense of something like $\mathcal{M}, w \vDash [\mathcal{X}, x]\varphi$ which strictly speaking is not well-defined — we only interpret $\mathcal{L}_D$ on Kripke models and not $\mathcal{L}_S$. Vice versa, also $\mathcal{F}, s \vDash [\mathcal{A}, a]\varphi$ would not make sense directly. Hence in the `Form` type below we use *labels* for public and (semi-)private announcements, which then get mapped to the appropriate action model or transformer, depending on where they are interpreted. This does not limit the power of our program, because we can still define the result of more complex updates outside the language and then evaluate the remaining formula on the resulting model or structure.

3.3 Knowledge Structures with BDDs

In this section we describe the implementation of knowledge structures, our symbolic equivalent of `S5` Kripke models. In Section 2.3 we showed how epistemic operators get replaced by boolean connectives when a new state law is computed. Syntactically, the state law became more and more complex, but semantically the same boolean function could be represented with a much shorter formula, allowing us to write down an equivalent but more succinct knowledge structure.

In the implementation we go one step further. We never actually need the syntax of a state law θ in a knowledge structure $\mathcal{F} = (V, \theta, O)$. Even though θ is a formula from the boolean language $\mathcal{L}_B(V)$, we only care about the boolean *function* which it represents. This is where Binary Decision Diagrams (BDDs), as introduced in Section 1.9, come in extremely handy. In our code we let the second component of a knowledge structure be of type `Bdd`. The complete data type for knowledge structures in SMCDEL is shown in Figure 3.5.

```
data KnowStruct = KnS [Prp]              -- vocabulary
                      Bdd                -- state law
                      [(Agent,[Prp])]    -- observational variables
                    deriving (Eq,Show)
type State = [Prp]
type Scenario = (KnowStruct,State)
```

Figure 3.5: Data type for knowledge structures.

3.3.1. Example. We consider again the knowledge structure

$$\mathcal{F} := (V = \{p_1, p_2\}, \theta = p_1 \to p_2, O_1 = \{p_1\}, O_2 = \{p_2\})$$

from Example 2.2.2, now with p_1 and p_2 instead of p and q, respectively. Figure 3.6 shows how $\mathcal{F}$ is represented in the implementation, with a BDD for θ.

```
KnS [P 1,P 2] (Var 1 (Var 2 Top Bot) Top) [("1",[P 1]),("2",[P 2])]
```

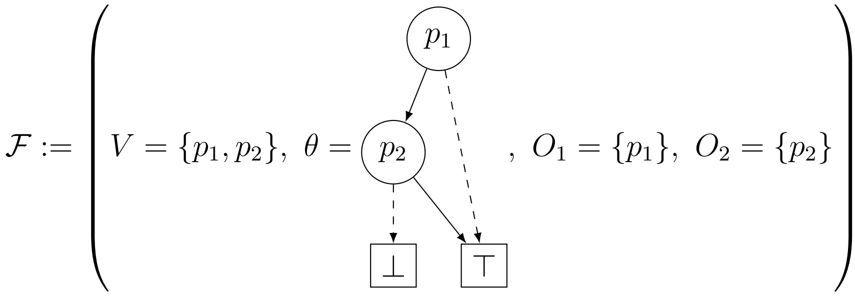


Figure 3.6: A knowledge structure with a BDD for θ.

3.3.2. Example. Consider again the muddy children example from Section 2.3. Figure 3.7 shows the BDDs of the state laws θ_0 to θ_3, reflecting the smaller and smaller set of allowed states after each announcement.

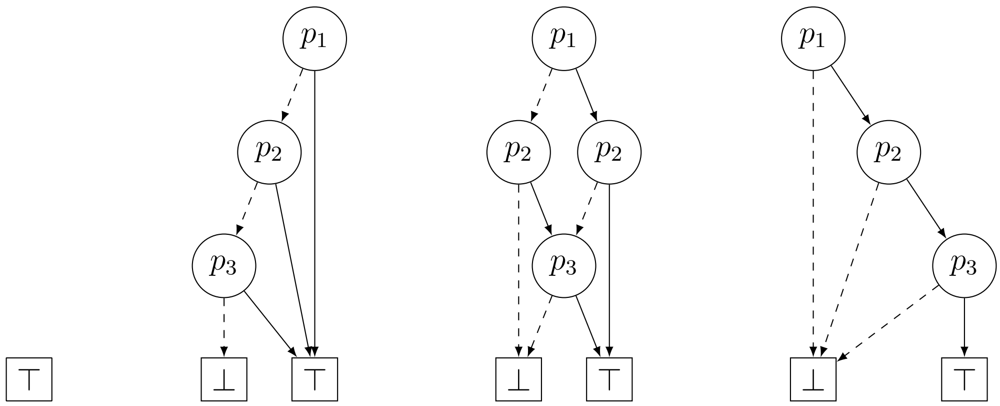


Figure 3.7: Four BDDs representing the Muddy Children state laws.

Figure 3.8 shows the core of SMCDEL: The function `bddOf` implements the DEL-to-boolean translation $\|\cdot\|_{\mathcal{F}}$ given in Definition 2.2.6. Intuitively, the type of this translation would be `KnowStruct -> Form -> Form`, where the output only uses boolean connectives and could be given to a function that computes BDDs. But this type would be inefficient, because we do not need the possibly lengthy formula given by Definition 2.2.6. Hence we skip the intermediate computation and translate a given DEL formula φ directly to the BDD of $\|\varphi\|_{\mathcal{F}}$.

For clarity, here we leave out parts of the language that are primitives in the implementation but abbreviations in the previous chapters. For the full code, see the module `SMCDEL.Symbolic.HasCacBDD` in [Gat18].

For all operations on BDDs we use the CacBDD package [LSX13], via the binding library HasCacBDD [Gat17a] which we developed during this research. Still, our framework and implementation can easily be adapted to use other

```
bddOf :: KnowStruct -> Form -> Bdd
bddOf _   Top            = top
bddOf _   (PrpF (P n))   = var n
bddOf kns (Neg form)     = neg (bddOf kns form)
bddOf kns (Conj forms)   = conSet (map (bddOf kns) forms)
bddOf kns@(KnS allprops lawbdd obs) (K i form) =
  forallSet otherps (imp lawbdd (bddOf kns form)) where
    otherps = map ( \ (P n) -> n) (allprops \\ apply obs i)
bddOf kns@(KnS allprops lawbdd obs) (Ck ags form) =
  gfp lambda where
    otherps i = map ( \ (P n) -> n) (allprops \\ apply obs i)
    lambda z  = conSet (bddOf kns form :
                     [ forallSet (otherps i) (imp lawbdd z) | i <- ags ])
bddOf kns (PubAnnounce form1 form2) =
  imp (bddOf kns form1) (bddOf (pubAnnounce kns form1) form2)
```

Figure 3.8: Implementing the boolean translation from Definition 2.2.6.

BDD packages, as long as they provide the same functions to create and manipulate BDDs that we use on the right side in Figure 3.8: `top`, `var`, `neg`, `conSet`, `forallSet` and so on. For example, SMCDEL already includes the module `SMCDEL.Symbolic.CUDD`. Instead of CacBDD it uses the CUDD library [Som12] which is also the base of many other symbolic model checkers. Because CUDD is not written in Haskell, we use it via bindings from [Wal15].

After implementing the boolean translation, we can write a symbolic evaluation function `evalViaBdd`. To check whether φ holds at state s of $\mathcal{F}$, it first computes the BDD of the equivalent boolean formula $\|\varphi\|_{\mathcal{F}}$ according to Definition 2.2.6. Then it checks the boolean satisfaction $s \vDash \|\varphi\|_{\mathcal{F}}$.

The function `validViaBdd` decides whether a formula φ is valid on $\mathcal{F}$, i.e. true at *all* states. We simply check whether the boolean formula $\theta \to \|\varphi\|_{\mathcal{F}}$ is a tautology, i.e. whether the boolean equivalent of φ is implied by the state law.

Note that both functions do not have to generate the set of all states. In case we are interested in the set of all states of $\mathcal{F}$, we provide the function `whereViaBdd`. It asks the BDD package for all satisfying assignments of the state law θ and then converts assignments of type `Assignment = [(Int, Bool)]` to states of type `State = [Prp]`. All three functions are shown in Figure 3.9.

```
evalViaBdd :: Scenario -> Form -> Bool
evalViaBdd (kns,s) f = evaluateFun (bddOf kns f) (\n -> P n `elem` s)

validViaBdd :: KnowStruct -> Form -> Bool
validViaBdd kns@(KnS _ lawbdd _) f = top == lawbdd `imp` bddOf kns f

whereViaBdd :: KnowStruct -> Form -> [State]
whereViaBdd kns@(KnS props lawbdd _) f =
 map (sort . map (toEnum . fst) . filter snd) $
   allSatsWith (map fromEnum props) $ con lawbdd (bddOf kns f)
```

Figure 3.9: Functions to check truth and validity symbolically via BDDs.

Figure 3.10 shows examples of how the functions `evalViaBdd`, `validViaBdd` and `whereViaBdd` can be used with the knowledge structures from Example 3.3.1.

```
λ> statesOf mykns
[[P 1,P 2],[],[P 2]]
λ> evalViaBdd (mykns,[P 1]) (K "1" (PrpF $ P 1))
True
λ> evalViaBdd (mykns,[P 1]) (K "2" (PrpF $ P 1))
False
λ> validViaBdd mykns (Ck ["1","2"] (PrpF (P 1) ‘Impl‘ PrpF (P 2)))
True
λ> validViaBdd mykns (Ck ["1","2"] (PrpF (P 2) ‘Impl‘ PrpF (P 1)))
False
λ> whereViaBdd mykns (PrpF (P 2) ‘Impl‘ PrpF (P 1))
[[P 1,P 2],[]]
```

Figure 3.10: Usage examples for the `_ViaBdd` functions.

This completes our first symbolic model checker for S5 PAL. In the next section we will give some more input and output examples, and after that we will extend our implementation to belief structures and transformers.

3.4 S5 Input and Output Examples

In Figure 3.11 we show the function `mudScnInit`, which is the symbolic equivalent of `mudDemoKrpInit` shown above in Figure 3.1. It takes the same parameters n and m, but instead of a Kripke model generates a knowledge structure for SMCDEL. The state law is simply $\top$ and each agent observes all but one proposition. Below the function we again list the example output for $n = m = 3$ and include a mathematical description of the structure. We can see that both the specification and the output are much shorter than their Kripke equivalents on page 88.

```
mudScnInit :: Int -> Int -> Scenario
mudScnInit n m = (KnS vocab law obs, actual) where
  vocab  = [P 1 .. P n]
  law    = boolBddOf Top
  obs    = [ (show i, delete (P i) vocab) | i <- [1..n] ]
  actual = [P 1 .. P m]
```

$$\left(\left(V = \{p_1, p_2, p_3\}, \theta_0 = \top, \begin{array}{c} O_1 = \{p_2, p_3\} \\ O_2 = \{p_1, p_3\} \\ O_3 = \{p_1, p_2\} \end{array}\right), \{p_1, p_2, p_3\}\right)$$

Figure 3.11: Muddy Children input and output for SMCDEL.

To further simplify the usage of our model checker, we also provide an interface in which knowledge structures can be specified using a simple text format. In particular no knowledge of Haskell is needed here.

An example input file for the Dining Cryptographers which we will discuss in the next chapter is shown in Figure 3.12. We first describe the vocabulary in the `VARS` section. Then `LAW` contains a boolean formula, the state law. Under `OBS` we list the observational variables for each agent. After this we use `VALID?` and `WHERE?` followed by formulas. The former checks for validity while the latter returns a list of states where the argument is true.

To read such text files, SMCDEL includes a simple parser based on the Haskell parser generator *Happy* [GM17] and lexer *alex* [DM17]. Note that the indentation is just for readability. The parser actually ignores all whitespace and Haskell style comments marked by two dashes and a space. The output can be printed to the command line as text (Figure 3.13) or as ready to use LaTeX code (Figure 3.14).

```
VARS  0,     -- the NSA paid
      1,2,3, -- cryptographer i paid
      4,5,6  -- shared bits/coins

-- exactly one cryptographer or the NSA paid
LAW   AND ( OR (0,1,2,3), ~(0&1), ~(0&2), ~(0&3), ~(1&2), ~(1&3), ~(2&3) )

OBS   alice: 1, 4,5
      bob  : 2, 4,  6
      carol: 3,   5,6

VALID?  (alice,bob,carol) comknow that (OR (0,1,2,3))

WHERE?  alice knows whether 0

VALID?  [?! XOR (1, 4, 5)] -- After everyone announces the
        [?! XOR (2, 4, 6)] -- XOR of whether they paid and
        [?! XOR (3, 5, 6)] -- the coins they see ...
        AND (
          -- if the NSA paid this is common knowledge:
          0 -> (alice,bob,carol) comknow that 0,
          -- if one of the agents paid, the others don't know that:
          1 -> AND (~ bob   knows that 1, ~ carol knows that 1),
          2 -> AND (~ alice knows that 2, ~ carol knows that 2),
          3 -> AND (~ alice knows that 3, ~ bob   knows that 3) )
```

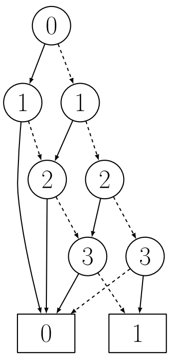
Figure 3.12: Three Dining Cryptographers in SMCDEL.

```
Is Ck ["alice",...] (Disj [PrpF (P 0),...]) valid on the given structure?
True

Is Ck ["alice","bob","carol"] (Disj [...]) valid on the given structure?
True

At which states is Kw "alice" (PrpF (P 0)) true?
[1],[1,6],[1,5],[1,5,6],[1,4],[1,4,6],[1,4,5],[1,4,5,6]

Is PubAnnounceW (...) ... (Conj [...]) valid on the given structure?
True
```

Figure 3.13: Output of SMCDEL on the command line (shortened).

Given Knowledge Structure

$$\mathcal{F} = \left(\{p, p_1, p_2, p_3, p_4, p_5, p_6\}, \quad \text{[BDD]} \quad , \begin{array}{c} \{p_1, p_4, p_5\} \\ \{p_2, p_4, p_6\} \\ \{p_3, p_5, p_6\} \end{array} \right)$$



Results

- Is $Ck_{\{alice,bob,carol\}} \bigvee \{p, p_1, p_2, p_3\}$ valid on $\mathcal{F}$?
  True.
- At which states is $K^?_{\text{alice}} p$ true?
  $\{p_1\}, \{p_1, p_6\}, \{p_1, p_5\}, \{p_1, p_5, p_6\}, \{p_1, p_4\}, \{p_1, p_4, p_6\}, \{p_1, p_4, p_5\}, \{p_1, p_4, p_5, p_6\}$
- Is $[?! \bigoplus \{p_1, p_4, p_5\}][?! \bigoplus \{p_2, p_4, p_6\}][?! \bigoplus \{p_3, p_5, p_6\}] \bigwedge \{(p \to Ck_{\{\text{a,b,c}\}} p), (p_1 \to (\neg K_{\text{b}} p_1 \wedge \neg K_{\text{c}} p_1)), (p_2 \to (\neg K_{\text{b}} p_1 \wedge \neg K_{\text{c}} p_3)), (p_3 \to (\neg K_{\text{b}} p_1 \wedge \neg K_{\text{c}} p_2))\}$ valid on $\mathcal{F}$?
  True.

Figure 3.14: Output of SMCDEL in LaTeX.

3.5 Type-Safe Vocabulary Management

Before we generalize our implementation from knowledge to belief structures, we need to find a good way to manage fresh vocabularies like V'. In mathematical notation, to get fresh variables we can just write p' instead of p, p_2' instead of p_2 and so on. In our implementation some more work is needed to manage copies of propositional variables. If variables are represented by integers and we need fresh propositions or copies, then we need to be careful not to create overlap: Say p was represented by the integer 1, so we might want to use 2 for p', but then we can no longer use it for p_2 etc. Finding a good mapping for fresh variables is similar to solving the famous problem of "Hilbert's Hotel" [Hil13, p. 730].

Moreover, for our BDDs we also have to choose a variable ordering in the double vocabulary. The two obvious candidates are to interleave original and primed variables or to stack all primed variables above or below all unprimed ones. We choose the interleaving order because it has two advantages: First, relations in epistemic models are often already decided by a difference in one specific propositional variable. Hence p and p' should be close to each other to keep the BDD small. Second, we can write general functions to go back and forth between the vocabularies, independent of how many variables we actually use.

Table 3.1 shows the first few variables in $V \cup V'$ and how they are represented in SMCDEL. To switch between the normal and the double vocabulary, we use the functions `mv`, `cp` and their inverses listed in Figure 3.15 and visualized in Figure 3.16.

We now want to lift `mv` and `cp` from single variables to BDDs. It is tempting to use the same `Bdd` type for observational BDDs as for laws. But θ and Ω_i need to use different variable mappings. For example, the BDD of p_1 in the standard vocabulary V uses the integer 1, but in the vocabulary of $V \cup V'$ proposition p_1 is mapped to the integer 2 while p_1' is mapped to 3. Given these two different mappings, taking a conjunction of the BDD of p_1 in V and the BDD of p_2 in $V \cup V'$ makes no sense. We first need to translate the first BDD to the vocabulary of the other.

Variable	Single vocabulary	Double vocabulary
p	`P 0`	`P 0`
p'		`P 1`
p_1	`P 1`	`P 2`
p_1'		`P 3`
p_2	`P 2`	`P 4`
p_2'		`P 5`
$\vdots$	$\vdots$	$\vdots$

Table 3.1: Implementation of single and double vocabulary.

```
mvP, cpP :: Prp -> Prp
mvP (P n) = P  (2*n)      -- represent p  in the double vocabulary
cpP (P n) = P ((2*n) + 1) -- represent p' in the double vocabulary

mv, cp :: [Prp] -> [Prp]
mv = map mvP
cp = map cpP

unmv, uncp :: [Prp] -> [Prp]
unmv = map f where -- Go from p in double vocabulary to p in single
    vocabulary:
  f (P m) | odd m     = error "unmv failed: Number is odd!"
          | otherwise = P $ m `div` 2
uncp = map f where -- Go from p' in double vocabulary to p in single
    vocabulary:
  f (P m) | even m    = error "uncp failed: Number is even!"
          | otherwise = P $ (m-1) `div` 2
```

Figure 3.15: Helper functions `mv`, `cp`, `unmv` and `uncp`.

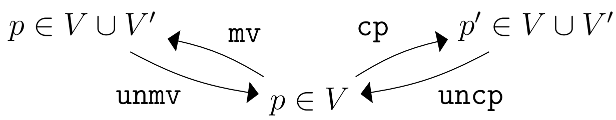


Figure 3.16: Visualization of `mv`, `cp`, `unmv` and `uncp`.

If `RelBDD` and `Bdd` were synonyms — as was actually the case in previous versions of SMCDEL — then it would be up to us users to make sure that BDDs for different vocabularies are not combined. As long as the types match, Haskell would happily generate the chaotic meaningless conjunction.

The reader who got lost between all the fresh variables from V', V° and V^* in the previous chapter might fear that we now create even more confusion by translating our theory to Haskell. But our implementation goal is exactly the opposite: The worry that we forget a prime or a star somewhere should be outsourced to the Haskell compiler.

To catch these problems at compile time we introduce a separate type for BDDs in the double vocabulary: `RelBDD`. In principle `RelBDD` could be a `newtype` of `Bdd`, as we want them to be isomorphic, but there are two problems with `newtype`.

First, we want to separate different BDDs but also have a convenient way of applying the standard BDD functions without converting back and forth. The natural way to do this in Haskell is to use applicative functors and monads, for which we would have to write the appropriate instances.

Second, looking ahead a bit, we will need even more different vocabularies for factual change and symbolic bisimulations — recall the fresh sets V° and V^* from Definitions 2.8.2 and 2.11.1, respectively. Ideally, our design choice now should already solve or at least anticipate these additions.

Combining both problems, it would be tedious to repeat essentially the same instances of Functor, Applicative and Monad each time we add a new vocabulary.

The good news is that the *tagged* library [Kme16] solves both problems and minimizes the code we have to write ourselves.

```
import Data.Tagged

data Dubbel
type RelBDD = Tagged Dubbel Bdd
```

Now suppose we have a BDD representing a formula in the single vocabulary. The following function relabels the BDD to represent the formula with primed propositions in the double vocabulary. It also changes the type to reflect this change.

```
cpBdd :: Bdd -> RelBDD
cpBdd b = pure $ case maxVarOf b of
  Nothing -> b
  Just m  -> relabel [ (n, (2*n) + 1) | n <- [0..m] ] b
```

And similarly, mapping to the unprimed variables in the double vocabulary:

```
mvBdd :: Bdd -> RelBDD
mvBdd b = pure $ case maxVarOf b of
  Nothing -> b
  Just m  -> relabel [ (n, 2*n) | n <- [0..m] ] b
```

Note that `Dubbel` is an empty type, isomorphic to `()`. We only use it as a tag (also called label) on the type level, not to store actual data. Thanks to `Data.Tagged`, our `Tagged Dubbel` automatically becomes an applicative functor. Hence we can lift all Bdd functions to RelBDD using standard notation. For example, the BDDs of $\top$ and $\bot$ in the double vocabulary, which represent the total and empty relation respectively, can also be defined using the generic `pure` instead of `mvBdd` or `cpBdd`.

```
totalRelBdd, emptyRelBdd :: RelBDD
totalRelBdd = pure $ boolBddOf Top
emptyRelBdd = pure $ boolBddOf Bot
```

For another example, the BDD of the conjunction $p_1' \land p_2'$ is now given by

```
λ> con <$> (cpBdd $ var 1) <*> (cpBdd $ var 2)
Tagged Var 3 (Var 5 Top Bot) Bot
```

On the other hand, the aforementioned "wrong" conjunction of p_1 in V and p_2 in $V \cup V'$ would now be represented as follows and no longer has a valid type, just like we wanted:

```
λ> con <$> (var 1) <*> (cpBdd $ var 3)
error:  Couldn't match expected type 'Tagged Dubbel Bdd'
                    with actual type 'Bdd'
```

This will prevent us from accidentally mixing up BDDs in different vocabularies.

3.6 Belief Structures with BDDs

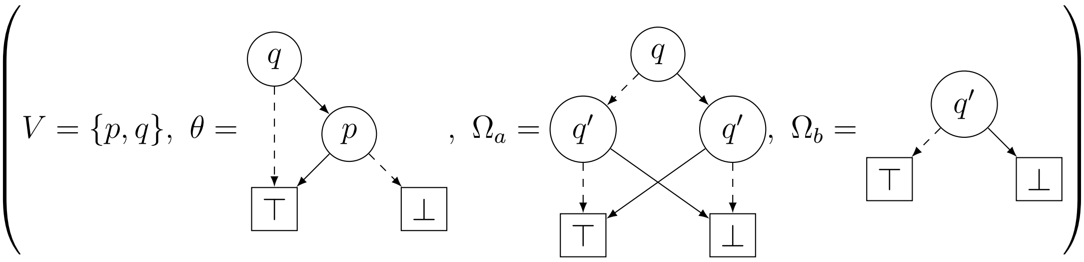
Given our preparation of the `RelBdd` type in the previous section, we can now present the data type for belief structures. To increase efficiency and ensure laziness we use `Map Agent RelBDD` instead of the isomorphic `[(Agent,RelBDD)]`.

```
data BelStruct = BlS [Prp]                  -- vocabulary
                     Bdd                    -- state law
                     (Map Agent RelBDD)     -- observation laws
                   deriving (Eq,Show)

type BelScene = (BelStruct,State)
```

3.6.1. Example. Consider the following belief structure from Example 2.6.6:

$$(V = \{p, q\}, \theta = (q \rightarrow p), \Omega_a = (q \leftrightarrow q'), \Omega_b = \neg q')$$

In SMCDEL the three boolean formulas get replaced with BDDs:



To evaluate formulas on belief structures symbolically, we again implement the boolean translation, now from Definition 2.6.3. In Figure 3.17 we show the cases for `K` and `Ck`. The other connectives are implemented exactly the same way as for knowledge structures. Also the definitions of `evalViaBdd` etc. are exactly the same as those shown in Figure 3.9 above, so we do not repeat them here.

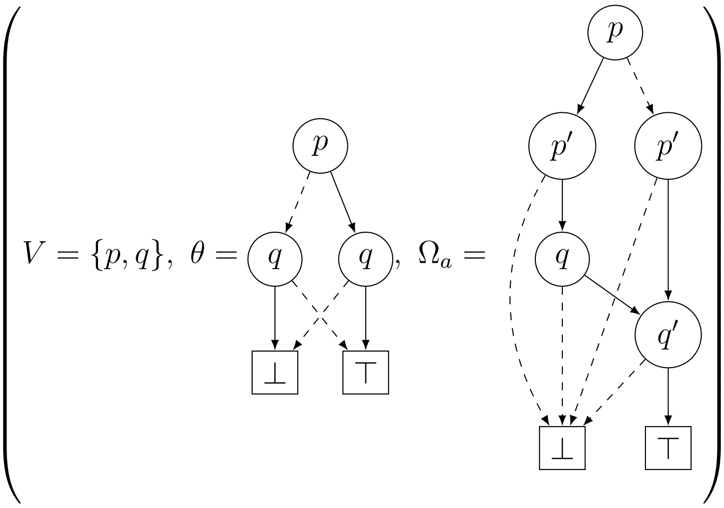
```
bddOf kns@(BlS allprops lawbdd obdds) (K i form) = unmvBdd result where
  result = forallSet ps' <$> (imp <$> cpBdd lawbdd
                                  <*> (imp <$> omegai
                                           <*> cpBdd (bddOf kns form) ) )
  ps'    = map fromEnum $ cp allprops
  omegai = obdds ! i

bddOf kns@(BlS voc lawbdd obdds) (Ck ags form) = lfp lambda top where
  ps' = map fromEnum $ cp voc
  lambda z = unmvBdd $ forallSet ps' <$>
                       (imp <$> cpBdd lawbdd
                            <*> (imp <$> (disSet <$> sequence [ obdds ! i
                                                              | i <- ags ])
                                     <*> cpBdd (con (bddOf kns form) z) ) )
```

Figure 3.17: Boolean translation on belief structures.

3.7 Reduction and Optimization

In Section 2.12 we discussed ways in which our structures can be redundant and provided methods to optimize them. We now describe and illustrate how this optimization works concretely on BDDs.

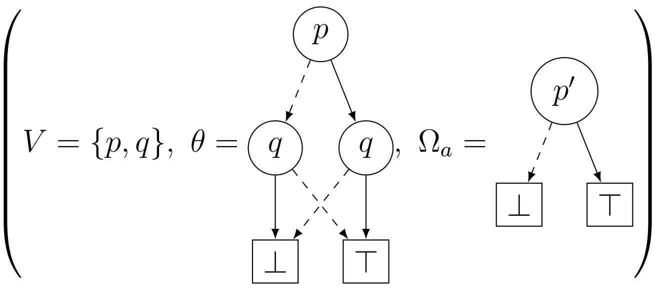
3.7.1. Example. The belief structure ($V = \{p, q\},\ \theta = (p \leftrightarrow q),\ \Omega_a = (p \rightarrow q) \wedge (p' \leftrightarrow q') \wedge p' \wedge q'$) from Example 2.12.3 with BDDs for the state law θ and the observation law Ω_a looks as follows:



To remove the redundancy in Ω_a we can use `restrictLaw` from the HasCacBDD library as follows. Note that `0` is p, `1` is p', `2` is q and `3` is q'.

```
λ> let theta = (var 0 ‘equ‘ var 2)
λ> let theta’ = (var 1 ‘equ‘ var 3)
λ> let orig = conSet [var 0 ‘imp‘ var 2, var 1 ‘equ‘ var 3, var 1, var 3]
λ> orig ‘restrictLaw‘ (theta ‘con‘ theta’)
Var 1 Top Bot
```

Hence $\mathcal{F}$ is equivalent to ($V = \{p, q\},\ \theta = p \leftrightarrow q,\ \Omega_a = p'$), with these BDDs:



3.8 Transformers

We now describe our implementation of transformers, the general symbolic update introduced in Section 2.8. The data type that we use to represent a transformer $\mathcal{X} = (V^+, \theta^+, V_-, \theta_-, \Omega^+)$ is shown with explanatory comments in Figure 3.18.

```
data Transformer = Trf [Prp]                -- addprops, added vocabulary
                       Form                 -- addlaw, event law
                       [Prp]                -- changeprops, modified subset
                       (M.Map Prp Bdd)      -- changelaw, encoded
                           postconditions
                       (M.Map Agent RelBDD) -- eventObs, observation laws
                     deriving (Eq,Show)

type Event = (Transformer,State)
```

Figure 3.18: Types for transformers and events.

Figure 3.19 shows the source code of the `transform` function which applies a transformer to a belief structure. More precisely, it works on pointed structures, so the inputs are a scene $(\mathcal{F}, s)$, where $\mathcal{F}$ is a belief structure, and an event $(\mathcal{X}, x)$. The implementation essentially consists of three parts.

First, we shift the variables in V^+ (`addprops`) to ensure that they are indeed disjoint from the original vocabulary V. By doing this as part of the `transform` function we do not have to manually change transformers if we want to apply them to different knowledge structures with different vocabulary. We also need to shift the variables in V_- occurring in θ, and those in $V_- \cup V'_-$ occurring in Ω. The mapping is defined in the variables `shiftrel` and `shiftrelMVCP`, respectively.

Second, we make copies of the propositions which are to be modified. Similar to the variable shifting, the copying has to be done from V to V° (in `copyrel`) and from $V \cup V'$ to $(V \cup V')^\circ$ (in `copyrelMVCP`).

Third and last, we compute the new vocabulary, the new state law, the new observation laws and the new actual state as in Definition 2.8.2.

3.8.1. Example. A transformer to publicly change p to $\bot$ is shown in the first part of Figure 3.20. This is an instance of the more general Example 2.8.4. The second part implements a transformer which flips the truth value of a given proposition, but only lets a given list of agents observe it.

```
transform :: BelScene -> Event -> BelScene
transform (kns@(BlS props law obdds),s) (Trf addprops addlaw changeprops
    changelaw eventObs, eventFacts) =
  (BlS newprops newlaw newobs, news) where
    -- PART 1: SHIFTING addprops to ensure props and newprops are disjoint
    shiftaddprops = [(freshp props)..]
    shiftrel = sort $ zip addprops shiftaddprops
    relabelWith r = relabel (sort $ map (over both fromEnum) r)
    -- apply the shifting to addlaw and changelaw:
    addlawShifted = replPsInF shiftrel addlaw
    changelawShifted = M.map (relabelWith shiftrel) changelaw
    -- to apply the shifting to eventObs we need shiftrel for the double
        vocabulary:
    shiftrelMVCP = sort $ zip (mv addprops) (mv shiftaddprops)
                   ++ zip (cp addprops) (cp shiftaddprops)
    eventObsShifted = M.map (fmap $ relabelWith shiftrelMVCP) eventObs
    -- the actual event:
    x = map (apply shiftrel) eventFacts
    -- PART 2: COPYING the modified propositions
    copychangeprops = [(freshp $ props ++ map snd shiftrel)..]
    copyrel = zip changeprops copychangeprops
    copyrelMVCP = sort $ zip (mv changeprops) (mv copychangeprops)
    -- PART 3: actual transformation
    newprops = sort $ props ++ map snd shiftrel ++ map snd copyrel
    newlaw = conSet $ relabelWith copyrel (con law (bddOf kns addlawShifted))
                : [var (fromEnum q) `equ` relabelWith copyrel (
                    changelawShifted ! q) | q <- changeprops]
    newobs = M.mapWithKey (\i oldobs -> con <$> (relabelWith copyrelMVCP <$>
        oldobs) <*> (eventObsShifted ! i)) obdds
    news | bddEval (s ++ x) (con law (bddOf kns addlawShifted)) = sort $
        concat
            [ s \\ changeprops
            , map (apply copyrel) $ s `intersect` changeprops
            , x
            , filter (\ p -> bddEval (s ++ x) (changelawShifted ! p))
                changeprops ]
         | otherwise = error "Transformer is not applicable!"
```

Figure 3.19: The implementation of $(\mathcal{F}, s) \times (\mathcal{X}, x)$.

```
publicMakeFalse :: [Agent] -> Prp -> Event
publicMakeFalse agents p = (Trf [] Top [p] changelaw eventobs, []) where
  changelaw = fromList [ (p,boolBddOf Bot) ]
  eventobs  = fromList [ (i,totalRelBdd) | i <- agents ]

flipOverAndShowTo :: [Agent] -> Prp -> Agent -> Event
flipOverAndShowTo everyone p i = (Trf [q] eventlaw [p] changelaw eventobs, [q
    ]) where
  q         = freshp [p]
  eventlaw  = PrpF q `Equi` PrpF p
  changelaw = fromList [ (p, boolBddOf . Neg . PrpF $ p) ]
  eventobs  = fromList $ (i, allsamebdd [q])
                : [ (j,totalRelBdd) | j <- everyone \\ [i] ]
```

Figure 3.20: Two transformers in Haskell code.

3.9 Module Overview

The following is an alphabetical list of the most important modules of SMCDEL and a short summary of their content, as of SMCDEL version 1.0.0.

- `Examples` and submodules: Examples, partially discussed in Chapter 4.
- `Explicit.DEMO_S5`: A full copy of DEMO-S5 [Eij14a] for convenience.
- `Explicit.K`: Explicit model checking with general Kripke models.
- `Explicit.K.Change`: General action models with factual change.
- `Explicit.S5`: Explicit model checking with S5 Kripke models.
- `Internal.Help`: Helper functions, mainly for lists, sets and relations.
- `Internal.Lex`: Lexer for simple input files, made by *alex* [DM17].
- `Internal.MyHaskCUDD`: Wrapper functions for CUDD.
- `Internal.Parse`: Parser for simple input files, made by *happy* [GM17].
- `Internal.TexDisplay`: Type classes for LaTeX and *graphviz* [Ell+04].
- `Internal.Token`: Parsing and Lexing tokens for *alex* and *happy*.
- `Language`: Types defining the DEL language, functions to simplify and print formulas, methods to generate LaTeX code.
- `Other.BDD2Form`: Translation of BDDs back to boolean formulas.
- `Other.MCTRIANGLE`: Implementation of [GS11], benchmarked in Section 4.1.
- `Symbolic.K`: Belief structures from Section 2.6, discussed in Section 3.5.
- `Symbolic.K.Change`: Transformers with factual change from Section 2.8.
- `Symbolic.S5`: Knowledge structures, boolean translation and symbolic evaluation, discussed in Section 3.3.
- `Symbolic.S5_CUDD`: Same as `Symbolic.S5`, with CUDD replacing CacBDD.
- `Symbolic.S5.Change`: Knowledge transformers with factual change, implementing Definition 2.8.5
- `Translations.K` and `Translations.K.Change`: Translations for the general case. Implementing Definitions 2.6.8, 2.6.9, 2.9.1 and 2.9.2.
- `Translations.S5`: Translations between S5 Kripke models and knowledge structures. Implementing Definitions 2.4.2, 2.4.5, 2.5.4 and 2.5.5.

3.10 Automated Testing

It is good practice in modern software engineering to test implementations against specifications: Write down the expected behavior of our program and then check whether it actually does what you want. Especially for a model checker which itself is meant to check specifications, we want to be sure that the implementation is correct.

Some classes of mistakes can be excluded via type safety and we can sometimes prove statements about Haskell code, but it is also very helpful during the development to do randomized property-based testing. We use the famous *QuickCheck* library [CH00] to test the main parts of our implementation.

For example, Figure 3.21 shows an instance of QuickCheck's `Arbitrary` type class for `S5` Kripke models. This tells QuickCheck how to randomly generate Kripke models with five agents and up to nine worlds, using random assignments and random partitions. Moreover, we provide a `shrink` function which QuickCheck uses to find smaller counterexamples that are easier to read and understand.

```
instance Arbitrary KripkeModelS5 where
  arbitrary = do
    let agents = map show [1..(5::Int)]
    let props = map P [0..4]
    worlds <- sort . nub <$> listOf1 (elements [0..8])
    val <- mapM (\w -> do
      randomAssignment <- zip props <$> infiniteListOf (choose (True,False))
      return (w,randomAssignment)
      ) worlds
    parts <- mapM (\i -> do
      randomPartition <- randomPartFor worlds
      return (i,randomPartition)
      ) agents
    return $ KrMS5 worlds parts val
  shrink m@(KrMS5 worlds _ _) =
    [ m `withoutWorld` w | w <- worlds, length worlds > 1 ]
```

Figure 3.21: Generating random S5 Kripke models with QuickCheck.

To illustrate the usage of QuickCheck, consider the conjecture "All `S5` Kripke models get translated to a knowledge structure with the same vocabulary as the original Kripke model". This can be easily falsified using `quickCheck`:

```
λ> quickCheck (\m -> vocabOf (kripkeToKns (m, head (worldsOf m))) === vocabOf m)
*** Failed! Falsifiable (after 5 tests and 2 shrinks):
KrMS5 [5,7]
      [("1",[[5],[7]]),("2",[[7],[5]]),("3",[[7],[5]])
      ,("4",[[7],[5]]),("5",[[5],[7]])]
      [(5,[(P 0,False),(P 1,True),(P 2,False),(P 3,False),(P 4,False)])
      ,(7,[(P 0,True),(P 1,True),(P 2,False),(P 3,True),(P 4,False)])]
[P 0,P 1,P 2,P 3,P 4,P 5,P 6,P 7,P 8,P 9] /= [P 0,P 1,P 2,P 3,P 4]
```

The opposite does not hold either, i.e. there are Kripke models which do get translated to knowledge structures with the same vocabulary as the original Kripke model, for example those models consisting of a single possible world:

```
λ> quickCheck (\m -> vocabOf (kripkeToKns (m, head (worldsOf m))) /= vocabOf m)
*** Failed! Falsifiable (after 1 test):
KrMS5 [1]
      [("1",[[1]]),("2",[[1]]),("3",[[1]]),("4",[[1]]),("5",[[1]])]
      [(1,[(P 0,True),(P 1,False),(P 2,True),(P 3,True),(P 4,False)])]
```

To further simplify the specification of tests and instead of writing our own wrappers around QuickCheck, we use the *Hspec* library [Hen17] which allows us to list properties and examples to be tested in a natural way. Figure 3.22 shows part of a test module containing checks of both randomized (`prop`) and fixed (`it`) examples.

During the development of SMCDEL we use the continuous integration service *travis* to automatically run tests after every commit in our public git repository. The results can be found at `https://travis-ci.org/jrclogic/SMCDEL`.

```
main :: IO ()
main = hspec $ do
describe "SMCDEL.Language" $ do
  prop "simplifying a boolean formula yields something equivalent" $
    \(BF bf) -> boolBddOf bf == boolBddOf (simplify bf)
  prop "simplifying a boolean formula only removes propositions" $
    \(BF bf) -> all (`elem` propsInForm bf) (propsInForm (simplify bf))
describe "SMCDEL.Symbolic.HasCacBDD" $
  prop "boolEvalViaBdd agrees on simplified formulas" $
    \(BF bf) props -> let truths = nub props in
      boolEvalViaBdd truths bf == boolEvalViaBdd truths (simplify bf)
describe "SMCDEL.Explicit.S5" $
  prop "generatedSubmodel preserves truth" $
    \m f -> Exp.eval (m, head $ Exp.worldsOf m) f == Exp.eval (Exp.
        generatedSubmodel (m, head $ Exp.worldsOf m)) f
describe "SMCDEL.Examples" $ do
  it "Three Muddy Children" $
    evalViaBdd mudScn0 (nobodyknows 3) &&
    evalViaBdd mudScn1 (nobodyknows 3) &&
    evalViaBdd mudScn2 (Conj [knows i | i <- [1..3]]) &&
    length (SMCDEL.Symbolic.HasCacBDD.statesOf mudKns2) == 1
  it "Thirsty Logicians: valid for up to 10 agents" $
    all thirstyCheck [3..10]
  it "Dining Crypto: valid for up to 9 agents" $
    dcValid && all genDcValid [3..9]
  it "Russian Cards: 102 solutions" $
    length (filter checkSet allHandLists) == 102
  it "Sum and Product: There is exactly one solution." $
    length sapSolutions == 1
  it "Sum and Product: (4,13) is a solution." $
    validViaBdd sapKnStruct (Impl (Conj [xIs 4, yIs 13]) sapProtocol)
  it "Sum and Product: (4,13) is the only solution." $
    validViaBdd sapKnStruct (Impl sapProtocol (Conj [xIs 4, yIs 13]))
```

Figure 3.22: Automated testing with QuickCheck and HSpec.

3.11 Further Development

Our model checker SMCDEL provides both symbolic model checking methods for DEL, based on BDDs. It is thus a symbolic alternative to DEMO and DEMO-S5. In the next chapter we go through more examples to further illustrate the usage of SMCDEL and to benchmark its performance in comparison with other model checkers.

An alternative approach to symbolic model checking is so-called *bounded* model checking which uses satisfiability (SAT) solvers instead of BDDs — see [Cla+01] for an introduction. Bounded model checking has been successful for temporal logics, and it would be interesting to see if the boolean reasoning needed in SMCDEL could also be reduced to SAT solving in a similar way and what the performance would be.

For the future, we also hope to make SMCDEL more accessible by not only exposing the simple `S5` functions via the command line and web interface, but also the general methods for transformers. One of the challenges here is how dynamic languages like $\mathcal{L}_D$ and $\mathcal{L}_S$ can be exposed to the user without making the `Form` type too general, as discussed in Section 3.2. Embedding a domain-specific language in Haskell such that it can easily be extended in different ways is a tricky problem. An interesting solution which might also be used for SMCDEL in the future are "Typed Tagless Final Interpreters" from [CKS09]. An example how to embed propositional and basic modal logic into Haskell in this way can be found at `https://github.com/m4lvin/logic/`.

Finally, during the development of SMCDEL other abstraction ideas appeared in the DEL literature and should be implemented to compare their performance to our approach. The authors of [CS17], for example, use mental programs [CS15] to give a succinct representation of Kripke and action models.

Chapter 4

Examples and Benchmarks

Edurne: I need to tell you something.
Eduard: I don't want to know it.
Edurne: But I want you to know.
Eduard: I already know it.

Oscar van den Boogaard: *&Me* (2013)

In this chapter, we look at several concrete examples of epistemic modeling and the corresponding model checking tasks that can be solved by SMCDEL. We start with classic logic puzzles from the literature on epistemic logic, but also cover security protocols like the Dining Cryptographers.

We consider it a core feature of SMCDEL to be free software and thoroughly documented. In particular, all results mentioned in this chapter can easily be reproduced with the Haskell tool `stack` [Com18].

All experiments and benchmarks described in this chapter were done using 64-bit Debian GNU/Linux 9 with kernel 4.9.65–3, GHC 8.2.2 and g++ 6.3.0 on an Intel Core i3–2120 3.30 GHz processor and 12 GB of memory.

4.1 Muddy Children

In Section 2.3 we already introduced the Muddy Children example, which we will now use for a comparison between existing explicit model checking methods and our new symbolic methods.

We compared the performance of SMCDEL to DEMO-S5, the explicit model checker optimized for multi-agent S5 [Eij14a]. As a benchmark we use the question "For n children, all of them muddy, how many announcements of 'Nobody knows their own state.' are needed until they do know their own state?". We measured how long each method takes to find and verify the correct answer $(n-1)$ by iteratively evaluating DEL formulas saying that after this many announcements nobody/everybody knows their own state. The exact input can be found in the technical report [Gat18]. To get precise timing results we use the library Criterion [OSu16].

Figure 4.1 shows the results on a logarithmic scale: Explicit model checking with DEMO-S5 quickly becomes unfeasible for more than 12 agents, whereas our symbolic model checker SMCDEL deals with scenarios up to 40 agents in less than a second. We tested both the original SMCDEL using CacBDD [LSX13] and an alternative version based on CUDD [Som12]. The performance is almost the same, but CUDD is slightly faster for less than 20 agents while CacBDD is faster for higher values. Note that we are measuring the performance not only of the BDD packages, but at the same time the Haskell bindings.

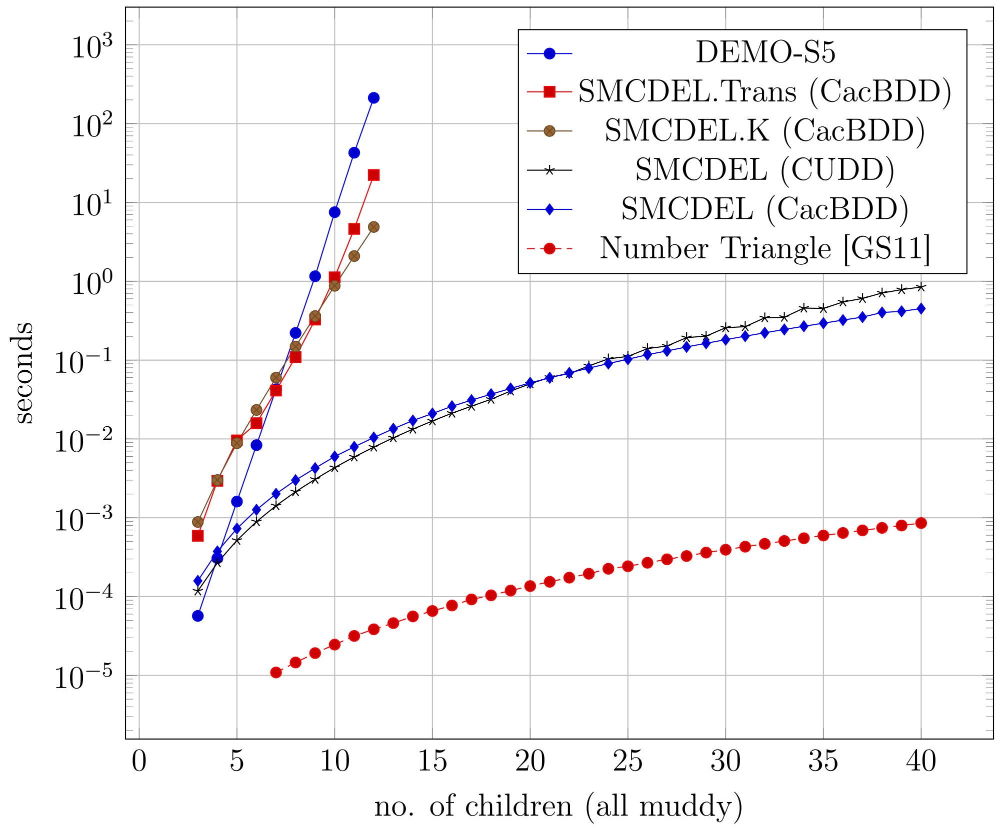


Figure 4.1: Muddy Children benchmark results (logarithmic scale).

Our benchmark is a comparison of different programs and representations at the same time: DEMO-S5 uses a Kripke model, SMCDEL uses the knowledge structure. The speedup could therefore arise at different steps: First at the generation of the initial knowledge structure or Kripke model, second during the update because the formula to be evaluated starts with an announcement, or finally when a formula is evaluated on the result.

To test in which of the steps our new implementation is faster we also benchmarked a variant of SMCDEL which takes a Kripke model as input. It uses the translation from Definition 2.2.6 to construct an equivalent knowledge structure and checks the given formula on that structure. The results are "SMCDEL.Trans (CacBDD)" in Figure 4.1. We can see that the performance of this method is worse than DEMO-S5 for small instances but becomes slightly better for nine or more agents. This reveals that the standard semantics are slow because the generation of large Kripke models takes a long time, and not the evaluation of updates and formulas afterwards.

In some sense this is where theory and practice of model checking part ways, because only the evaluation of formulas is considered part of "model checking" itself, not the time to generate or read in the description of the model. In particular, the computational complexity of model checking is measured with the size of the model as a parameter [AS13]. But this size will depend heavily on the representation: The Kripke model for situations like the Muddy Children grows exponentially in the number of agents, so even if model checking takes time polynomial in the size of the model, it is exponential in the number of agents. In contrast, consider the size of a knowledge structure: For n Muddy Children the initial model is given by $(\{p_1, \dots, p_n\}, \top, O_1 = \{p_1\}, \dots, O_n = \{p_n\})$ which we can write as a string of length $\mathcal{O}(n^2)$. Moreover, the BDDs describing intermediate state laws will maximally have $\lceil \frac{n}{2} \rceil^2$ many nodes.

We also implemented and benchmarked an alternative modeling of Muddy Children given in [GS11]. Inspired by the number triangle, the authors use models without names or indices for agents. Only two *kinds* of agents, the muddy and non-muddy children, are distinguished. Moreover, instead of epistemic relations the model contains *observational states*, which describe the perspective of a type of agents. This yields a model for n agents with only $2n + 1$ instead of 2^n states, as shown in Figure 4.2 for the case $n = 3$.

$$\begin{tikzcd}[column sep=tiny]
& (0,2) \arrow[dl] \arrow[dr] & & (1,1) \arrow[dl] \arrow[dr] & & (2,0) \arrow[dl] \arrow[dr] & \\
(0,3) & & (1,2) & & (2,1) & & \boxed{(3,0)}
\end{tikzcd}$$

Figure 4.2: Triangle model for Muddy Children.

The authors of [GS11] do not provide a formal syntax and semantics and it is impossible to evaluate the standard DEL language on such triangle models. For our implementation we therefore defined the following new language

$$\varphi ::= \neg\varphi \mid \varphi \land \varphi \mid Q \mid K_b \mid \overline{K}_b$$

where Q is a generalized quantifier, b is a bit for muddy or non-muddy, K_b means that all agents of kind b know their own state and $\overline{K}_b$ means that all agents of kind

b do *not* know their own state. The knowledge operators do not take any further formula arguments and the semantics of both operators start with a universal quantifier. It is crucial to note that $\overline{K}_b$ is *not* the negation of K_b. In contrast, $\neg K_b$ means that there is at least one agent not knowing their own state.

Also the updates need to be translated differently than to standard DEL: The first announcement "At least one of you is muddy." is the announcement of a quantifier and for example removes the $(0, 3)$ state at the left end of the lower layer in Figure 4.2. After that, the announcements of "Nobody knows their own state." are given by $\overline{K}_0 \wedge \overline{K}_1$ and each announcement of this formula removes some of the observational states in the upper layer.

Figure 4.3 shows how this language and its semantics can be defined in Haskell. For more details we refer to an appendix of the SMCDEL documentation in [Gat18].

The performance of this number triangle model is impressive, as shown in Figure 4.1. However, the modeling is very specific to the Muddy Children, while DEMO-S5 and SMCDEL are general DEL model checkers. Similar abstractions and concise models might be found for other examples, but they need to be constructed for each specific case. Still, the results are strong evidence that additional abstraction methods such as agent kinds can improve the performance of DEL model checking.

Muddy Children has also been used to benchmark MCMAS [LQR15] but the formula checked there concerns correctness of behavior and not how many rounds are needed. Moreover, the interpreted system semantics of model checkers like MCMAS are very different from DEL. Better suited for a direct comparison between SMCDEL and MCMAS is the protocol for the Dining Cryptographers [Cha88] which we discuss in detail in Section 4.3.

Ending this section, to check whether our more general *belief* structures have similar computational advantages as knowledge structures, we repeated the muddy children benchmark using the BDD encoding for relations instead of observational variables. The runtime of this method is "SMCDEL.K (CacBDD)" in Figure 4.1.

As expected this worsens performance, but for the cases of ten or more agents, model checking on belief structures is still faster than DEMO-S5 which uses partitions. For example, it takes around 15 instead of 200 seconds to check the case of 12 agents.

However, a better comparison would be with the original non-optimized DEMO that can also handle non-S5 models, and should be done with other scenarios than Muddy Children. We leave this as future work.

```haskell
data Kind = Muddy | Clean

type State = (Int,Int)

data McModel = McM [State] [State] State deriving Show

mcModel :: State -> McModel
mcModel cur@(c,m) = McM ostates fstates cur where
  total   = c + m
  ostates = [ ((total-1)-m',m') | m'<-[0..(total-1)] ] -- observ. states
  fstates = [ (total-m',    m') | m'<-[0..total    ] ] -- factual states

posFrom :: McModel -> State -> [State]
posFrom (McM _ fstates _) (oc,om) =
  filter (`elem` fstates) [ (oc+1,om), (oc,om+1) ]

obsFor :: McModel -> Kind -> State
obsFor (McM _ _ (curc,curm)) Clean = (curc-1,curm)
obsFor (McM _ _ (curc,curm)) Muddy = (curc,curm-1)

posFor :: McModel -> Kind -> [State]
posFor m status = posFrom m $ obsFor m status

type Quantifier = State -> Bool

some :: Quantifier
some (_,b) = b > 0

data McFormula = Neg McFormula    -- negations
               | Conj [McFormula] -- conjunctions
               | Qf Quantifier    -- quantifiers
               | KnowSelf Kind    -- all b agents DO    know their status
               | NotKnowSelf Kind -- all b agents DON'T know their status

nobodyknows,everyoneKnows :: McFormula
nobodyknows   = Conj [ NotKnowSelf Clean, NotKnowSelf Muddy ]
everyoneKnows = Conj [    KnowSelf Clean,    KnowSelf Muddy ]

eval :: McModel -> McFormula -> Bool
eval m (Neg f)          = not $ eval m f
eval m (Conj fs)        = all (eval m) fs
eval (McM _ _ s) (Qf q) = q s
eval m@(McM _ _ (_,curm)) (KnowSelf    Muddy) = curm==0 ||
                                                length (posFor m Muddy) == 1
eval m@(McM _ _ (curc,_)) (KnowSelf    Clean) = curc==0 ||
                                                length (posFor m Clean) == 1
eval m@(McM _ _ (_,curm)) (NotKnowSelf Muddy) = curm==0 ||
                                                length (posFor m Muddy) == 2
eval m@(McM _ _ (curc,_)) (NotKnowSelf Clean) = curc==0 ||
                                                length (posFor m Clean) == 2

update :: McModel -> McFormula -> McModel
update (McM ostates fstates cur) f =
  McM ostates' fstates' cur where
    fstates' = filter (\s -> eval (McM ostates fstates s) f) fstates
    ostates' = filter (not . null . posFrom (McM [] fstates' cur)) ostates

step :: State -> Int -> McModel
step s 0 = update (mcModel s) (Qf some)
step s n = update (step s (n-1)) nobodyknows
```

Figure 4.3: Part of the SMCDEL.Other.MCTRIANGLE module.

4.2 Drinking Logicians

Another entertaining example for epistemic reasoning among multiple agents is the story of the Drinking Logicians. Figure 4.4 shows a comic version of this scenario, published in [Spi11].

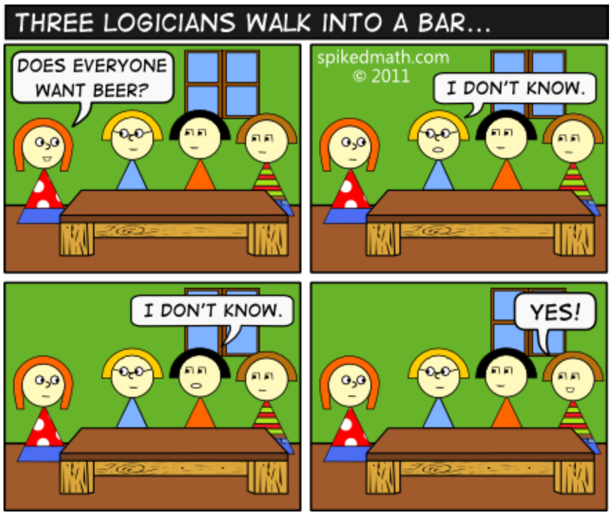


Figure 4.4: Drinking Logicians. CC-BY-NC-SA SpikedMath.com [Spi11].

Recall that in the Muddy Children example each child can observe the status — whether they are muddy — of everyone else, but not their own. The Drinking Logicians example is exactly the dual situation: Here each agent observes/knows their own status — whether they want a beer — but cannot observe the state of the other agents.

Let p_1 mean that agent a wants a beer, p_2 that agent b wants a beer and p_3 that agent c wants a beer. The knowledge structure for three drinking logicians is

$$(V = \{p_1, p_2, p_3\},\ \theta = \top,\ O_a = \{p_1\},\ O_b = \{p_2\},\ O_c = \{p_3\})$$

and the actual state in which everyone is thirsty is $\{p_1, p_2, p_3\}$. Figure 4.5 shows an input file for the command line version of SMCDEL. It describes the initial knowledge structure with three drinking logicians and the three following specifications to be checked.

First, agent a says "I don't know". We model this as an announcement that a does not know whether $p_1 \land p_2 \land p_3$ holds. After this announcement, it is common knowledge among all three agents that p_1 is true:

$$[!\neg K_a^?(p_1 \land p_2 \land p_3)]C_{a,b,c}p_1$$

Second, after two announcements, c knows whether everyone wants beer:

$$[!\neg K_a^?(p_1 \land p_2 \land p_3)][!\neg K_b^?(p_1 \land p_2 \land p_3)]K_c^?(p_1 \land p_2 \land p_3)$$

```
VARS    1, 2, 3

LAW     Top

OBS     a: 1
        b: 2
        c: 3

VALID?  [ ! ~ a knows whether (1 & 2 & 3) ]
        (a,b,c) comknow that 1

VALID?  [ ! ~ a knows whether (1 & 2 & 3) ]
        [ ! ~ b knows whether (1 & 2 & 3) ]
        c knows whether (1 & 2 & 3)

VALID?  ( < ! ~ a knows whether (1 & 2 & 3) >
          < ! ~ b knows whether (1 & 2 & 3) >
          < ! c knows that (1 & 2 & 3) > Top )
        iff (1 & 2 & 3)
```

Figure 4.5: Input for three Drinking Logicians.

Third, the three announcements can be made in this order iff everyone wants beer:

$$(\langle !\neg K^?_a(p_1 \land p_2 \land p_3)\rangle\langle !\neg K^?_b(p_1 \land p_2 \land p_3)\rangle\langle !K_c(p_1 \land p_2 \land p_3)\rangle\top) \leftrightarrow (p_1 \land p_2 \land p_3)$$

Just like the Muddy Children example, the Drinking Logicians can be generalized to any number of agents. In Table 4.1 we show how long it takes SMCDEL to generate and check the model for up to 400 agents.

n	seconds
3	0.12
10	0.14
100	0.22
200	0.61
400	2.87

Table 4.1: Runtime results for larger numbers of Drinking Logicians.

4.3 Dining Cryptographers

A scenario which fits nicely into both the framework of Dynamic Epistemic Logic and that of epistemic temporal logics (see Section 1.5), is the story of the dining cryptographers, first described in a well-known paper by David Chaum:

> "Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maître d'hôtel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying." [Cha88]

They can accomplish this with the following protocol: Every pair of cryptographers flips a coin in such a way (e.g. under the table) that only those two see the result. Then everyone announces whether the two coins they saw were different. But, there is an exception: If one of them paid, then this person says the opposite. After these announcements are made, the cryptographers can infer that the NSA paid iff the number of people saying that they saw the same result on both coins is 1 or 3. Figure 4.6 shows an example of how the agents could reason.

More formally, we use boolean variables and the XOR function as follows. Let p_0 mean that the NSA paid, p_i for $i \in \{1,2,3\}$ that i paid and let p_k for $k \in \{4,5,6\}$ represent the shared coins. The scenario can then be modeled by the knowledge structure

$$\mathcal{F} = \left(\begin{array}{l} V = \{p_0, \dots, p_k\},\ \theta = \bigvee \{p_i \sqsubseteq \{p_0, \dots, p_3\} \mid i \in \{0, \dots, 3\}\}, \\ O_1 = \{1,4,5\},\ O_2 = \{2,4,6\},\ O_3 = \{3,5,6\} \end{array} \right)$$

where intuitively the state law θ is saying that someone must have paid but not two of the agents or the NSA at the same time.

For an analysis using Kripke models, see [EO07], which discusses explicit model checking of the Dining Cryptographers with DEMO.

Recall the abbreviations for *announcing whether* $[!?\varphi]$ from Definition 1.2.1 and exclusive disjunction $\bigoplus$ from Definition 1.0.1. The announcements made by the three dining cryptographers can then be formalized as three public announcements:

$$[?!(\oplus\{p_1, p_4, p_5\})]\ [?!(\oplus\{p_2, p_4, p_6\})]\ [?!(\oplus\{p_3, p_5, p_6\})]$$

For the protocol to work it is actually enough to broadcast the XOR of all announcements made by the agents (though this reveals less information). Hence for efficiency we can also replace them with a single announcement:

$$[?! \oplus \{(\oplus\{p_1, p_4, p_5\}), (\oplus\{p_2, p_4, p_6\}), (\oplus\{p_3, p_5, p_6\})\}]$$

Figure 3.12 in the previous chapter shows the input for the command line interface of SMCDEL for the case of three agents.

The following goal of the protocol was translated to an epistemic temporal logic and then model checked in [LQR15]:

> "If cryptographer 1 did not pay, then after the announcements are made, 1 either knows that no cryptographers paid, or that someone paid, but in this case 1 does not know who did."

Following the translation ideas in [Ben+09; DHR13] we can formalize the same statement in $\mathcal{L}_P$ as

$$\neg p_1 \to [\dots](K_1(\bigwedge_{i=1}^{n} \neg p_i) \vee (K_1(\bigvee_{i=2}^{n} p_i) \wedge \bigwedge_{i=2}^{n}(\neg K_1 p_i)))$$

where p_i says that agent i paid and $[\dots]$ is the announcement from above.

The protocol can be generalized for any finite number of Dining Cryptographers. Table 4.2 shows how many propositions we need to model the situation for n agents and how long SMCDEL needs to run to check the above statement. In the next section we provide more runtime results, also using the Dining Cryptographers example, but with a ring topology instead of the complete graph.

n	Propositions	Seconds
10	56	0.0017
20	211	0.0092
40	821	0.0739
80	3241	0.9751
120	7261	3.2806
160	12881	8.1046

Table 4.2: Dining Cryptographers runtime (complete graph, single announcement).

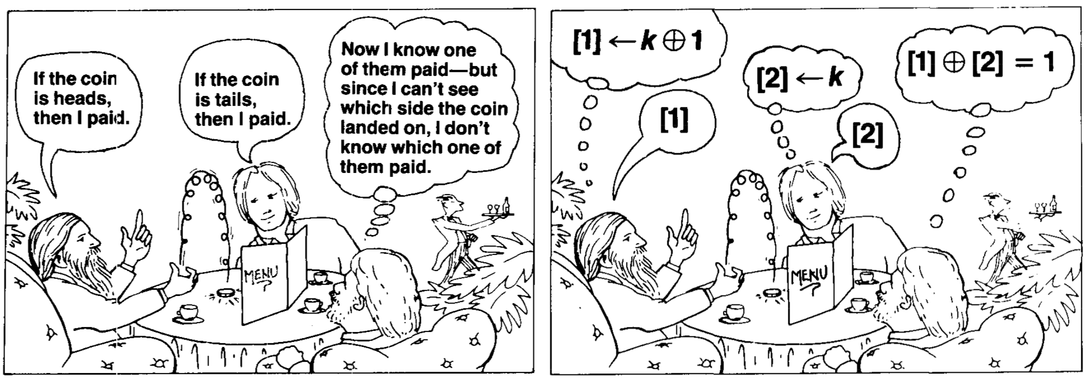


Figure 4.6: The Dining Cryptographers. Drawings from [Cha85] © 1985 Association for Computing Machinery, Inc. Reprinted by permission.

4.4 Comparing DEL and ETL model checkers

In Section 3.1 above we gave an overview of various model checking tools for both temporal and dynamic epistemic logics. Given different implementations, it is natural to model the same problems and examples in each of them to compare their performance in benchmarks. In this section we both summarize previous results and present new benchmark results based on the Dining Cryptographers example from the previous section.

It should be noted that we are comparing many things at the same time: the different languages and logics in which we formalize a protocol, different representations of their semantics, different model checking algorithms and finally different implementations. Our benchmarks are therefore to be taken with a grain of salt, because by design the different programs do not solve exactly the same task. However, we can argue that for specific protocols and scenarios DEL can serve as an alternative to ETL. Concretely, the translations between dynamic and temporal logics discussed in Section 1.6 give a systematic way to prove that we can check the same property of the same protocol in both frameworks. We can then meaningfully compare the performance of SMCDEL with that of temporal model checkers like MCK, MCTK and MCMAS.

In fact, model checking was the motivation in [DHR13] to explore the relation between temporal and dynamic epistemic logics. Based on previous experience with different model checkers, the authors wanted a canonical way to translate and thereby reduce DEL model checking to ETL model checking. To our knowledge this method has not been implemented yet, hence the complexity and performance are not known. SMCDEL is not an implementation of the translation strategy, but works by checking DEL formulas on symbolic structures directly.

In the first publication on MCTK [SSL07] the authors showed that it outperforms MCK and MCMAS in two benchmarks based on the Dining Cryptographers and Russian Cards examples (see Section 4.3 and 4.5, respectively). However, this comparison used MCMAS in version 0.7 and MCK in version 0.1.0 which by now are both outdated.

A more recent comparison between MCMAS in version 1.2.2, MCK and MCTK was done in [LQR15]. Unfortunately the authors do not state which version of MCK and MCTK they used, but based on the time of publication it should have been MCK 1.0.0 and MCTK 1.0.1. We repeated this benchmark to compare the newer versions of MCK, MCTK and MCMAS with SMCDEL. A script to automatically run all four model checkers can be found at `https://github.com/m4lvin/dining-benchmark`. In Table 4.3 we both show our own benchmark results using newer versions of the model checkers, and quote the older results from [LQR15]. Note that "t/o" stands for timeout. Besides SMCDEL 1.0.0 released with this thesis we used MCK 1.1.0, MCMAS 1.3.0 and MCTK 1.0.2.[1]

[1] Special thanks to Xiangyu Luo for providing a compiled version of MCTK.

	results from [LQR15]			reproduced results			SMCDEL	
n	MCK	MCTK	MCMAS	MCK	MCTK	MCMAS	single	separate
3	—	—	—	1.40	0.00	0.00	0.14	0.14
4	—	—	—	1.83	0.00	0.00	0.14	0.14
5	1.4	0.024	0.017	11.59	0.00	0.02	0.14	0.14
10	74.7	0.128	0.091	—	0.04	0.22	0.14	0.37
15	—	—	—	—	0.09	0.87	0.14	26.92
20	47937	34.790	0.667	—	0.37	3.63	0.14	—
30	t/o	2.946	1.476	—	1.70	16.19	0.14	—
40	t/o	20.786	5.053	—	4.30	54.13	0.15	—
50	t/o	72.444	13.437	—	6.23	20.73	0.15	—
60	t/o	t/o	14.180	—	8.80	49.71	0.16	—
70	—	—	—	—	20.44	80.37	0.18	—
80	—	—	—	—	37.83	465.17	0.19	—

Table 4.3: Dining Cryptographers runtime in seconds (ring topology).

Note that the runtime for SMCDEL with a single announcement is much lower than in the previous chapter. This is to be expected, because we now use the ring topology where each cryptographer only shares bits with two other agents — intuitively their left and right neighbors on the round dinner table. This variant of the protocol is also checked by the other programs and reduces the number of shared bits from $\frac{n(n-1)}{2}$ to n.

The "separate" column shows the results from an alternative check in which the public announcements are not combined into one step of broadcasting their XOR, but in a sequence of n separate announcements.

We think that this benchmark shows both the strengths and weaknesses of our implementation and maybe Dynamic Epistemic Logic in general: Reasoning about knowledge and its dynamics can be done fast. But if we want to be more precise about time and our model contains a long sequence of events such as different announcements, then the model checkers for temporal logics are faster.

When comparing the different temporal model checkers to each other, we were unable to reproduce the results from [LQR15]. This could be due to improvements in newer versions, the fact that we used a newer processor, or different parameters and options to fine-tune each model checker and BDD package. In general, our new results show that MCTK is faster than MCMAS, while they are both clearly faster than MCK. Two peculiar results are the one for MCTK with 30 agents in the statistics from [LQR15] and the one for MCMAS with 50 agents. These two are outliers in the sense that the runtime is lower than for the instance with less agents. Usually the model checking task becomes harder with more agents and we would expect runtime to increase monotonically. As our main focus here is not temporal logic, we did not explore these "gaps" further.

Besides in their performance, model checkers also differ in their usability. A comparison between MCK, MCMAS and DEMO in [Dit+06] also assessed the time it took to write models and specifications. Their conclusion was that given familiarity with the language and tool, it takes about the same time to formalize a given scenario in DEMO as in MCK and thereby in DEL as in ETL.

They also argue that MCK has a more succinct and intuitive input syntax which is better suited to specify protocols than the syntax of DEMO. Currently also SMCDEL provides no direct way to specify a protocol or the behavior of agents. However, we can use wrapper functions for protocols and epistemic planning problems, as we will show in the following sections with further examples. In general, SMCDEL should provide the same level of usability as DEMO, with much better performance thanks to symbolic representation. Additionally, the command line and web interfaces make the main functions of SMCDEL usable without any knowledge of Haskell, using a syntax that is more similar to the languages SMV and ISPL that are used by temporal model checkers.

4.5 Russian Cards

As another case study, we applied our symbolic model checker to the Russian Cards Problem. One of its first logical analyses is [Dit03]. The problem has since gained notable attention as an intuitive example of cryptography that is information-theoretically secure and does not rely on computational hardness assumptions [FG16; Cor+15; LF17]. The basic version of the problem is this:

> Seven cards, enumerated from 0 to 6, are distributed between Anne, Bob and Crow such that Anne and Bob both receive three cards and Crow one card. It is common knowledge which cards exist and how many cards each agent has. Everyone knows their own but not the others' cards. The goal of Anne and Bob now is to learn each others cards without Crow learning them. They can only communicate via public announcements.

The initial situation can easily be modeled in a knowledge structure. We use a vocabulary with 21 atomic propositions, each saying that a specific agent has a specific card. In the state law we then say that each card must be with exactly one of the agents. Finally, each agent gets 7 observational variables to encode that they see their own cards.

We do not include a figure of the full knowledge structure here, because the BDD of the state law has 129 nodes. Still, it is generated within a second and can easily be shown with the command `disp rusSCN` after loading the module `SMCDEL.Examples.RussianCards` of SMCDEL.

Many different solutions exist but here we will focus on the so-called five-hands protocols (and their extensions with six or seven hands): First Anne makes an announcement of the form "My hand is one of these: ...". If her hand is 012 she could for example take the set $\{012, 034, 056, 135, 246\}$. It can be checked that this announcement does not tell Crow anything, independent of which card it has. In contrast, Bob will be able to rule out all but one of the hands in the list depending on his own hand. Hence the second and last step of the protocol is an announcement by Bob about which card Crow has. For example, if Bob's hand is 345 he would finish the protocol with "Crow has card 6.".

Verifying this protocol for the fixed deal 012|345|6 with our symbolic model checker takes less than a second. Compared to that, the first DEMO implementation [Dit+06] needed nine seconds to check one protocol, and similar specifications for MCK and MCMAS took more than 100 seconds. Optimized specifications also given in [Dit+06] got this down to four seconds for DEMO and less than a second for MCK with CUDD.

These results are not directly comparable, as these benchmarks were done on older computers. For now we only conjecture that newer versions of MCK, MCTK and MCMAS will also be much faster, and leave it as future work to repeat the whole study of [Dit+06].

An advantage of the BDD representation is that checking multiple protocols in a row does not take much longer than checking only one protocol, because the BDD package caches results and the BDD for the state law is only generated once. We can use this to not just verify but *find* all 5/6/7-hands protocols, with the following combination of manual reasoning and brute-force.

By Proposition 32 in [Dit03], safe announcements from Anne never contain two hands with multiple cards in common. If we also assume that the hands are lexicographically ordered, this leaves us with 1290 possible lists of five, six or seven hands. Only some of them are safe announcements which can be used by Anne. We can find them by checking the corresponding 1290 formulas expressing that an announcement works as part of a successful protocol. Our model checker can filter out the 102 safe announcements within 1.6 seconds, generating and verifying the same list as in [Dit03] where it was manually generated. Out of the 102 announcements, there are 60 in which Alice announces a set of five cards, 36 with six cards and six with seven cards.

Going further, suppose we do not know any shortcuts like Proposition 32 from [Dit03] to restrict the search space. This perspective was adopted in [Eng+15] and turns the puzzle into an epistemic planning problem. If we only fix that Alice will announce five hands, including her own which w.l.o.g. is 012, then she has to pick four other hands of three cards each. The number of possible actions is then 46376. It takes our model checker about 160 seconds to find the 60 safe announcements among them. Finally, if we relax the condition that Bob will answer with "Crow has card 6" but instead consider "Crow has card n" for any card n, the search space grows by a factor of 7 to 324632. It then takes SMCDEL around 20 minutes to find the solutions. None of the additional plans are successful, hence the same 60 plans are generated.

4.6 Sum and Product

Maybe the most famous example in the DEL literature after the Muddy Children is the Sum and Product puzzle [Fre69, translated from Dutch]:

> A says to S and P: "I chose two numbers x, y such that $1 < x < y$ and $x + y \leq 100$. I will tell $s = x + y$ to S alone, and $p = xy$ to P alone. These messages will stay secret. But you should try to calculate the pair (x, y)". He does as announced. Now follows this conversation: P says: "I do not know it." S says: "I knew that." P says: "Now I know it." S says: "Now I also know it." Determine the pair (x, y).

An overview of the literature on this puzzle and a solution using standard DEL with explicit models can be found in [DRV08].

Our model checker can also solve this classic and we can improve upon the results of existing implementations. However, this comes with a trade-off in convenience. In DEMO-S5 [Eij14a] Kripke models are parameterized with a type, as illustrated in Example 3.1.1. This allows the user to encode information in possible worlds directly. For example, the worlds in a model for Sum and Product can be pairs of integers. In contrast, because of the underlying BDD representation, knowledge structures have to be completely propositional and we have to use an encoding like the following manual translation.

To represent numbers we use binary encodings for x, y, $s := x + y$ and $p := xy$. Recall that $\lceil \cdot \rceil$ denotes the smallest natural number not less than the argument. We need $\lceil \log_2 N \rceil$ propositions for every variable that should take values up to N. For example, suppose to represent $x \leq 100$ we use $p_1, \ldots, p_7$. The statement $x = 5$ is then encoded as $p_1 \land p_2 \land p_3 \land p_4 \land \neg p_5 \land p_6 \land \neg p_7$, corresponding to the bit-string `0000101` for 5. We map `1` to negations because this is easier to implement using the `powerset` function that is already part of SMCDEL, but the opposite mapping would work just as well.

Altogether we need seven variables for each of x, y and s, and twelve variables for p because it can take values up to 2500. We will get back to this way of encoding numeric variables and define a general version of it more formally in Section 5.1. Figure 4.7 shows how to implement the encoding for Sum and Product. We also implement am abbreviation `xyAre` to say that (x, y) is the actual pair.

Given this encoding, we have propositional formulas for $x = n$ etc. and can use them to formalize the puzzle as usual [DHK07, Section 4.11]. The state law for Sum and Product is a big disjunction over all possible pairs of x and y with the given restrictions. It is here where we ensure that s and p are actually the sum and the product of n and m:

$$\bigvee \{x = n \land y = m \land s = n+m \land p = n \cdot m \mid 2 \leq n < m \leq 100, n+m \leq 100\} \tag{4.1}$$

To let the agents S and P know the values of s and p respectively, we define the observational variables $O_S := \{s_1, \ldots, s_7\}$ and $O_P := \{p_1, \ldots, p_7\}$. Now we

```
pairs :: [(Int, Int)]
pairs = [(x,y) | x<-[2..100], y<-[2..100], x<y, x+y<=100]

xProps, yProps, sProps, pProps :: [Prp]
xProps = [(P  1)..(P  7)]
yProps = [(P  8)..(P 14)]
sProps = [(P 15)..(P 21)]
pProps = [(P 22)..(P 33)]

sapAllProps :: [Prp]
sapAllProps = xProps ++ yProps ++ sProps ++ pProps

xIs, yIs, sIs, pIs :: Int -> Form
xIs n = booloutofForm (powerset xProps !! n) xProps
yIs n = booloutofForm (powerset yProps !! n) yProps
sIs n = booloutofForm (powerset sProps !! n) sProps
pIs n = booloutofForm (powerset pProps !! n) pProps

xyAre :: (Int,Int) -> Form
xyAre (n,m) = Conj [ xIs n, yIs m ]

sapExplainState :: [Prp] -> String
sapExplainState truths = concat
  [ "x = ", explain xProps, ", y = ", explain yProps, "
  , x+y = ", explain sProps, " and x*y = ", explain pProps ] where
    explain = show . nmbr truths

nmbr :: [Prp] -> [Prp] -> Int
nmbr truths set = fromMaybe (error "Value not found") $
  elemIndex (set `intersect` truths) (powerset set)
```

Figure 4.7: Encoding Sum and Product with boolean propositions.

can use the usual formulas to say that an agent knows a variable and that the statements of the dialogue can be truthfully announced. The solutions to the puzzle are those states where this conjunction holds. Both the knowledge structure, the statements and the protocol are specified in code in Figure 4.8.

We can then ask in which states the formula characterizing a solution holds and use `sapExplainState` from Figure 4.7 to translate the state back to numbers:

```
λ> whereViaBdd sapKnStruct sapProtocol
[[P 1,P 2,P 3,P 4,P 6,P 7,P 8,P 9,P 10,P 13,P 15,P 16,P 18,
  P 19,P 20,P 22,P 23,P 24,P 25,P 26,P 27,P 30,P 32,P 33]]

λ> map sapExplainState (whereViaBdd sapKnStruct sapProtocol)
["x = 4, y = 13, x+y = 17 and x*y = 52"]
```

It takes less than two seconds to find this unique solution. In particular this is faster than the implementation in [Luo+08] which is also based on BDDs. However, it is still slower than an optimized version of explicit model checking with DEMO-S5 which can do it in less than one second. As SMCDEL includes a full copy of DEMO-S5, it is easy to compare the two directly. We provide a simple benchmark that outputs Figure 4.9.

```
sapKnStruct :: KnowStruct
sapKnStruct = KnS sapAllProps law obs where
  law = boolBddOf $ Disj [ Conj [xyAre (x,y), sIs (x+y), pIs (x*y)]
                         | (x,y) <- pairs ]
  obs = [ (alice, sProps), (bob, pProps) ]

sapKnows :: Agent -> Form
sapKnows i = Disj [ K i (xyAre p) | p <- pairs ]

sapForm1, sapForm2, sapForm3 :: Form
sapForm1 = K alice $ Neg (sapKnows bob) -- Sum: I knew that you didn't know
sapForm2 = sapKnows bob                 -- Product: Now I know the numbers
sapForm3 = sapKnows alice               -- Sum: Now I also know the numbers

sapProtocol :: Form
sapProtocol = Conj [ sapForm1
                   , PubAnnounce sapForm1 sapForm2
                   , PubAnnounce sapForm1 (PubAnnounce sapForm2 sapForm3) ]
```

Figure 4.8: Sum and Product knowledge structure and formulas.

Our specification is based on that given in [DRV08] where the Sum and Product puzzle was solved using the original DEMO. That is, we adopted the files from `http://www.cs.otago.ac.nz/staffpriv/hans/sumpro/` to define a model for DEMO-S5 instead of DEMO and used the same formulas and updates. It is clear that DEMO-S5 is much faster than the original DEMO.

The authors of [DRV08] use a trick to speed up model checking: To state that an agent knows the values of x and y we usually use a big disjunction

$$\bigvee \{K_i(x = n \land y = m) \mid n, m \in \{1, \dots, 100\}\}$$

but evaluating the more complex conjunction

$$\bigwedge \{(x = n \land y = m) \rightarrow K_i(x = n \land y = m) \mid n, m \in \{1, \dots, 100\}\}$$

which given factivity is equivalent on our model, is faster in DEMO. Intuitively, this is because for the first formula DEMO computes the list of reachable worlds

```
$ stack bench smcdel:bench:bench-sumandproduct
[...]
Benchmarking the complete run.
*** Running DEMO_S5 ***
Mo [(4,13)] [Ag 0,Ag 1] [] [(Ag 0,[[(4,13)]]),(Ag 1,[[(4,13)]])] [(4,13)]
This took 0.77287063s seconds.

*** Running SMCDEL ***
The solution is:
x = 4, y = 13, x+y = 17 and x*y = 52
This took 1.2729152s seconds.
```

Figure 4.9: Benchmark results for Sum and Product.

before starting to check if they agree on the numbers and repeats this process for all possible n and m. With the guarded version, DEMO first checks if n and m are the values at the actual world. Only if that is the case will DEMO go on and compute the reachable set of worlds.

The same trick does *not* speed up our symbolic algorithm. In fact, using the original big disjunction is faster, probably because less calls to the BDD package are made. We therefore use the original disjunction, as defined in the function `sapKnows` in Figure 4.8.

One may be surprised that the symbolic approach is slower here. This is probably due to a well-known problem already mentioned in [Bry86]: BDD representations of *products* tend to be large. Concretely, the BDD of the state law formula 4.1 given above on page 122 including the restriction $p = n \cdot m$ has 21258 nodes (computed using the `sizeOf` function from HasCacBDD). Interestingly, an interleaving variable order which places bits of the same significance near to each other leads to a smaller state law BDD with only 5273 nodes. However, it does not speed up the whole process of generating and checking. The reader will understand that we do not include drawings of either BDD here.

Our program thus spends most of its time to build the BDD of the state law before it can actually check any given formula. We can also see this by running an alternative benchmark. Using the criterion library [OSu16] we compare only the actual model checking processes, excluding the time it takes to generate the Kripke model or knowledge structure in the beginning. Figure 4.10 show the output of this alternative benchmark where SMCDEL is almost as fast as DEMO-S5.

```
$ stack bench :bench-sumandproduct --benchmark-arguments checkingOnly
[...]
Benchmarking only the checking, without model generation.
benchmarking checkingOnly/DEMO-S5
time                 788.4 ms   (787.8 ms .. 788.8 ms)
                     1.000 R²   (1.000 R² .. 1.000 R²)
mean                 787.9 ms   (787.6 ms .. 788.1 ms)
std dev              280.9 μs   (0.0 s .. 321.4 μs)
variance introduced by outliers: 19% (moderately inflated)

benchmarking checkingOnly/SMCDEL
time                 859.3 ms   (748.0 ms .. 967.9 ms)
                     0.998 R²   (0.992 R² .. 1.000 R²)
mean                 832.8 ms   (809.4 ms .. 846.1 ms)
std dev              20.79 ms   (0.0 s .. 22.88 ms)
variance introduced by outliers: 19% (moderately inflated)
```

Figure 4.10: Alternative Sum and Product benchmark using *criterion* [OSu16].

4.7 Sally and Anne

Most of our examples so far were clearly about knowledge and thus modeled with S5. They also did not involve factual change. To show how our implementation of belief structures and transformers works and performs, we now consider a classic example from the literature in which both false beliefs and factual change play an important role.

The Sally-Anne false belief task is a famous example from Psychology used to illustrate and test for a theory of mind. The basic version goes as follows (adapted from [BLF85]):

> Sally has a basket, Anne has a box. Sally also has a marble and puts it in her basket. Then Sally goes out for a walk. Anne moves the marble from the basket into the box. Now Sally comes back and wants to get her marble. Where will she look for it?

We also show a comic version which was used in experiments in Figure 4.11.

To answer the question where Sally will look for her marble correctly, one needs to realize that Sally did not observe that the marble was moved. She will thus look for it in the basket. Our choice to implement this example is also motivated by a recent interest in the computational complexity of theory of mind [Pol15; PRS15] where our symbolic representation might provide a new perspective.

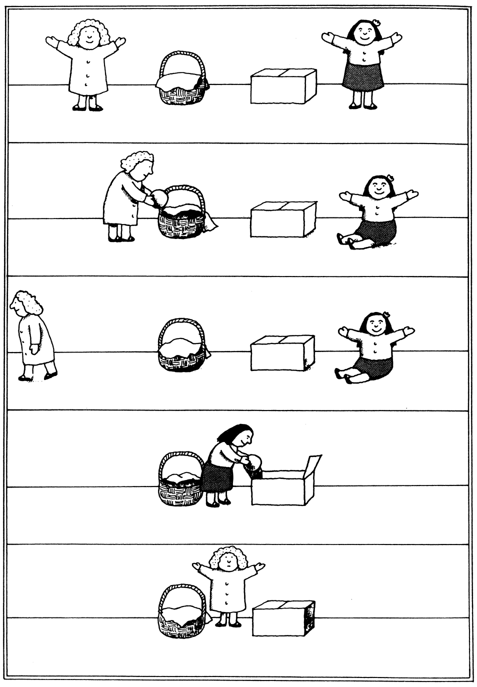


Figure 4.11: Sally and Anne. Drawing by Axel Scheffler used with permission.

We now translate a DEL modeling from [Bol14] to our framework. For simplicity we adopt the first naive modeling given there, leaving it as future work to adopt the refinement with edge-conditions and other improvements.

We use the vocabulary $V = \{p, t\}$ where p means that Sally is in the room and t that the marble is in the basket. In the initial scene Sally is in the room, the marble is not in the basket and both of this is common knowledge:

$$(\mathcal{F}_0, s_0) = ((V = \{p, t\},\ \theta = (p \land \neg t),\ \Omega_{\mathrm{S}} = \top,\ \Omega_{\mathrm{A}} = \top), \{p\})$$

The sequence of events is:

$\mathcal{X}_1$: Sally puts the marble in the basket: $((\varnothing, \top, \{t\}, \theta_-(t) = \top, \top, \top), \varnothing)$.

$\mathcal{X}_2$: Sally leaves: $((\varnothing, \top, \{p\}, \theta_-(p) = \bot, \top, \top), \varnothing)$.

$\mathcal{X}_3$: Anne puts the marble in the box, not observed by Sally:
$((\{q\}, \top, \{t\}, \theta_-(t) = (\neg q \to t) \land (q \to \bot), \neg q', q \leftrightarrow q'), \{q\})$.

Here q is a fresh proposition to distinguish two possible events, namely whether Anne moves the marble or not. The change law θ_- then updates t, depending on the actual event: If q is false, then t stays as it is. If q is true, then t is set to false.

$\mathcal{X}_4$: Sally comes back: $((\varnothing, \top, \{p\}, \theta_-(p) = \top, \top, \top), \varnothing)$.

We calculate the result in Figure 4.12, using Lemma 2.12.2 to remove superfluous variables after the updates.

$$\begin{array}{lll}
 & ((\{p,t\}, (p \land \neg t), \top, \top), p) & \mathcal{F}_0 \\
\times & ((\varnothing, \top, \{t\}, \theta_-(t) = \top, \top, \top), \varnothing) & \mathcal{X}_1 \\
= & ((\{p,t,t^\circ\}, (p \land \neg t^\circ) \land t, \top, \top), \{p,t\}) & \\
 & & \\
\times & ((\varnothing, \top, \{p\}, \theta_-(p) = \bot, \top, \top), \varnothing) & \mathcal{X}_2 \\
= & ((\{p,t,t^\circ,p^\circ\}, (p^\circ \land \neg t^\circ) \land t \land \neg p, \top, \top), \{t, p^\circ\}) & \\
\equiv_V & ((\{p,t\}, t \land \neg p, \top, \top), \{t\}) & \\
 & & \\
\times & ((\{q\}, \top, \{t\}, \theta_-(t) = (\neg q \to t) \land (q \to \bot), \neg q', q \leftrightarrow q'), \{q\}) & \mathcal{X}_3 \\
= & ((\{p,t,q,t^\circ\}, t^\circ \land \neg p \land (t \leftrightarrow ((\neg q \to t^\circ) \land (q \to \bot))), \neg q', q \leftrightarrow q'), \{q\}) & \\
= & ((\{p,t,q,t^\circ\}, t^\circ \land \neg p \land (t \leftrightarrow \neg q), \neg q', q \leftrightarrow q'), \{q\}) & \\
\equiv_V & ((\{p,t,q\}, \neg p \land (t \leftrightarrow \neg q), \neg q', q \leftrightarrow q'), \{q\}) & \\
 & & \\
\times & ((\varnothing, \top, \{p\}, \theta_-(p) = \top, \top, \top), \varnothing) & \mathcal{X}_4 \\
= & ((\{p,t,q,p^\circ\}, \neg p^\circ \land (t \leftrightarrow \neg q) \land p, \neg q', q \leftrightarrow q'), \{p,q\}) & \\
\equiv_V & ((\{p,t,q\}, (t \leftrightarrow \neg q) \land p, \neg q', q \leftrightarrow q'), \{p,q\}) & \mathcal{F}_4
\end{array}$$

Figure 4.12: Sally-Anne on belief structures and transformers.

Finally, we can check that Sally believes that the marble is in the basket:

$$\begin{aligned}(\mathcal{F}_4, \{p,q\}) \vDash \Box_{\mathrm{S}} t &\iff \{p,q\} \vDash \forall V'(\theta' \rightarrow (\Omega_{\mathrm{S}} \rightarrow t')) \\ &\iff \{p,q\} \vDash \forall \{p', t', q'\}((t' \leftrightarrow \neg q') \land p' \rightarrow (\neg q' \rightarrow t')) \\ &\iff \{p,q\} \vDash \top\end{aligned}$$

Due to the small structure, running SMCDEL on this example is almost instant: The computation including the transformation takes 0.25 seconds.

4.8 Epistemic Planning

Planning problems in general are given by an initial state, a set of available actions and a goal that should be reached. Strategic planning is a field which expanded throughout the last three centuries, with the first standardization of the *Planning Domain Definition Language* (PDDL) in 1998 [McD+98], which has since been revised and extended regularly [Kov11].

All three parts of a planning problem can have epistemic aspects: An agent can be uncertain about the initial state, the actions might generate uncertainty, and the goal might be that an agent should (not) know something. Official versions of PDDL do not include operators for knowledge or belief, but recently variants of DEL have been used to describe epistemic planning problems, for example in [BA11; WL12; Eng+15]. While epistemic planning is often done using knowledge bases, combining this work with our symbolic methods for DEL could bring logic back in the game.

In particular, some epistemic planning problems can be reduced to the DEL model checking problem. Hence we can also use SMCDEL as a tool for epistemic planning. The Russian cards example above in Section 4.5 was already an example for this, and the following definition provides a general method. We adopt a common distinction from the literature, between online and offline plans.

4.8.1. Definition. A *planning problem* is a tuple $(\mathcal{F}, \mathfrak{X}, \varphi)$ where $\mathcal{F}$ is a knowledge or a belief structure, $\mathfrak{X}$ is a set of transformers, and φ is a formula from the epistemic language $\mathcal{L}$. We call the first component the *initial situation*, the second the *available moves* and the third the *goal*.

An *offline plan* for a planning problem is a finite sequence of available moves. We say that it *succeeds* iff applying the elements to the initial situation in the given order yields a model or structure in which the goal holds.

Recall how we found the working protocols in the Russian Cards puzzle by checking a formula for each possible protocol saying that this protocol is successful. This reduction of plan-success to model checking can be generalized as follows.

4.8.2. Fact. An offline plan $X_1, \dots, X_k$ succeeds on a planning problem $(\mathcal{F}, \mathfrak{X}, \varphi)$ iff we have $\mathcal{F} \vDash [X_1] \dots [X_k]\varphi$.

We note that this is similar to the definition of a *puzzle* used in the logic of agent-types, utterances and questions in [LW13] and the model checking implementation of the "Hardest Logic Puzzle Ever" in [Gat16].

4.8.3. Example. The Dining Cryptographers from Section 4.3 can be modeled as a planning problem where the initial situation is given by the knowledge structure from page 116, the set of available moves are all public announcements and the goal is $(K_1(\bigwedge_{i=1}^{n} \neg p_i) \lor (K_1(\bigvee_{i=2}^{n} p_i) \land \bigwedge_{i=2}^{n}(\neg K_1 p_i)))$ as explained on page 117. The sequence of "whether" announcements

$$[?!(\oplus\{p_1,p_4,p_5\})][?!(\oplus\{p_2,p_4,p_6\})][?!(\oplus\{p_3,p_5,p_6\})]$$

is a successful offline plan for it.

Such plans are called "offline" because the sequence of moves to be made is fixed in advance and not changed during the execution of the plan. An *online plan* in contrast, may use "If-Then-Else" to evaluate formulas while being executed and use the result to decide between different moves on the spot.

Formally, an online plan can be defined as a directed acyclic graph: Nodes are labeled formulas to be tested and edges are similar to those in binary decision diagrams. Every non-terminal node should have two outgoing edges, one dashed and one solid, labeled with the action to be made and leading to the remaining plan. The *success* of an online plan can then be defined by recursion and again is equivalent to the truth of a formula describing its success. For the formal details we refer to the similar definition of a *solution* in [LW13].

In the module `SMCDEL.Other.Planning` we implement offline and online plans with public announcements, and a `succeeds` function to generate the DEL formula expressing that a given plan is successful. We show the important types and functions in Figure 4.13.

```
class Plan a where
  succeeds :: a -> Form

type OfflinePlan = [(Form,Form)] -- list of (announcement,goal) tuples

instance Plan OfflinePlan where
  succeeds [] = Top
  succeeds ((step,goal):rest) =
    Conj [step, PubAnnounce step goal, PubAnnounce step (succeeds rest)]

data OnlinePlan = Stop | DoAnnounce Form OnlinePlan | IfThenElse Form
    OnlinePlan OnlinePlan

instance Plan OnlinePlan where
  succeeds Stop = Top
  succeeds (DoAnnounce step next) =
    Conj [step, PubAnnounce step (succeeds next)]
  succeeds (IfThenElse check planA planB) =
    Conj [ check `Impl` succeeds planA, Neg check `Impl` succeeds planB ]
```

Figure 4.13: Part of the `SMCDEL.Other.Planning` module.

4.9 Conclusion and Future Work

We saw that SMCDEL can be used to model and check various examples from the literature on epistemic logic, protocol verification and epistemic planning. Some examples that we have modeled in SMCDEL are not part of this chapter. This includes the "What Sum" puzzle as studied in [VR07], and multiple different versions of the "Hundred Prisoners and a Lightbulb" from [DEW10]. We also plan to check more examples of epistemic planning with SMCDEL in the future. Suitable examples are spatial reasoning as modeled in [WL12], variations of the classical problem of the Wolf, Goat and Cabbage as solved in [GV17] and small instances of the card game Hanabi as discussed in [Bar+17, Section 4.1.3.4].

As expected, SMCDEL performs better than the previously existing explicit model checkers DEMO and DEMO-S5. Moreover, for examples where a temporal and a dynamic modeling are equivalent, SMCDEL can also compete with temporal model checkers.

Admittedly, most of the models in this chapter are puzzles and toy examples. But there are two exceptions. First, the Dining Cryptographers example from Sections 4.3 and 4.4, in its general form of "DC-nets" has been implemented many times and is used in practice [GJ04; CF10]. Second, the Russian Cards scenario, discussed in Section 4.5, is a simple version of information-based cryptography and can be generalized to protocols for the secure aggregation of distributed information (SADI) as described in [FG16].

For temporal logic model checkers, other real world applications are mainly the analysis of circuits and programs. For DEL, we think that future applications will focus on protocol analysis and epistemic planning. We hope that SMCDEL will be useful for this research, both when it is used as a tool and when it is further developed and extended to other logics with different semantics, as discussed in Section 2.13.

Chapter 5

Knowing and Inspecting Values

Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt.

Ludwig Wittgenstein: *Tractatus logico-philosophicus*, § 5.6

Standard epistemic logic studies propositional knowledge expressed by "knowing that". However, in everyday life we talk about knowledge in many other ways, such as "knowing what the password is", "knowing how to swim", "knowing why he was late" and so on. Also in the philosophical literature it is debated whether these various kinds of knowledge can be reduced to "knowing that" or are fundamentally different, see for example [Fan17]. Recently the epistemic logics of such "knowing X" expressions are drawing more and more attention (see [Wan18] for a survey).

In this chapter we investigate a specific "knowing X" construction, namely "knowing what" or "knowing the value". Both intuitively and formally, this epistemic modality can also be seen as a generalization of "knowing what the truth-value is".

We discuss and compare three different approaches to modeling the knowledge of values in variants of Dynamic Epistemic Logic. First, in Section 5.1 we discuss the binary encoding for numeric values that we also used in Section 4.6. Second, in Section 5.2 we present register models, an abstraction method to represent Kripke models with numeric variables in a compact way. Third, the main part of this chapter presents Public Inspection Logic (PIL) in Sections 5.3 to 5.6.

For simplicity we assume that all values are numeric, but of course the natural language "knowing what" is more general and not always about numbers — see e.g. the password example above. The binary encoding and register models are limited to finite domains which can be enumerated, but the models for Public Inspection Logic are more general. PIL and its variants can be interpreted on any domain, including infinite sets.

For the whole chapter we use the following simple running example.

5.0.1. Example. Suppose we have two agents, Alice and Bob, and two numeric variables x and y which both can take values from the set $\{0, \ldots, 7\}$. Alice knows the value of x but not that of y and Bob knows the value of y but not that of x and the actual values are $x = 5$ and $y = 7$.

While this is a very small toy example, it will suffice to highlight the differences between the three approaches we present here. In practice, the range for x and y might be much larger. The variables could for example be private keys in a security protocol involving encryption or signatures.

5.1 Binary Encoding

A simple method to represent numbers in standard, propositional Kripke models is a binary encoding which maps each numeric variable with a finite range to multiple boolean variables. We already used such an encoding to check the Sum and Product example in Section 4.6 above. The following definition generalizes the example to all numeric variables with a finite range. Essentially, we map all values of a variable to their bit-string representation, with negations taking the role of ones.

5.1.1. Definition. Consider a numeric variable x such that $0 \leq x \leq M$ for some $M \in \mathbb{N}$. Let $P_x := \{p^x_0, \ldots, p^x_{k-1}\}$ be $k := \lceil \log_2 M \rceil$ many fresh variables and enumerate (a part of) the powerset of P_x with the following map:

$$\begin{array}{llcll} f_x & : & \{0, \ldots, M\} & \to & \mathcal{P}(P_x) \\ & & n & \mapsto & \{p^x_n \in P_x \mid x - (x \,\mathsf{rem}(2^n)) \equiv 0 \mod 2^{n+1}\} \end{array}$$

The *binary encoding* of x is then defined by the boolean formula $f_x(n) \sqsubseteq P_x$ for each $n \leq M$ which we abbreviate as $x = n$.

5.1.2. Example. Suppose we want to encode the variable $0 \leq x \leq 100$ with the actual value $x = 42$. We need $\lceil \log_2 100 \rceil = 7$ boolean variables and the bit-string representation of 42 is 0101010. Hence we have $f_x(42) = \{0, 2, 4, 6\}$ and encode $x = 42$ with $\{p_0, p_2, p_4, p_6\} \sqsubseteq \{p_0, \ldots, p_6\}$, which by definition is equivalent to $p_6 \land \neg p_5 \land p_4 \land \neg p_3 \land p_2 \land \neg p_1 \land p_0$. Note that the BDD of this formula has 7 non-terminal nodes.

Using a function for the powerset and the standard Haskell operator `!!` for elements of a list, it is easy to implement the binary encoding — see Figure 4.7 on page 123.

If we have k many numeric variables $x_1, \ldots, x_k$ which each have values from 0 to corresponding maxima $M_1, \ldots, M_k$, then we can encode all combinations of their values with $\sum_{1 \leq i \leq k} \lceil \log_2 M_i \rceil$ many boolean variables.

Given Definition 5.1.1 to write $x = n$ as a boolean formula, we can then define the statement that a knows the value of x by

$$Kv_a x \quad := \quad \bigvee\{K_i(x = n) \mid n \in \{0, \dots, M\}\}$$

as done in Section 4.6. This formula has the advantage that it is independent of how equalities are translated. The downside is that it becomes much longer for larger ranges. To be precise, its length is $\mathcal{O}(M \log(M))$.

If we use binary encoding in the S5 setting, we can do better with an equivalent shorter formula, because knowing the value of a numeric variable is the same as knowing the value of all its bits, i.e. observing all the corresponding boolean variables. Formally, with the above definition we have the equivalence

$$Kv_a x \;\leftrightarrow\; K_a^? p_0^x \wedge \dots \wedge K_a^? p_{k-1}^x$$

where $K_a^?$ is the knowing-whether operator from Definition 1.1.1. Note that the right part of this equivalence only grows with the numbers of bits needed and not with M directly, so its length is $\mathcal{O}(\log(M))$.

5.1.3. Example. Recall that in Example 5.0.1 Alice knows that $x = 5$ and Bob knows that $y = 7$, and that both variables could take values up to 7. We need 3 boolean variables for each numeric variable, and thus get a Kripke model with $2^6 = 64$ many possible worlds. We show part of it in Figure 5.1, with solid lines for the accessibility relation for Alice and dashed lines for Bob. To save space, we do not encircle each world and mark the actual world with a simple rectangle.

$$\begin{tikzcd}
 & \vdots & \vdots & \vdots \\
\cdots \arrow[r, dash] & \begin{array}{c} \neg p_2^x, p_1^x, \neg p_0^x \\ \neg p_2^y, p_1^y, \neg p_0^y \\ (x = 5,\ y = 5) \end{array} \arrow[r, dash] \arrow[d, dash, dashed] & \begin{array}{c} \neg p_2^x, p_1^x, \neg p_0^x \\ \neg p_2^y, \neg p_1^y, p_0^y \\ (x = 5,\ y = 6) \end{array} \arrow[r, dash] \arrow[d, dash, dashed] & \boxed{\begin{array}{c} \neg p_2^x, p_1^x, \neg p_0^x \\ \neg p_2^y, \neg p_1^y, \neg p_0^y \\ (x = 5,\ y = 7) \end{array}} \arrow[d, dash, dashed] \\
\cdots \arrow[r, dash] & \begin{array}{c} \neg p_2^x, \neg p_1^x, p_0^x \\ \neg p_2^y, p_1^y, \neg p_0^y \\ (x = 6,\ y = 5) \end{array} \arrow[r, dash] \arrow[d, dash, dashed] & \begin{array}{c} \neg p_2^x, \neg p_1^x, p_0^x \\ \neg p_2^y, \neg p_1^y, p_0^y \\ (x = 6,\ y = 6) \end{array} \arrow[r, dash] \arrow[d, dash, dashed] & \begin{array}{c} \neg p_2^x, \neg p_1^x, p_0^x \\ \neg p_2^y, \neg p_1^y, \neg p_0^y \\ (x = 6,\ y = 7) \end{array} \arrow[d, dash, dashed] \\
\cdots \arrow[r, dash] & \begin{array}{c} \neg p_2^x, \neg p_1^x, \neg p_0^x \\ \neg p_2^y, p_1^y, \neg p_0^y \\ (x = 7,\ y = 5) \end{array} \arrow[r, dash] & \begin{array}{c} \neg p_2^x, \neg p_1^x, \neg p_0^x \\ \neg p_2^y, \neg p_1^y, p_0^y \\ (x = 7,\ y = 6) \end{array} \arrow[r, dash] & \begin{array}{c} \neg p_2^x, \neg p_1^x, \neg p_0^x \\ \neg p_2^y, \neg p_1^y, \neg p_0^y \\ (x = 7,\ y = 7) \end{array}
\end{tikzcd}$$

Figure 5.1: Part of a Kripke model with binary encoding for Example 5.0.1.

More compact than the Kripke model is this equivalent knowledge structure:

$$((V = \{p_0^x, p_1^x, p_2^x, p_0^y, p_1^y, p_2^y\},\ \theta = \top,\ O_a = \{p_0^x, p_1^x, p_2^x\},\ O_b = \{p_0^y, p_1^y, p_2^y\}), \{p_1^x\})$$

Note that because the range is exactly what can be represented with three boolean variables, the state law is just $\top$. If not all bit-strings were allowed, θ is where the upper bound would be defined. For example, if the maximum for both x and y was 6, at least one bit of each numeric variable would have to be true, so the state law would be $(p_0^x \vee p_1^x \vee p_2^x) \wedge (p_0^y \vee p_1^y \vee p_2^y)$.

A clear advantage of the binary encoding is that after translating everything to boolean variables, we can use the whole standard machinery for DEL based on propositional logic. In particular, the symbolic methods we presented in the previous chapters can be used to model numeric knowledge in this way.

Finally, note that theoretically the finite range is not needed, for we could also map each numeric variable x to an infinite set of boolean variables that could encode all bit-strings. While this is well-defined in theory, it cannot directly be implemented and used for model checking in practice. We could model *potential infinity*, by only adding additional boolean variables when they are needed for actual values, but in any Kripke model the current upper bounds will still be common knowledge among all the agents.

To really work with infinite models, finite representations using automata such as discussed in [CGP99, Chapter 9 and 15] are more suited, but those are outside the scope of this thesis. We leave it as future work to adapt such methods to model numeric knowledge in DEL and now move on to a second method of representing knowledge about variables with finite ranges.

5.2 Register Models

We saw in the previous section that ignorance about large numbers — agent a does not know the value of x — is not feasible in standard Kripke semantics because it leads to very large models. The binary encoding together with our symbolic knowledge structures mitigate the blow-up somewhat, but working with such models is often counter-intuitive as we have to translate back and forth between numeric and boolean statements.

In this section we discuss *register models*, another version of Kripke models to represent ignorance about numbers. Register models can be seen as compressed versions of Kripke models where possible worlds that play the same role were merged into one world.

We summarize the main results from [Gat14; EG15] where we also presented a sound and complete logic for number guessing games based on register models and discussed applications to cryptographic protocols. We adapt the definitions to fit the notation of previous chapters, hence there are some differences to the original publications.

5.2.1. Definition. A *register model* for agents I and vocabulary V is a tuple $\mathcal{M} = (W, R, \pi)$ where (W, R) is a frame as per Definition 1.1.2 and π is a function that maps each possible world $w \in W$ to a tuple $\pi(w) = (P_w, f_w, C_w^+, C_w^-)$ where

- $P_w \subseteq V$ is the set of atomic propositions true at w,
- $f_w \colon V \to \mathbb{N} \times \mathbb{N} \times \mathcal{P}(\mathbb{N})$ assigns to each variable a *register* (n, m, X) such that if $q \in P_v \cap P_w$, then $f_v(p) = f_w(q) = (n, m, X)$ with $n = m$ and $X = \varnothing$.
- $C_w^+ \subseteq V^2$ and $C_w^- \subseteq V^2$ are relations over V such that no $(p, q) \in C_w^-$ is in the transitive symmetric reflexive closure of C_w^+.

We say that an assignment $h \colon V \to \mathbb{N}$ *agrees* with the world w iff we have for all variables $p, q \in V$ that (i) $f_w(p) = (n, m, X)$ implies $n \leq h(p) \leq m$ and $h(p) \notin X$, (ii) $(p, q) \in C_w^+$ implies $h(p) = h(q)$, and (iii) $(p, q) \in C_w^-$ implies $h(p) \neq h(q)$.

Registers (n, m, X) in Definition 5.2.1 limit the values a variable can take. In the most simple case only one value might be allowed. For example, if we have $f_w(x) = (5, 5, \varnothing)$ then $x = 5$ must hold at w. Alternatively, an interval might be allowed: $f_w(x) = (0, 7, \varnothing)$ says that x must take a value from 0 to 7 at w. The third part excludes values: $f_w(x) = (1, 20, \{3\})$ means that x must take a value from 1 to 20, but not 3 at w. Parts C_w^+ and C_w^- are relations between variables describing equality and inequality constraints that should hold at world w.

We use the same vocabulary for boolean and numeric variables. The constraint on P_w and f_w connects boolean and numeric semantics: Whenever an atomic proposition is true, its numeric variable must be fixed to a single value.

The following is a simple language to be interpreted on register models. For more complex languages with dynamic operators, see [Gat14; EG15].

5.2.2. Definition. The *register language* extends $\mathcal{L}(V)$ and is defined by:

$$\varphi \quad ::= \quad \top \mid p \mid p = p \mid p = N \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_i\varphi$$

where i is an agent and $N \in \mathbb{N}$.

5.2.3. Definition. For a register model $\mathcal{M} = (W, R, \rho)$, a possible world $w \in W$ and a numeric assignment h agreeing with w we define the following interpretation of the register language:

$$\begin{array}{lll}
\mathcal{M}, w, h \models \top & & \text{always} \\
\mathcal{M}, w, h \models p & \text{iff} & p \in P_w \\
\mathcal{M}, w, h \models p_1 = p_2 & \text{iff} & h(p_1) = h(p_2) \\
\mathcal{M}, w, h \models p = N & \text{iff} & h(p) = N \\
\mathcal{M}, w, h \models \neg\varphi & \text{iff} & \text{not } \mathcal{M}, w, h \models \varphi \\
\mathcal{M}, w, h \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, w, h \models \varphi \text{ and } \mathcal{M}, w, h \models \psi \\
\mathcal{M}, w, h \models K_i\varphi & \text{iff} & R_i w w' \text{ and } h' \text{ agreeing with } w' \text{ implies } \mathcal{M}, w', h' \models \varphi
\end{array}$$

We say that φ is *true at a world* w and write $\mathcal{M}, w \models \varphi$ iff for all assignments h agreeing with w we have $\mathcal{M}, w, h \models \varphi$.

One might wonder why we keep the assignment h separate and do not make it part of the worlds. This would indeed be equivalent, but make the models much larger. The point of register models is exactly not to make all possible assignments part of the model. Instead, possible worlds only provide a way to check or generate assignments that agree with them. Register models can thus be seen as an application of *abstraction* as discussed in [CGP99, Section 13.2].

5.2.4. Example. Figure 5.2 shows Example 5.0.1 as a register model, again with solid lines for Alice and dashed lines for Bob. Note that the two variables x and y are now used as numeric and propositional variables at the same time. Our visualization is slightly different from the formal definition of register models. For example, in the top right world we do not show the register of y as a tuple $(0, 7, \{7\})$ but instead use the more standard notation $0 \leq y \leq 7$ and $y \notin \{7\}$. Moreover, there are no equality or inequality constraints in this model, i.e. both C_w^+ and C_w^- are empty, for all w.

We again assume that variables are in the range $\{0, \ldots, 7\}$. But note that the register model would not become larger for a bigger range. In fact, no matter what the maximum for both variables is, it will always consist of only 4 worlds. The register model is thus much smaller than the Kripke model using the binary encoding, which needs 64 worlds and becomes bigger for larger maxima.

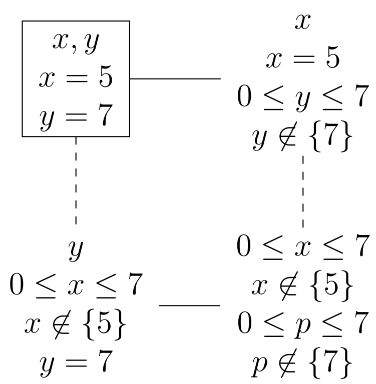


Figure 5.2: Register model for Example 5.0.1.

It is clear that register models are smaller than standard Kripke models. We developed a model checker for register models in [Gat14] and used Monte Carlo methods to speed it up. However, the results of such methods are probabilistic. If we want absolute certainty about register models, any general explicit model checking method has to unravel them to much larger standard Kripke models in which every assignment of values is a separate possible world. Also the symbolic methods developed in Chapter 2 are not applicable to register models without adding again a binary encoding. Given that knowledge structures are already much smaller and more efficient to use than Kripke models, we conclude that for large examples a binary encoding, though less intuitive, is the better tool.

For further details on register models we refer to [Gat14], which includes a sound and complete axiomatization of *Guessing Game Logic* (GGL) based on register models. Extending Definition 5.2.2, GGL has dynamic operators to create new registers with secret values only known to the creating agent. To give a formal example, a validity in this framework is $[p \overset{i}{\leftarrow} 42]K_i(p = 42)$.

Even more expressive is *Epistemic Crypto Logic* (ECL), also introduced in [Gat14] and summarized in [EG15]. It includes operators for modular arithmetic. Combined with locally listening agents, similar to [Dit+13], this language is expressive enough to formalize cryptographic protocols such as the famous Diffie-Hellman key exchange.

However, on register models all agents are logically omniscient and not computationally bounded as one would like them to be in cryptography. Defining and checking security properties, such as the secrecy of a shared key established via the Diffie-Hellman protocol, is possible in ECL, but it relies on syntactic restrictions which terms an agent is allowed to compute.

5.3 Public Inspection

Reasoning about static knowledge is important, but it is also interesting to study changes of knowledge. Recall from the previous chapters that in Public Announcement Logic (PAL) we can update the propositional knowledge of agents with public propositional announcements. Fact 1.2.6 lists the reduction axioms to completely describe the interplay of "knowing that" and "announcing that". Given this, we can also ask: What are natural dynamic counterparts for the knowledge expressed by other expressions such as knowing what, knowing how etc.? How can we formalize "announcing what"?

In the rest of this chapter we study a basic dynamic operation that updates the knowledge of the values of certain variables. The action of *public inspection* is the knowing value counterpart of a public announcement and we will see that it fits well with the logic of knowing value. As an example, consider a sensor to measure the current temperature of a room. It is reasonable to say that after using the sensor you will know the temperature of the room. But it is not feasible to encode this with a standard public announcement since it results in a possibly infinite formula:

$$[!t = 27.1\ ^\circ\mathrm{C}]K(t = 27.1\ ^\circ\mathrm{C}) \wedge [!t = 27.2\ ^\circ\mathrm{C}]K(t = 27.2\ ^\circ\mathrm{C}) \wedge \ldots$$

Moreover, if we use action models instead of simple public announcements, the inspection action itself may require an infinite action model in the standard DEL framework introduced in Section 1.3, with a separate event for each possible value. Hence public inspection can be viewed as a public announcement of the actual value, but new techniques are required to express it formally. In our simple framework we define knowing and inspecting values as primitive operators.

The main design choices we make is to leave the actual values out of our logical language, thereby avoiding infinite formulas like above.

The notions of knowing and inspecting values have a natural connection with dependencies in databases. This will play a crucial role in the technical development of this section. In particular, our completeness proofs employ the famous set of axioms from [Arm74]. For now, consider the following example.

5.3.1. Example. Suppose a university course has been evaluated using anonymous questionnaires, which besides an assessment for the teacher also asked the students for their main subject. See Table 5.1 for the results.

Student	Subject	Assessment
1	Mathematics	good
2	Mathematics	very good
3	Logic	good
4	Computer Science	bad

Table 5.1: Evaluation Results.

Now suppose a student tells you, the teacher, that his major is Computer Science. Then clearly you know how that student assessed the course, since there is some dependency between the two columns. More precisely, in the cases of students 3 and 4, telling you the value of “Subject” effectively also tells you the value of “Assessment”. In practice, a better questionnaire would only ask for combinations of questions that do not allow the identification of students.

Other examples abound. For instance, the author of [Swe15] gives an account of how easily so-called ‘de-identified data’ produced from medical records could be ‘re-identified’, by matching patient names to publicly available health data.

These examples illustrate that reasoning about knowledge of values in isolation, i.e. separated from knowledge *that*, is both possible and informative. It is such knowledge and its dynamics that we will study here.

5.4 Richer Languages

Our work relates to a collection of papers on epistemic logics with other operators than the standard “knowing that” $K\varphi$. We are particularly interested in the Kv operator expressing that an agent knows the value of a variable. This operator was already mentioned in the seminal work [Pla07] which introduced public announcement logic (PAL). However, a complete axiomatization of PAL together with Kv was only given in [WF13; WF14] using the relativized operator $Kv(\varphi, c)$ for the single and multi-agent cases. It has been shown in [GW16] that by treating the negation of Kv as a primitive diamond-like operator, the logic can be seen as a normal modal logic in disguise with binary modalities.

Inspired by a talk that was partly based on an earlier version of this chapter, Baltag proposed the very expressive Logic of Epistemic Dependency (LED) [Bal16], where knowing that, knowing value, announcing that, announcing value can all be encoded in a general language which also includes equalities like $c = 4$ to facilitate the axiomatization.

In the following sections we go in the other direction: Instead of extending the standard PAL framework with Kv, we study knowing-the-value in isolation, together with its dynamic counterpart $[c]$ for public inspection. In general, the motto of our work here is to see how far one can get in formalizing knowledge and inspection of values without going all the way to or even beyond PAL. In particular, we do not include values in the syntax and do not have any nested epistemic modalities.

As one would expect, our simple language is accompanied by simpler models and the proofs are less complicated than those for existing logics. Still, we consider our Public Inspection Logic (PIL) more than a toy logic. Our completeness proof includes a novel construction which we call "canonical dependency graph" (Definition 5.5.11). We also establish the precise connection between our axioms and the Armstrong axioms widely used in database theory [Arm74].

Table 5.2 shows how PIL fits into the family of existing languages. LED from [Bal16] is the most expressive language. It encodes all the other operators using $K_i^{t_1,\dots,t_n} t$, which expresses that given the current values of t_1 to t_n, agent i knows the value of t. Moreover, to obtain a complete proof system for LED one also needs to include equality and rigid constants in the language. As far as we know, it is an open question to find axiomatizations for languages between PIL and LED, like PIL $+ K$.

Language								References
PAL	p	$K\varphi$				$[!\varphi]\varphi$		[Pla07]
PAL$+Kv$	p	$K\varphi$	$Kv(c)$			$[!\varphi]\varphi$		[Pla07]
PAL$+Kv^r$	p	$K\varphi$	$Kv(c)$	$Kv(\varphi, c)$		$[!\varphi]\varphi$		[WF13; WF14; GW16]
PIL			$Kv(c)$		$[c]\varphi$			Sections 5.5 and 5.6
PIL$+K$		$K\varphi$	$Kv(c)$		$[c]\varphi$			open, see Section 5.7
LED	p	$K\varphi$	$Kv(c)$	$Kv(\varphi, c)$	$[c]\varphi$	$[!\varphi]\varphi$	$c = c$	[Bal16]

Table 5.2: Comparison of Languages.

All languages include the standard boolean operators $\top$, $\neg$ and $\wedge$ which we do not list in Table 5.2. We will discuss other related work in Section 5.7.

5.5 Single-Agent PIL

We first consider a simple single-agent language to talk about knowing and inspecting values. Throughout the rest of this chapter we assume a fixed set of variables $\mathbb{C}$.

5.5.1. DEFINITION. Let c range over $\mathbb{C}$. The language $\mathcal{L}_1$ for *Public Inspection Logic* (PIL) is given by:

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid Kv(c) \mid [c]\varphi$$

Besides standard interpretations of the boolean connectives, the intended meanings are as follows: $Kv(c)$ reads "the agent knows the value of c" and the formula $[c]\varphi$ is meant to say "after revealing the actual value of c, φ is the case". We also use the standard abbreviations $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$ and $\varphi \to \psi := \neg\varphi \vee \psi$.

Notably, the PIL language $\mathcal{L}_1$ is *not* an extension of the standard epistemic language $\mathcal{L}$ from Definition 1.1.1, because it has neither atomic propositions nor $K_i\varphi$ for "knowing that φ". Leaving out the latter is crucial to simplify our framework. We will get back to the more expressive language PIL $+\ K$ in Section 5.7.

5.5.2. DEFINITION. A *PIL model* for $\mathcal{L}_1$ is a tuple $\mathcal{M} = \langle S, \mathcal{D}, V\rangle$ where S is a non-empty set of worlds (also called states), $\mathcal{D}$ is a non-empty domain and V is a valuation $V\colon (S \times \mathbb{C}) \to \mathcal{D}$. To denote $V(s,c) = V(t,c)$, i.e. that c has the same value at s and t according to V, we write $s =_c t$. If this holds for all $c \in C \subseteq \mathbb{C}$ we write $s =_C t$. The semantics are as follows:

$\mathcal{M}, s \vDash \top$		always
$\mathcal{M}, s \vDash \neg\varphi$	$\Leftrightarrow$	$\mathcal{M}, s \nvDash \varphi$
$\mathcal{M}, s \vDash \varphi \wedge \psi$	$\Leftrightarrow$	$\mathcal{M}, s \vDash \varphi$ and $\mathcal{M}, s \vDash \psi$
$\mathcal{M}, s \vDash Kv(c)$	$\Leftrightarrow$	for all $t \in S : s =_c t$
$\mathcal{M}, s \vDash [c]\varphi$	$\Leftrightarrow$	$\mathcal{M}\|^s_c, s \vDash \varphi$

where $\mathcal{M}|^s_c$ is the new model $\langle S', \mathcal{D}, V|_{S' \times \mathbb{C}}\rangle$ based on the new set of states $S' = \{t \in S \mid s =_c t\}$. This is the result of publicly inspecting c at s.

If for a set of formulas Γ and a formula φ we have that whenever a model $\mathcal{M}$ and a state s satisfy $\mathcal{M}, s \vDash \Gamma$ then they also satisfy $\mathcal{M}, s \vDash \varphi$, then we say that φ *follows semantically* from Γ and write $\Gamma \vDash \varphi$. If this holds for $\Gamma = \varnothing$ we say that φ is *semantically valid* and write $\vDash \varphi$.

Note that the actual state s plays an important role in the last clause of our semantics: Public inspection of c at s reveals the *local actual* value of c at s to the agent. The model is restricted to those worlds which agree with s on c. This is different from PAL and other DEL variants based on action models, where updates are usually defined on models directly and not on pointed models.

We employ the usual abbreviation $\langle c\rangle\varphi$ for $\neg[c]\neg\varphi$. Note however, that public inspection of c can always take place and is deterministic. Hence the determinacy axiom $\langle c\rangle\varphi \leftrightarrow [c]\varphi$ is semantically valid and we include it in the following system.

5.5.3. DEFINITION. The proof system $\mathbb{SPIL}_1$ for PIL in the language $\mathcal{L}_1$ consists of the following axiom schemata and rules. If a formula φ is provable from a set of premises Γ, we write $\Gamma \vdash \varphi$. If this holds for $\Gamma = \varnothing$, we write $\vdash \varphi$.

Axiom Schemata		**Rules**	
`TAUT`	all instances of propositional tautologies	`MP`	$\dfrac{\varphi, \varphi \to \psi}{\psi}$
`DIST`	$[c](\varphi \to \psi) \to ([c]\varphi \to [c]\psi)$		
`LEARN`	$[c]Kv(c)$		
`NF`	$Kv(c) \to [d]Kv(c)$	`NEC`	$\dfrac{\varphi}{[c]\varphi}$
`DET`	$\langle c\rangle\varphi \leftrightarrow [c]\varphi$		
`COMM`	$[c][d]\varphi \leftrightarrow [d][c]\varphi$		
`IR`	$Kv(c) \to ([c]\varphi \to \varphi)$		

Intuitively, `LEARN` captures the effect of the inspection; `NF` says that the agent does not forget; `DET` says that inspection is deterministic; `COMM` says that inspections commute; finally, `IR` expresses that inspection does not bring any new information if the value is known already. Note that `DET` says that $[c]$ is a function. It also implies seriality which we list in the following lemma.

5.5.4. LEMMA. *The following schemes are provable in* $\mathbb{SPIL}_1$*:*

- $\langle c\rangle\top$ *(seriality)*
- $Kv(c) \to (\varphi \to [c]\varphi)$ *(`IR'`)*
- $[c](\varphi \wedge \psi) \leftrightarrow [c]\varphi \wedge [c]\psi$ *(`DIST'`)*
- $[c_1]\dots[c_n](\varphi \to \psi) \to ([c_1]\dots[c_n]\varphi \to [c_1]\dots[c_n]\psi)$ *(multi-`DIST`)*
- $[c_1]\dots[c_n](\varphi \wedge \psi) \leftrightarrow [c_1]\dots[c_n]\varphi \wedge [c_1]\dots[c_n]\psi$ *(multi-`DIST'`)*
- $[c_1]\dots[c_n](Kv(c_1) \wedge \dots Kv(c_n))$ *(multi-`LEARN`)*
- $(Kv(c_1) \wedge \dots \wedge Kv(c_n)) \to [d_1]\dots[d_n](Kv(c_1) \wedge \dots \wedge Kv(c_n))$ *(multi-`NF`)*
- $(Kv(c_1) \wedge \dots \wedge Kv(c_n)) \to ([c_1]\dots[c_n]\varphi \to \varphi)$ *(multi-`IR`)*

Moreover, the multi-`NEC` rule is admissible: If $\vdash \varphi$*, then* $\vdash [c_1]\dots[c_n]\varphi$*.*

Proof:
We only prove three of the items and leave the others as an exercise for the reader. For `IR`’, we use `IR`, `DET` and `TAUT`:

$$\dfrac{\dfrac{\dfrac{}{Kv(c) \to ([c]\neg\varphi \to \neg\varphi)}\,(\texttt{IR})}{Kv(c) \to (\neg[c]\varphi \to \neg\varphi)}\,(\texttt{DET})}{Kv(c) \to (\varphi \to [c]\varphi)}\,(\texttt{TAUT})$$

To show multi-`NEC`, we use `DIST`, `NEC` and `TAUT`. For simplicity, consider the case where $C = \{c_1, c_2\}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[c_2](\varphi \to \psi) \to ([c_2]\varphi \to [c_2]\psi)}\,(\texttt{DIST})}{[c_1]([c_2](\varphi \to \psi) \to ([c_2]\varphi \to [c_2]\psi))}\,(\texttt{NEC})}{[c_1][c_2](\varphi \to \psi) \to [c_1]([c_2]\varphi \to [c_2]\psi)}\,(\texttt{DIST, TAUT})}{[c_1][c_2](\varphi \to \psi) \to ([c_1][c_2]\varphi \to [c_1][c_2]\psi)}\,(\texttt{DIST, TAUT})$$

For multi-`LEARN`, we use `LEARN`, `NEC`, `COMM`, `DIST`’ and `TAUT`:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[c_1]Kv(c_1)}\,(\texttt{LEARN})}{[c_2][c_1]Kv(c_1)}\,(\texttt{NEC})}{[c_1][c_2]Kv(c_1)}\,(\texttt{COMM}) \qquad \dfrac{\dfrac{}{[c_2]Kv(c_2)}\,(\texttt{LEARN})}{[c_1][c_2]Kv(c_2)}\,(\texttt{NEC})}{[c_1]([c_2]Kv(c_1) \wedge [c_2]Kv(c_2))}\,(\texttt{DIST', TAUT})}{[c_1][c_2](Kv(c_1) \wedge Kv(c_2))}\,(\texttt{DIST', TAUT})$$

□

5.5.5. Definition. We use the following abbreviations for any two finite sets of variables $C = \{c_1, \ldots, c_m\}$ and $D = \{d_1, \ldots, d_n\}$.

- $Kv(C) := Kv(c_1) \wedge \cdots \wedge Kv(c_m)$
- $[C]\varphi := [c_1] \ldots [c_m]\varphi$
- $Kv(C, D) := [C]Kv(D)$.

Note that by multi-`DIST`’ and `COMM` the exact enumeration of C and D in Definition 5.5.5 do not matter modulo logical equivalence.

In particular, these abbreviations allow us to shorten the “multi” items from Lemma 5.5.4 to $Kv(C, C)$, $Kv(C) \to Kv(D, C)$ and $Kv(C) \to ([C]\varphi \to \varphi)$. The abbreviation $Kv(C, D)$ allows us to define dependencies and it will be crucial in our completeness proof. We have that:

$$\boxed{\mathcal{M}, s \vDash Kv(C, D) \Leftrightarrow \text{for all } t \in S : \text{if } s =_C t \text{ then } s =_D t}$$

5.5.6. DEFINITION. Let $\mathcal{L}_2$ be the language given by:

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid Kv(C, C)$$

Note that the language $\mathcal{L}_2$ is a fragment of $\mathcal{L}_1$, due to the above definition of $Kv(\cdot, \cdot)$ as an abbreviation. In $\mathcal{L}_2$ the dynamic $[c]$ operators can only occur in front of Kv operators or conjunctions thereof. The next lemma might count as a small surprise.

5.5.7. LEMMA. *$\mathcal{L}_1$ and $\mathcal{L}_2$ are equally expressive.*

Proof:
As $Kv(\cdot, \cdot)$ was just defined as an abbreviation, we already know that $\mathcal{L}_1$ is at least as expressive as $\mathcal{L}_2$, i.e. $\mathcal{L}_2 \subseteq \mathcal{L}_1$.

We can also translate in the other direction by pushing all dynamic operators through negations and conjunctions. Formally, let $t\colon \mathcal{L}_1 \to \mathcal{L}_2$ be defined by

$$\begin{array}{lcl}
Kv(d) & \mapsto & Kv(\varnothing, \{d\}) \\
\neg\varphi & \mapsto & \neg t(\varphi) \\
\varphi \wedge \psi & \mapsto & t(\varphi) \wedge t(\psi) \\
[c]\neg\varphi & \mapsto & \neg t([c]\varphi) \\
[c](\varphi \wedge \psi) & \mapsto & t([c]\varphi) \wedge t([c]\psi) \\
[c]\top & \mapsto & \top \\
[c_1] \ldots [c_n] Kv(d) & \mapsto & Kv(\{c_1, \ldots, c_n\}, \{d\})
\end{array}$$

This translation preserves and reflects truth because determinacy and distribution are valid (determinacy allows us to push $[c]$ through negations; distribution to push $[c]$ through conjunctions). Note that we have not yet established completeness, but determinacy is also an axiom. Hence $\varphi \leftrightarrow t(\varphi)$ is provable and the translation t preserves and reflects provability and consistency. □

5.5.8. EXAMPLE. The translation of formulas of the form $[c]\varphi$ depends on the top connective within φ. For example, we have

$$\begin{array}{rcl}
t([c](\neg Kv(d) \wedge [e]Kv(f))) & = & t([c]\neg Kv(d)) \wedge t([c][e]Kv(f)) \\
& = & \neg Kv(\{c\}, \{d\}) \wedge Kv(\{c, e\}, \{f\})
\end{array}$$

The language $\mathcal{L}_2$ allows us to connect PIL to the maybe most famous axioms in database theory and dependence logic, from [Arm74].[1]

[1] It is baffling that this classic paper from 1974 with more than 1200 citations was still not available online in 2018. This probably helped to create quite some confusion and disagreement on what exactly *the* Armstrong axioms are. Here we follow the original paper which first uses the rather technical axioms F1 to F4 to prove the main characterization result, but later (in Section 9) argues that the axioms DC1, DC3 and DC4 are sufficient. These three are projectivity, transitivity and additivity which we use here.

5.5.9. LEMMA. *Armstrong's axioms are semantically valid and derivable in* $\mathbb{SPIL}_1$*:*

- $Kv(C, D)$ *for any* $D \subseteq C$ *(projectivity)*
- $Kv(C, D) \wedge Kv(D, E) \to Kv(C, E)$ *(transitivity)*
- $Kv(C, D) \wedge Kv(C, E) \to Kv(C, D \cup E)$ *(additivity)*

Proof:
The semantic validity is easy to check, hence we focus on the derivations.

For projectivity, take any two finite sets C and D such that $D \subseteq C$. If $D = C$, then we only need a derivation like the following, which basically generalizes learning to finite sets.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[c_1]Kv(c_1)}\,(\texttt{LEARN})}{[c_2][c_1]Kv(c_1)}\,(\texttt{NEC})}{[c_1][c_2]Kv(c_1)}\,(\texttt{COMM}) \qquad \dfrac{\dfrac{}{[c_2]Kv(c_2)}\,(\texttt{LEARN})}{[c_1][c_2]Kv(c_1)}\,(\texttt{NEC})}{[c_1]([c_2]Kv(c_1) \wedge [c_2]Kv(c_2))}\,(\texttt{DIST})}{[c_1][c_2](Kv(c_1) \wedge Kv(c_2))}\,(\texttt{DIST})$$

If $D \subsetneq C$, continue by applying NEC for all elements of $C \setminus D$ to get $Kv(C, D)$.

Transitivity follows from IR and NF as follows. For simplicity, we first only consider the case where C, D and E are singletons.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Kv(e) \to [c]Kv(e)}\,(\texttt{NF})}{[d](Kv(e) \to [c]Kv(e))}\,(\texttt{NEC})}{[d]Kv(e) \to [d][c]Kv(e)}\,(\texttt{DIST})}{[d]Kv(e) \to [c][d]Kv(e)}\,(\texttt{COMM}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{Kv(d) \to ([d]Kv(e) \to Kv(e))}\,(\texttt{IR})}{[c](Kv(d) \to ([d]Kv(e) \to Kv(e)))}\,(\texttt{NEC})}{[c]Kv(d) \to [c]([d]Kv(e) \to Kv(e))}\,(\texttt{DIST})}{[c]Kv(d) \to ([c][d]Kv(e) \to [c]Kv(e))}\,(\texttt{DIST})}{[c]Kv(d) \to ([d]Kv(e) \to [c]Kv(e))}\,(\texttt{TAUT})$$

Now consider any three finite sets of variables C, D and E. Using the abbreviations from Definition 5.5.5 and the "multi" rules given in Lemma 5.5.4, it is easy to generalize the proof. In fact, the proof is exactly the same with capital letters.

Similarly, additivity follows immediately from multi-DIST'. □

We can now use Armstrong's axioms to prove completeness of our logic. The crucial idea is a new definition of a canonical dependency graph.

5.5.10. THEOREM (Strong Completeness). *For all sets of formulas* $\Delta \subseteq \mathcal{L}_1$ *and all formulas* $\varphi \in \mathcal{L}_1$*, if* $\Delta \models \varphi$*, then also* $\Delta \vdash \varphi$*.*

Proof:
By contraposition using a canonical model. Suppose $\Delta \nvdash \varphi$. Then $\Delta \cup \{\neg\varphi\}$ is consistent and there is a maximally consistent set $\Gamma \subseteq \mathcal{L}_1$ such that $\Gamma \supseteq (\Delta \cup \{\neg\varphi\})$. We now build a model $\mathcal{M}_\Gamma$ in which worlds are subsets of $\mathbb{C}$ and the value of each $c \in \mathbb{C}$ at world w reflects whether we have $c \in w$. Then we use the full set $\mathbb{C}$ as the actual world, so that all non-actual worlds v are the set of variables that at v have the actual value. We can then show $\mathcal{M}_\Gamma, \mathbb{C} \vDash \Gamma$, which implies $\Delta \nvDash \varphi$.

5.5.11. Definition. Given Γ, we define the *canonical graph* $G_\Gamma := (\mathcal{P}(\mathbb{C}), \to)$ where $A \to B$ iff $Kv(A, B) \in \Gamma$. By Lemma 5.5.9 this graph has properties corresponding to the Armstrong axioms: projectivity, transitivity and additivity. We call a set of variables $s \subseteq \mathbb{C}$ *closed* under G_Γ iff whenever $A \subseteq s$ and $A \to B$ in G_Γ, then also $B \subseteq s$. We define the *canonical model* as $\mathcal{M}_\Gamma := (S, \mathcal{D}, V)$ where

- $S := \{s \subseteq \mathbb{C} \mid s \text{ is closed under } G_\Gamma\}$
- $\mathcal{D} := \{0, 1\}$
- $V(s, c) = \begin{cases} 0 & \text{if } c \in s \\ 1 & \text{otherwise} \end{cases}$

Note that our domain is just $\{0, 1\}$. This is possible because we do not have to find a model where the dependencies hold globally. Instead, $Kv(C, d)$ only says that given the values of all elements of C at the *actual* world, also the values of d are the same at all other worlds. The dependency does not have to hold between two non-actual worlds. This distinguishes our models from relationships as discussed in [Arm74] where no actual world or state is used, see Example 5.5.16 below.

We can now state and prove our Truth Lemma. Before doing so, let us emphasize two peculiarities: First, the states in our canonical model are not maximally consistent sets of formulas but sets of variables. Second, we only claim the Truth Lemma at one specific state, namely at $\mathbb{C}$ where all variables have value 0. As our language does not include nested epistemic modalities, we actually never evaluate formulas at other states of the canonical model.

5.5.12. Lemma (Truth Lemma). *$\mathcal{M}_\Gamma, \mathbb{C} \vDash \varphi$ iff $\varphi \in \Gamma$.*

Proof:
It suffices to show this for all φ in $\mathcal{L}_2$: Given some $\varphi \in \mathcal{L}_1$, by Lemma 5.5.7 we have that $\mathcal{M}_\Gamma, \mathbb{C} \vDash \varphi \iff \mathcal{M}_\Gamma, \mathbb{C} \vDash t(\varphi)$ because the translation preserves and reflects truth. Moreover, we have $\varphi \in \Gamma \iff t(\varphi) \in \Gamma$, because $\varphi \leftrightarrow t(\varphi)$ is provable in $\mathbb{SPIL}_1$. Hence it suffices to show that $\mathcal{M}_\Gamma, \mathbb{C} \vDash t(\varphi)$ iff $t(\varphi) \in \Gamma$, i.e. to show the Truth Lemma for $\mathcal{L}_2$. Again, negation and conjunction are standard — the crucial case are dependencies.

Suppose $Kv(C, D) \in \Gamma$. By definition $C \to D$ in G_Γ. To show $\mathcal{M}_\Gamma, \mathbb{C} \vDash Kv(C, D)$, take any t such that $\mathbb{C} =_C t$ in $\mathcal{M}_\Gamma$. Then by definition of V we have $C \subseteq t$. As t is closed under G_Γ, this implies $D \subseteq t$. Now by definition of V we have $\mathbb{C} =_D t$.

For the converse, suppose $Kv(C, D) \notin \Gamma$. Then by definition $C \not\to D$ in G_Γ. Now we define the set $t := \{c' \in \mathbb{C} \mid C \to \{c'\} \text{ in } G_\Gamma\}$. This gives us $C \subseteq t$. But we also have $D \not\subseteq t$ because otherwise additivity would imply $C \to D$ in G_Γ. Moreover, because G_Γ is transitive it is enough to "go one step" in G_Γ to get a set that is closed under G_Γ. This means that t is closed under G_Γ and therefore a state in our model, i.e. we have $t \in S$. Now by definition of V and projectivity, we have $\mathbb{C} =_C t$ but $\mathbb{C} \neq_D t$. Thus t is a witness for $\mathcal{M}_\Gamma, \mathbb{C} \nvDash Kv(C, D)$. □

This also finishes the completeness proof. Note that we used all three properties corresponding to the Armstrong axioms. □

5.5.13. Example. To illustrate the idea of the canonical dependency graph, let us study a concrete example of what the graph and model look like. Consider a maximally consistent set $\Gamma \supseteq \{\neg Kv(c), \neg Kv(d), Kv(e), Kv(c, d), \dots\}$.

The interesting part of the canonical graph G_Γ then looks as follows, where the nodes are subsets of $\{c, d, e\}$. For clarity we only draw $\to \cap \not\supseteq$, i.e. we omit edges given by inverse inclusions. For example, all nodes will also have an edge going to the $\varnothing$ node.

$$\begin{tikzcd}
\{e\} & \{d,e\} & \{e,c\} \arrow[r] & \{c,d,e\} \\
\varnothing \arrow[u] & \{d\} \arrow[u] & \{c\} \arrow[l] \arrow[ul] \arrow[u] \arrow[ur] \arrow[r] & \{c,d\} \arrow[u]
\end{tikzcd}$$

To get a model out of this graph, note that there are exactly three subsets of $\mathbb{C}$ closed under following the edges. Namely, let $S := \{s := \{e\}, t := \{d, e\}, u := \{c, d, e\}\}$ and use the binary valuation which says that a variable has value 0 iff it is an element of the state. It is then easy to check that $\mathcal{M}, u \vDash \Gamma$.

	s	t	u
c	1	1	0
d	1	0	0
e	0	0	0

It is straightforward to define an appropriate notion of bisimulation for our logic and to obtain the usual characterization results for it.

5.5.14. DEFINITION. Two pointed PIL models $((S, \mathcal{D}, V), s)$ and $((S', \mathcal{D}', V'), s')$, are *bisimilar* iff the following two conditions are fulfilled:

(i) Forth: For all finite $C \subseteq \mathbb{C}$ and all $d \in \mathbb{C}$: If there is a $t \in S$ such that $s =_C t$ and $s \neq_d t$, then there is a $t' \in S'$ such that $s' =_C t'$ and $s' \neq_d t'$.

(ii) Back: For all finite $C \subseteq \mathbb{C}$ and all $d \in \mathbb{C}$: If there is a $t' \in S'$ such that $s' =_C t'$ and $s' \neq_d t'$, then there is a $t \in S$ such that $s =_C t$ and $s \neq_d t$.

Note that we do not need the bisimulation to link non-actual worlds. This is because all formulas are evaluated at the same world. In fact, the following characterization theorem only holds because we do not link non-actual worlds.

5.5.15. THEOREM. *Two pointed PIL models satisfy the same formulas iff they are bisimilar.*

Proof:
By Lemma 5.5.7 we only have to consider formulas of $\mathcal{L}_2$. Moreover, it suffices to consider formulas $Kv(C, d)$ with a singleton in the second set, because $Kv(C, D)$ is equivalent to $\bigwedge_{d \in D} Kv(C, d)$. Then it is straightforward to show that if $\mathcal{M}, s$ and $\mathcal{M}', s'$ are bisimilar then $\mathcal{M}, s \vDash \neg Kv(C, d) \iff \mathcal{M}', s' \vDash \neg Kv(C, d)$ by definition of our bisimulation. The other way around is also obvious since the two conditions for bisimulation are based on the semantics of $\neg Kv(C, d)$. $\square$

Note that a bisimulation characterization for a language without the dynamic operator $[c]$ can be obtained by restricting Definition 5.5.14 to $C = \varnothing$. We leave it as an exercise for the reader to use this and Theorem 5.5.15 to show that $[c]$ is not reducible, which distinguishes it from $[!\varphi]$ in PAL.

5.5.16. EXAMPLE (Pointed Models Make a Difference). It seems that the following theorem of our logic does not translate to Armstrong's system from [Arm74].

$$[c](Kv(d) \vee Kv(e)) \leftrightarrow ([c]Kv(d) \vee [c]Kv(e))$$

First, to see that this is provable, note that it follows from determinacy and seriality. Second, it is valid because we consider pointed models which convey more information than a simple list of possible values. Consider the following table which represents four possible worlds.

	w_1	w_2	w_3	w_4
c	1	1	2	2
d	1	1	2	3
e	3	2	1	1

Here we would say that "After learning c we know d or we know e.", i.e. the antecedent of above formula holds. However, the consequent only holds if we evaluate formulas while pointing at a specific world/row: It is globally true that given c we will learn d or that given c we will learn e. But none of the two disjuncts holds globally, which would be needed for a dependency in Armstrong's sense. Note that this is more a matter of expressiveness than of logical strength. In Armstrong's system there is just no way to express $[c](Kv(d) \vee Kv(e))$.

5.6 Multi-Agent PIL

We now generalize Public Inspection Logic to multiple agents. In the language, we use Kv_i to say that agent i knows the value of c and in the models, we add an accessibility relation for each agent to describe their knowledge. To obtain a complete proof system, we can leave most axioms as above but have to restrict the irrelevance axiom. Again the completeness proof uses a canonical model construction and a truth lemma for a restricted but equally expressive syntax. The only change is that we now define a dependency graph for each agent in order to define accessibility relations instead of restricted sets of worlds.

5.6.1. DEFINITION (Multi-Agent PIL). We fix a non-empty set of agents I. The language $\mathcal{L}_1^I$ of *multi-agent Public Inspection Logic* (PIL) is given by

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid Kv_i c \mid [c]\varphi$$

where $i \in I$. We interpret it on models $\langle S, \mathcal{D}, V, R \rangle$ where S, $\mathcal{D}$ and V are as before and R assigns to each agent i an equivalence relation $\sim_i$ over S. The semantics are standard for the boolean operators and as follows for Kv_i and $[c]$:

$$\begin{array}{lcl} \mathcal{M}, s \vDash Kv_i c & :\Longleftrightarrow & \forall t \in S : s \sim_i t \Rightarrow s =_c t \\ \mathcal{M}, s \vDash [c]\varphi & :\Longleftrightarrow & \mathcal{M}|_c^s, s \vDash \varphi \end{array}$$

where $\mathcal{M}|_c^s$ is the new model $\langle S', \mathcal{D}, V|_{S' \times \mathbb{C}}, R|_{S' \times S'} \rangle$ with $S' = \{t \in S \mid s =_c t\}$.

Analogous to Definition 5.5.5 we define the following abbreviation to express dependencies known by agent i and note its semantics:

$$Kv_i(C, D) := [c_1] \dots [c_n](Kv_i(d_1) \wedge \dots \wedge Kv_i(d_m))$$

$$\mathcal{M}, s \vDash Kv_i(C, D) \Leftrightarrow \text{for all } t \in S : \text{if } s \sim_i t \text{ and } s =_C t \text{ then } s =_D t$$

The proof system $\mathbb{SPIL}$ for PIL in the language $\mathcal{L}_1^I$ is obtained by replacing each Kv in the axioms of $\mathbb{SPIL}_1$ by Kv_i, and replacing `IR` by the following restricted version:

`RIR` $Kv_i c \to ([c]\varphi \to \varphi)$ for any φ not mentioning any agent besides i

Before summarizing the completeness proof for the multi-agent setting, let us highlight some details of this definition. As before, the actual state s plays an important role in the semantics of $[c]$. To make the update less local we can use an alternative but equivalent definition: Instead of deleting states, only delete the $\sim_i$ links between states that disagree on the value of c. Then the update no longer depends on the actual state.

For traditional reasons we define $\sim_i$ to be an equivalence relation. This is not necessary, because our language cannot tell whether the relations are actually equivalences. Removing the constraint and extending the class of models would thus not make any difference.

For the proof system, the original irrelevance axiom `IR` is *not* valid in the multi-agent setting, because φ might talk about agents for which $[c]$ does matter.

5.6.2. THEOREM (Strong Completeness for $\mathbb{SPIL}$). *For all sets of formulas $\Delta \subseteq \mathcal{L}_1^I$ and all formulas $\varphi \in \mathcal{L}_1^I$, if $\Delta \vDash \varphi$, then also $\Delta \vdash \varphi$.*

Proof:
By the same methods as for Theorem 5.5.10. Given a maximally consistent set $\Gamma \subseteq \mathcal{L}_1^I$, we want to build a model $\mathcal{M}_\Gamma$ such that for the world $\mathbb{C}$ in that model we have $\mathcal{M}_\Gamma, \mathbb{C} \vDash \Gamma$.

First, for each agent $i \in I$, let G_Γ^i be the graph given by $A \to_i B \quad :\iff \quad \Gamma \vdash Kv_i(A, B)$. Given that $\mathbb{SPIL}$ was obtained by indexing the axioms of $\mathbb{SPIL}_1$, it is easy to check that indexed versions of all the Armstrong axioms are provable and therefore all the graphs G_Γ^i for $i \in I$ will have the corresponding properties. In particular, `RIR` suffices for this.

Second, define the canonical model $\mathcal{M}_\Gamma := (S, \mathcal{D}, V, R)$ where $S := \mathcal{P}(\mathbb{C})$, $\mathcal{D} := \{0, 1\}$, $V(s, c) := 0$ if $c \in s$ and $V(s, c) := 1$ otherwise, and $s \sim_i t$ iff s and t are either both closed or both not closed under G_Γ^i.

5.6.3. LEMMA (Multi-Agent Truth Lemma). $\mathcal{M}_\Gamma, \mathbb{C} \vDash \varphi$ *iff* $\varphi \in \Gamma$.

Proof:
Again it suffices to consider a restricted language and we proceed by induction on φ. The crucial case is when φ is of the form $Kv_i(C, D)$.

Suppose $Kv_i(C, D) \in \Gamma$. Then by definition $C \to D$ in G_Γ^i. To show $\mathcal{M}_\Gamma, \mathbb{C} \vDash Kv_i(C, D)$, take any t such that $\mathbb{C} \sim_i t$ and $\mathbb{C} =_C t$ in $\mathcal{M}_\Gamma$. Then by definition of V we have $C \subseteq t$. Moreover, $\mathbb{C}$ is closed under G_Γ^i. Hence by definition of $\sim_i$, also t must be closed under G_Γ^i, which implies $D \subseteq t$. Now by definition of V we have $\mathbb{C} =_D t$.

For the converse, suppose $Kv_i(C, D) \notin \Gamma$. Then by definition $C \not\to D$ in G_Γ^i. Now, let $t := \{c' \in \mathbb{C} \mid C \to \{c'\} \text{ in } G_\Gamma^i\}$. This gives us $C \subseteq t$. But we also have $D \not\subseteq t$, because otherwise additivity would imply $C \to D$ in G_Γ^i. Moreover, because G_Γ^i is transitive, it is enough to "go one step" in G_Γ^i to get a set that is closed under G_Γ^i. This means that t is closed under G_Γ^i and therefore, by definition of $\sim_i$, we have $\mathbb{C} \sim_i t$. Moreover, by definition of V and using projectivity, we have $\mathbb{C} =_C t$ but $\mathbb{C} \neq_D t$. Thus t is a witness for $\mathcal{M}_\Gamma, \mathbb{C} \not\models Kv_i(C, D)$. □

Again the Truth Lemma also finishes the completeness proof. □

5.6.4. Example. Analogously to Example 5.5.13, the following illustrates the multi-agent version of our canonical model construction. Consider a maximally consistent set Γ extending this set:

$$\{\neg Kv_1(c), \neg Kv_1(d), Kv_1(c,d), \neg Kv_1(d,c), \neg Kv_2(c), \neg Kv_2(d), \neg Kv_2(c,d), Kv_2(d,c)\}$$

Note that agents 1 and 2 do not differ in which values they know right now, but there is a difference in what they will learn from inspections of c and d.

Two canonical dependency graphs generated from Γ are shown in Figure 5.3. Again, for clarity we omit superset edges. The subsets of $\mathbb{C} = \{c, d\}$ closed under the graphs are thus $\{\{c,d\}, \{d\}, \varnothing\}$ and $\{\{c,d\}, \{c\}, \varnothing\}$ for agent 1 and 2 respectively, inducing the equivalence relations as shown in Figure 5.3.

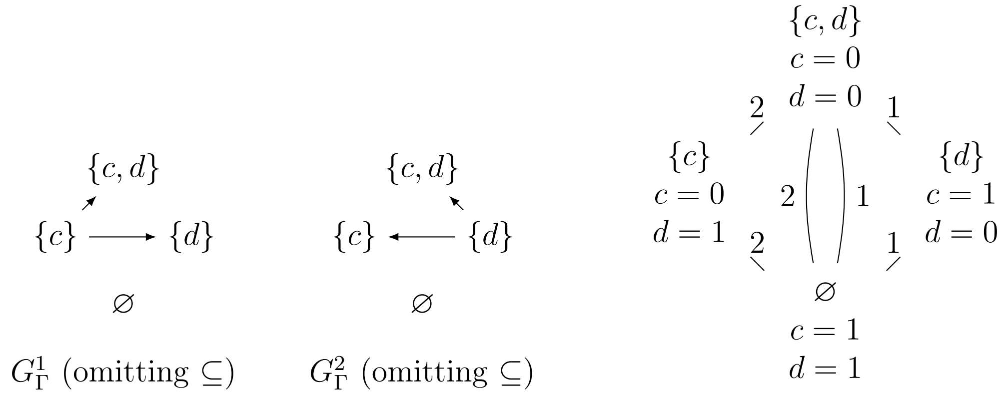


Figure 5.3: Two canonical dependency graphs and the resulting canonical model.

Just like for $\mathbb{SPIL}_1$ we can also give a notion of bisimulation for $\mathbb{SPIL}$.

5.6.5. Definition. Two pointed models $((S, \mathcal{D}, V, R), s)$ and $((S', \mathcal{D}', V', R'), s')$ for multi-agent PIL are *bisimilar* iff the following two conditions are fulfilled:

(i) Forth: For all finite $C \subseteq \mathbb{C}$, all $d \in \mathbb{C}$ and all agents i: If there is a $t \in S$ such that $s \sim_i t$ and $s =_C t$ and $s \neq_d t$, then there is a $t' \in S'$ such that $s' \sim_i t'$ and $s =_C t$ and $s' \neq_d t'$.

(ii) Back: For all finite $C \subseteq \mathbb{C}$, all $d \in \mathbb{C}$ and all agents i: If there is a $t' \in S'$ such that $s' \sim_i t'$ and $s' =_C t'$ and $s' \neq_d t'$, then there is a $t \in S$ such that $s \sim_i t$ and $s' =_C t'$ and $s \neq_d t$.

5.6.6. THEOREM. *Two pointed multi-agent models satisfy the same formulas of the multi-agent language $\mathcal{L}_1^I$ iff they are bisimilar.*[2]

Proof:
Similar to the proof of Theorem 5.5.15. First, note that we can again consider a restricted but equally expressive language with atoms of the form $Kv_i(C,d)$. Second, it is easy to check if $\mathcal{M}, s$ and $\mathcal{M}', s'$ are bisimilar, then we have $\mathcal{M}, s \vDash \neg Kv_i(C,d) \iff \mathcal{M}', s' \vDash \neg Kv_i(C,d)$ for all i. The converse also holds, because the forth and back conditions are now based on the semantics of $\neg Kv_i(C,d)$. □

Finally, we can show what our running example for this chapter looks like as a model for Public Inspection Logic.

5.6.7. EXAMPLE. Example 5.0.1 looks almost the same as the Kripke model from Figure 5.1 above if we model it in PIL in the obvious way, with the same number of worlds. But there is a much smaller model which is equivalent from the perspective of PIL, i.e. it satisfies exactly the same formulas.

We show it in Figure 5.4 and note the similarity to the register model in Figure 5.2 which also needed only four worlds. But in contrast to the register models, in PIL the values and the range of the variables do not matter, because the language never refers to them. Replacing both 5 and 7 with 1 in Figure 5.4 would not change the truth value of any PIL formula.

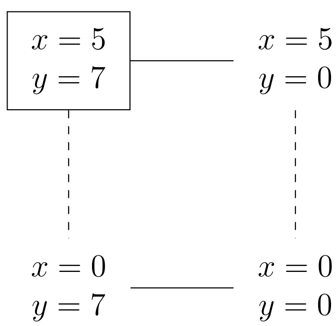


Figure 5.4: PIL model for Example 5.0.1.

[2]Unfortunately, in the original paper [EGW17] which this chapter is based on, the definition of multi-agent bisimulation contains an error. In [EGW17, Definition 9, page 88] we also linked non-actual worlds. This is too restrictive, because multi-agent PIL does not have any nested modalities. Definition 5.6.5 here is the corrected version.

5.7 Conclusion and Future Work

We compared three different ways to model the knowledge of numeric variables, starting with binary encodings, then treating register models, and finally focusing on a new logic for knowing values and public inspection. Table 5.3 gives a comparison of the three languages we discussed in this chapter. For each language we show whether and how certain statements can be expressed. It is clear that Public Inspection Logic uses the most succinct but also the least expressive language. The language for register models still provides succinct formulas and can also refer to concrete values. For Kripke models with binary encodings we use the language of epistemic logic with public announcements, $\mathcal{L}_P$ from Definition 1.2.1, which leads to rather long formulas — especially if "whether" is spelled out.

Statement	Binary Encoding	Register	PIL
x has value 5	$\neg p_2^x \land p_1^x \land \neg p_0^x$	$x = 5$	n/a
x and y are equal	$\bigwedge_i (p_i^x \leftrightarrow p_i^y)$	$x = y$	n/a
a knows the value of x	$K_a^? p_2^x \land K_a^? p_1^x \land K_a^? p_0^x$	$K_a x$	$Kv_a x$
given x, a knows y	$[!?p_0^x] \dots [!?p_2^x](\bigwedge_j K_a^? p_j^y)$	$[!x]K_a y$	$[x]y$ or $Kv_a(x, y)$

Table 5.3: Three ways to model numeric knowledge.

Besides syntax we can also compare the semantics of the three logics. First, note that the models for the situation with Alice and Bob from Example 5.0.1 differ in size. The Kripke model with binary encoding in Example 5.1.3 has 64 worlds and the equivalent knowledge structure only has length 47 if seen as a simple string. The register model in Example 5.2.4 has 4 worlds, but there are 64 agreeing assignments which might have to be generated when a formula is checked on the model. The PIL model from Example 5.6.7 is the smallest, with only 4 worlds and no additional memory needs during the checking of formulas.

More relevant in practice is how the models grow for larger numbers. Suppose that instead of $\{0, \dots, 7\}$, we would use a different range with M different values. The binary encoding then needs $\mathcal{O}(M^2)$ worlds whereas the register model still only needs 4 worlds, though a tiny amount of additional memory might be needed to represent larger bounds. The number of agreeing assignments is $\mathcal{O}(M^2)$. Clearly beating those representations, the PIL model has a constant size of 4 worlds. In fact, the same model represents all variations of the example with a bigger range.

Does this mean that PIL is the best representation? Only if it is expressive enough for the particular use case. In settings where "who knows what" and dependencies between variables are all we want to check, PIL is an optimal abstraction method. But as soon as we need more expressivity or relations between propositional and numeric knowledge are of interest, PIL will no longer be expressive enough.

Between our specific approach and the general language of [Bal16], a lot can still be explored. An advantage of having a weaker language with explicit operators, instead of encoding them in a more general language, is that we can clearly see the properties of those operators showing up as intuitive axioms.

The framework of PIL can be extended in different directions. We could for example add equalities $c = d$ to the language, together with knowledge $K(c = d)$ and announcement $[c = d]$. No changes to the models are needed, but axiomatizing these operators seems not straightforward. Alternatively, just like Plaza added Kv to PAL in [Pla07], we can also add K to PIL. The next language to be studied is thus PIL + K from Table 5.2 above, and given by

$$\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid Kv_i c \mid K_i\varphi \mid [c]\varphi.$$

While the language from [Bal16] also includes this language, to our knowledge, it is an open question whether and how PIL + K without any further additions can be axiomatized.

It is easy to check that the standard axioms for multi-agent $\mathsf{S5}$ are sound on multi-agent PIL models from Definition 5.6.1. We can also add introspection axioms for the interplay between Kv and K. That is, $Kv_i c \to K_i Kv_i c$ and $\neg Kv_i c \to K_i Kv_i c$ are both valid. But we do not know which other axioms might have to be added for a complete axiomatization. In particular, the simple proof via Armstrong axioms no longer works for PIL + K.

We note that PIL + K can also express *knowledge of* dependency in contrast to *de facto* dependency. For example, $K_i[c]Kv_i d$ expresses that agent i knows that d functionally depends on c, while $[c]Kv_i d$ expresses that the value of d (given the information state of i) is determined by the *actual value* of c *de facto*. In particular the latter does not imply that i knows this. The agent can still consider other values of c possible that would not determine the value of d. To see the difference technically, we can spell out the truth condition for $K_i[c]Kv_i(d)$ under standard Kripke semantics for K_i on S5 models:

$$\mathcal{M}, s \vDash K_i[c]Kv_i(d) \iff \text{for all } t_1 \sim_i s,\ t_2 \sim_i s : t_1 =_c t_2 \text{ implies } t_1 =_d t_2$$

Now consider Example 5.5.16: $[c]Kv(d)$ holds in the first row, but $K[c]Kv(d)$ does not hold since the semantics of K require $[c]Kv(d)$ to hold at *all* worlds considered possible by the agent. This also shows that $[c]Kv(d)$ is not positively introspective (i.e. the formula $[c]Kv(d) \to K_i[c]Kv(d)$ is not valid), and it is essentially not a subjective epistemic formula.

In this way, $K[c]Kv(d)$ can also be viewed as the atomic formula $=(c, d)$ in *dependence logic* (DL) from [Vää07]. A *team model* of DL can be viewed as the set of epistemically accessible worlds, i.e., a single-agent model in our case. The connection with dependence logic also brings PIL closer to the first-order variant of *epistemic inquisitive logic* by [CR15], where knowledge of entailment of interrogatives is the knowledge of dependency. For a detailed comparison with our approach, see [Cia16, Section 6.7.4].

Another approach is to make the dependency more explicit and include functions in the syntax. In [Din16] a functional dependency operator $\mathcal{K}f_i$ is added to the epistemic language with Kv_i operators: $\mathcal{K}f_i(c,d) := \exists f K_i(d = f(c))$ where f ranges over a pool of functions.

Finally, there is an independent but related line of work on (in)dependency of variables using predicates, see for example [MN10; Nau12; NN14; HN16]. In particular [NN14] also uses a notion of dependency as the epistemic implication "Knowing c implies knowing d.", similar to our formula $Kv(c,d)$. A "dependency graph" is also used in [HN16] to describe how different variables, in this case payoff functions in strategic games, may depend on each other. Note however, that these graphs are not the same as our canonical dependency graphs from Definition 5.5.11. Our graphs are directed and describe determination between sets of variables. In contrast, the graphs in [HN16] are undirected and consist of singleton nodes for each player in a game. We leave a more detailed comparison for another occasion.

Chapter 6

Dynamic Gossip

> Liaisons were supposed to be announced when
> they were formed and when they were dissolved.
> It was a way to curtail gossip and intrigue,
> which could so easily run rampant in a math.
>
> Neal Stephenson: *Anathem*

The so-called *gossip problem* is a problem about peer-to-peer information sharing: A number of agents each start with some private information, and the goal is to share this information among all agents, using only peer-to-peer communication channels [Tij71]. For example, the agents could be autonomous sensors that need to pool their individual observations in order to obtain a composite group observation. Or the agents could be distributed copies of a database that can each be edited separately, and that need to synchronize with each other [Eug+04; Hae+16; Irv16].

The example that is typically used in the literature, however, is a bit more frivolous: as the name suggests, the gossip problem is usually represented as a number of people *gossiping* [HHL88; Dit+15; Dit+17]. This term goes back to the oldest sources on the topic, such as [BS72]. The gossip scenario gives us not only the name of the gossip problem, but also the names of some of the other concepts that are used: the private information that an agent starts out with is called that agent's *secret*, the communication between two agents is called a *telephone call* and an agent a is capable of contacting another agent b if *a knows b's telephone number*.

These terms should not be taken too literally. Results on the gossip problem can, in theory, be used by people that literally just want to exchange gossip by telephone. But we model information exchange in general and ignore all other social and fun aspects of gossip among humans — though they also can be modeled in epistemic logic [Kle17].

For our framework, applications where artificial agents need to synchronize their information are much more likely. For example, recent ideas to improve cryptocurrencies like bitcoin and other blockchain applications focus on the peer-to-peer exchange (gossip) happening in such networks [SLZ16] or even aim to replace blockchains with directed graphs storing the history of communication [Bai17]. Epistemic logic can shed new light on the knowledge of agents participating in blockchain protocols [HP17; BFS17].

There are many different sets of rules for the gossip problem [HHL88]. For example, calls may be one-on-one, or may be conference calls. Multiple calls may take place in parallel, or must happen sequentially. Agents may only be allowed to exchange one secret per call, or exchange everything they know. Information may go both ways during a call, or only in one direction. We consider only the most commonly studied set of rules: calls are one-on-one, calls are sequential, and the callers exchange all the secrets they know. So if a call between a and b is followed by a call between b and c, then in the second call agent b will also tell agent c the secret of agent a.

Our goal is to ensure that every agent knows every secret. An agent who knows all secrets is called an *expert*, so the goal is to turn all agents into experts.

The *classical* gossip problem, studied in the 1970s, assumed a total communication network (anyone could call anyone else from the start), and focused on optimal call sequences, i.e. schedules of calls which spread all the secrets with a minimum number of calls, which happens to be $2n - 4$ for $n \geq 4$ agents [Tij71; Hur00]. Later, this strong assumption on the network of the gossiping agents was dropped, giving rise to studies on different network topologies (see [HHL88] for a survey), with $2n - 3$ calls sufficing for most networks.

Unfortunately, these results about optimal call sequences only show that such call sequences exist. They do not provide any guidance to the agents about how to achieve an optimal call sequence. Effectively, these solutions assume a central scheduler with knowledge of the entire network, who will come up with an optimal schedule of calls, to be sent to the agents, who will eventually execute it in the correct order. Most results also rely upon some notion of synchronicity so that agents can execute their calls at the appropriate time (i.e. after some calls have been made, and before some other calls are made).

The requirement that there be a central scheduler that tells the agents exactly what to do is against the spirit of the peer-to-peer communication that we want to achieve. Computer science has shifted towards the study of *distributed algorithms* for the gossip problem [HLL99; Kar+00]. Indeed, the gossip problem becomes more natural without a central scheduler; the gossiping agents try to do their best with the information they have when deciding whom to call. Unfortunately, this can lead to sequences of calls that are redundant because they contain many calls that are uninformative in the sense that neither agent learns a new secret. Additionally, the algorithm may fail, i.e., it may deadlock, get stuck in a loop or terminate before all information has been exchanged.

For many applications it is not realistic to assume that every agent is capable of contacting every other agent. So we assume that every agent has a set of agents of which they "know the telephone number", their neighbors, so to say, and that they are therefore able to contact. We represent this as a directed graph, with an edge from agent a to agent b if a is capable of calling b.

In classical studies, this graph is typically considered to be unchanging. In more recent work on *dynamic gossip* the agents exchange both the secrets and the numbers of their contacts, therefore increasing the connectivity of the network [Dit+15]. We focus on dynamic gossip. In distributed protocols for dynamic gossip each agent decides on their own whom to call, depending on their current information [Dit+15], or also depending on the expectation for knowledge growth resulting from the call [Dit+17]. The latter requires agents to represent each other's knowledge, and thus epistemic logic.

Different protocols for dynamic gossip are successful in different classes of gossip networks. The main challenge in designing such a protocol is to find a good level of redundancy: we do not want superfluous calls, but the less redundant a gossip protocol, the easier it fails in particular networks. Another challenge is to keep the protocol simple. After all, a protocol that requires the agents to solve a computationally hard problem every time they have to decide whom to call next would not be practical. There is also a trade-off between the content of the message of which a call consists and the expected duration of gossip protocols. A nice example of that is [HM17], wherein the complexity of gossip protocol termination is reduced from $n \log n$ to linear n, however at the price of more 'expensive' messages, not only exchanging secrets but also knowledge about secrets.

A well-studied protocol is "Learn New Secrets" (LNS), in which agents are allowed to call someone if and only if they do not know the other's secret. This protocol excludes redundant calls in which neither participant learns any new secrets. As a result of this property, all LNS call sequences are finite. For small numbers of agents, it therefore has a shorter expected execution length than the "Any Call" (ANY) protocol that allows arbitrary calls at all times and thus allows infinite call sequences [DKS17]. Additionally, it is easy for agents to check whom they are allowed to call when following LNS. However, LNS is not always successful. On some graphs it can terminate unsuccessfully, i.e. when some agents do not yet know all secrets. In particular there are graphs where the outcome depends on how the agents choose among allowed calls [Dit+15].

Fortunately, it turns out that failure of LNS can often be avoided with some forethought by the calling agents. That is, if some of the choices available to the agents lead to success and other choices to failure, it is often possible for the agents to determine in advance which choices are the successful ones. This leads to the idea of *strengthening* a protocol. Suppose that P is a protocol that, depending on the choices of the agents, is sometimes successful and sometimes unsuccessful. A strengthening of P is an addition to P that gives the agents guidance on how to choose among the options that P gives them.

The idea is that such a strengthening can leave good properties of a protocol intact, while reducing the chance of failure. For example, any strengthening of LNS will inherit the property that there are no redundant calls: It will still be the case that agents only call other agents if they do not know their secrets.

Let us illustrate this with a small example, also featuring as a running example in the technical sections (see Figure 6.1 on page 169). There are three agents a, b, c. Agent a knows the number of b, and b and c know each other's number. Calling agents exchange secrets and numbers, which may expand the network, and they apply the LNS protocol, wherein you may only call another agent if you do not know its secret. If a calls b, it learns the secret of b and the number of c. All different ways to make further calls now result in all three agents knowing all secrets. If the first call is between b and c (and there are no other first calls than ab, bc, and cb), they learn each other's secret but no new number. The only possible next call now is ab, after which a and b know all secrets but not c. But although a now knows c's number, she is not permitted to call c, as she already learned c's secret by calling b. We are stuck. So, some executions of LNS on this graph are successful and others are unsuccessful.

Suppose we now strengthen the LNS protocol into LNS′ such that b and c have to wait before making a call until they are called by another agent. This means that b will first receive a call from a. Then all executions of LNS′ are successful on this graph. In fact, there is only *one* remaining execution: $ab; bc; ac$. The protocol LNS′ is a *strengthening* of the protocol LNS.

The main contributions of this chapter are as follows. We prove that with enough agents, all gossip graphs are constructible as subgraphs. We define what it means for a gossip protocol to be common knowledge between all agents. To this end we propose a logical semantics with an individual knowledge modality for such protocol knowledge. We then define various strengthenings of gossip protocols, both in the logical syntax and in the semantics. This includes a strengthening called uniform backward induction, a form of backward induction applied to (imperfect information) gossip protocol execution trees. We give some general results for strengthenings, but mainly apply our strengthenings to the protocol LNS: we investigate some basic gossip graphs (networks) on which we gradually strengthen LNS until all its executions are successful on that graph. However, no such strengthening will work for all gossip graphs. This is proved by a counterexample consisting of a six-agent gossip graph, that requires fairly detailed analysis. Some of our results involve the calculation and checking of large numbers of call sequences. For this we use an implementation in Haskell. While this implementation is an explicit state model checker, we also show how symbolic transformers can be used to model gossip calls.

In Section 6.1 we give the basic definitions to describe gossip graphs and calls. In Section 6.2 we prove that all gossip graphs are constructible as a subgraph. We then introduce a variant of epistemic logic to be interpreted on gossip graphs in Section 6.3. In particular we introduce a new operator for protocol-dependent knowledge. In Section 6.4 we define semantic and — using the new operator — syntactic ways to strengthen gossip protocols. We investigate how successful those strengthenings are and study their behavior under iteration. Section 6.5 contains our main result, that strengthening LNS to a strongly successful protocol is impossible. In Section 6.6 we discuss different ways how model checking can be used to automate the analysis of gossip. In Section 6.7 we wrap up and conclude.

6.1 Gossip graphs and calls

Gossip graphs are used to keep track of who knows which secrets and which telephone numbers.

6.1.1. Definition. Given a finite set of agents A, let id_A be the identity relation on A. A *gossip graph* G is a triple (A, N, S) where N and S are binary relations on A such that $\mathsf{id}_A \subseteq S \subseteq N$. An *initial gossip graph* is a gossip graph where $S = \mathsf{id}_A$. for all agents a we write N_a for $\{b \in A \mid N_a b\}$, and similarly S_a for $\{b \in A \mid S_a b\}$. The set of all initial gossip graphs is denoted by $\mathcal{G}$.

The relations model the basic knowledge of the agents. Agent *a knows the number* of b iff $N_a b$ and *a knows the secret* of b iff $S_a b$. If we have $N_a b$ and not $S_a b$ we also say that a knows the *pure number* of b.

6.1.2. Definition. A *call* is an ordered pair of agents $(a, b) \in (A \times A)$. We usually write ab instead of (a, b). Given a gossip graph G, a call ab is *possible* iff $N_a b$. Given a possible call ab, G^{ab} is the graph (A', N', S') such that $A' := A$, $N'_a := N'_b := N_a \cup N_b$, $S'_a := S'_b := S_a \cup S_b$, and $N'_c := N_c$, $S'_c := S_c$ for $c \neq a, b$. For a sequence of calls $ab; cd; \ldots$ we write σ or τ. The empty sequence is ϵ. We extend the notation G^{ab} to sequences of calls: $G^\epsilon := G$, $(G^\sigma)^{ab} := G^{\sigma;ab}$.

To visualize gossip graphs we draw N with dashed and S with solid arrows. When making calls, the property $S \subseteq N$ is preserved [Dit+15], so we omit the dashed N arrow if there already is a solid S arrow.

6.1.3. Example. Consider the following initial gossip graph G in which a knows the number of b, and b and c know each other's number:

$$\begin{tikzcd} a \arrow[r, dashed] & b \arrow[r, dashed, leftrightarrow] & c \end{tikzcd}$$

Suppose that a calls b. We obtain the gossip graph G^{ab} in which a and b know each other's secret and a now also knows the number of c:

$$\begin{tikzcd} a \arrow[r, leftrightarrow] \arrow[rr, dashed, bend left] & b \arrow[r, dashed, leftrightarrow] & c \end{tikzcd}$$

6.2 Constructible Graphs and Subgraphs

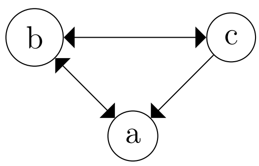
Given the rules of dynamic gossip, some situations or graphs are unreachable from initial gossip graphs. For example, if we only consider two agents Alice and Bob, then it cannot happen that Alice knows the secret of Bob but not vice versa. However, this situation changes if we consider configurations of subgraphs. Among three agents the asymmetric situation can occur, depending on calls involving the third one.

This raises the question which graphs can occur as subgraphs in a situation with more agents. This is particularly relevant if the number of agents is unknown or their reasoning power limited. In this section we give a simple answer: All finite gossip graphs can be constructed as parts of bigger graphs with more agents. Our proof is constructive and shows how to find an appropriate initial graph and which calls to make to construct any given subgraph.

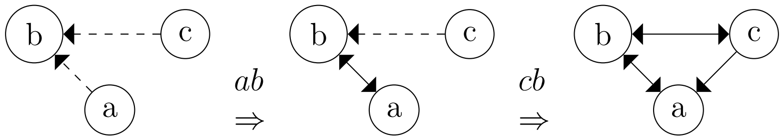
We start with a formal definition of what it means that a graph is reachable from an initial graph and then give two examples to show that some, but not all graphs have this property.

6.2.1. DEFINITION. A gossip graph $G = (A, N, S)$ is *reachable from an initial graph* iff there is a number relation $N_0 \subseteq N$ and a call sequence σ such that applying σ to the initial graph based on N_0 leads to the graph G, i.e. $(A, N_0, \mathsf{id}_A)^\sigma = G$.

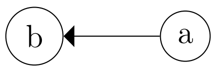
6.2.2. EXAMPLE. Is the gossip graph below reachable from an initial graph?



The answer is yes. We have $G = {G_0}^{ab;cb}$ where G_0 is the following graph on the left.



6.2.3. EXAMPLE. This graph is not reachable from an initial graph with two agents:

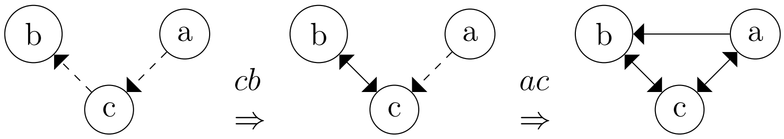


For any number of agents there are similar examples of graphs that are unreachable. But this changes, if we consider subgraphs.

6.2.4. DEFINITION. A gossip graph $G = (A, N, S)$ is called a *subgraph* of another gossip graph $G' = (A', N', S')$ iff (i) $A \subseteq A'$ and (ii) for all $a, b \in A$, we have Nab iff $N'ab$, and Sab iff $S'ab$. We then write $G \subseteq G'$. A gossip graph G is *constructible as a subgraph*, short *caas*, iff there is an initial gossip graph $G_0 = (A_0, N_0, S_0)$ and a calling sequence σ over A_0 such that $G \subseteq (G_0)^\sigma$.

It is easy to see that at least some unreachable graphs are caas.[1]

6.2.5. EXAMPLE. The graph from Example 6.2.3 is not reachable from a subgraph. But it is caas because we can start with the left graph below and do $\sigma := (cb); (ac)$ to construct it as a subgraph.



Our next question is, which gossip graphs are constructible as a subgraph?

One motivation to investigate this question is multi-agent reasoning. Consider a scenario where the number of agents is not known to some of the agents, or their reasoning power is limited so they cannot think about all agents at the same time. Then agents can no longer do reasoning like "There are only two other agents besides me and a call happened, so now they must have each other's secret."

A second motivation is that it can tell us something about formal approaches to the dynamic gossip problem: For any language that talks about gossip graphs with a formal syntax and semantics we can ask for its logic, i.e. validities. Those will depend on the *class of graphs* that we consider, which could include "unreachable" graphs or not. Does it matter if we include them?

The result that we will prove says that if agents do not know the total number of agents, then they have to consider more, and in fact all, subgraphs; and that if a language cannot express the number of agents then its validities will be the same with respect to the class of all gossip graphs as with respect to only the reachable ones.

6.2.6. THEOREM. *Every gossip graph is caas.*

We prove this by induction on the size of gossip graphs, defined as follows.

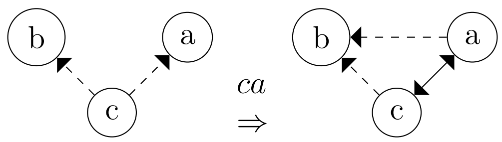
6.2.7. DEFINITION. For any gossip graph $G = (A, N, S)$, we define the size of G by $\mathsf{Size}(G) := |A| + |N \setminus S| + |S \setminus \mathsf{id}_A|$.

Intuitively, Definition 6.2.7 defines the size of a gossip graph as the number of things we draw: Each agent is a node, and we draw dashed ($N \setminus S$) or solid ($S \setminus \mathsf{id}_A$) arrows, but never both in the same direction.

[1] We invite the reader to pronounce 'caas' like 'kaas', the Dutch word for cheese.

6.2.8. EXAMPLE. For graph G in Example 6.2.3 we have $\mathsf{Size}(G) = 2+0+1 = 3$.

Our proof idea for Theorem 6.2.6 is to show that whenever all graphs with a certain size can be constructed as a subgraph starting from an initial graph, then we can also construct any graph that is one size bigger. For this we modify the initial graph and the sequence. In particular, we can use extra agents to build the relations we want. Per Definition 6.2.7 the size can only grow in three ways. First, if the new graph just has one more disconnected agent, then we can just add it to the initial constructing graph as well. Second, to add an $N \setminus S$-edge from a to b, we add a new agent c who knows numbers of a and b and then calls a:



Third, to add an $S \cap N$-edge from a to b, we add a new agent c who knows the number of b and whose number is known by a. Then we first let c call b and at the end let a call c. In fact, this is exactly Example 6.2.5.

Proof of Theorem 6.2.6:
By induction on $\mathsf{Size}(G)$. For any set X, let id_X denote the identity relation on X.

For the base case, suppose we have $G = (A, N, S)$ such that $\mathsf{Size}(G) = 0$. Then $A = N = S = \varnothing$ and it is easy to fulfill the claim with $G_0 := (\varnothing, \varnothing, \varnothing)$ and $\sigma := \epsilon$.

As an induction hypothesis, suppose any gossip graph G with $\mathsf{Size}(G) = k$ is caas. For the induction step, take any G' such that $\mathsf{Size}(G') = k + 1$. We want to show that G' is caas.

For this, let G be a proper subgraph of G' such that either (i) G' has one disconnected agent more than G, (ii) G' has one $N \setminus S$ edge more than G, or (iii) G' has one $S \cap N$ edge more than G. One of these must be the case by Definition 6.2.7.

In all cases $\mathsf{Size}(G) = k$, so by induction hypothesis G is caas. Hence there is an initial graph $G_0 = (A_0, N_0, \mathsf{id}_A)$ and a call sequence σ such that $G \subseteq (G_0)^\sigma$. Now consider the three cases:

(i) If G' has one disconnected agent more than G, say c, then let $G'_0 := (A_0 \cup \{c\}, N_0 \cup \{(c, c)\}, \mathsf{id}_{A_0 \cup \{c\}})$ and $\sigma' := \sigma$.

(ii) If G' has one $N \setminus S$ edge more than G, say $(a, b) \in (N \setminus S)$, let c be a fresh agent, let $G'_0 := (A_0 \cup \{c\}, N_0 \cup \{(c, a), (c, b)\}, \mathsf{id}_{A_0 \cup \{c\}})$ and $\sigma' := \sigma; (ca)$.

(iii) If G' has one $S \cap N$ edge more than G, say $(a, b) \in (N \cap S)$, let c be a fresh agent, let $G'_0 := (A_0 \cup \{c\}, N_0 \cup \{(a, c), (c, b)\}, \mathsf{id}_{A_0 \cup \{c\}})$ and $\sigma' := (cb); \sigma; (ac)$.

In each case we can check that $G' \subseteq (G'_0)^{\sigma'}$. Hence G' is caas. □

An informal corollary of Theorem 6.2.6 is the following. Suppose a logic describing dynamic gossip cannot "count" agents, i.e. the language it uses cannot express that there are n agents. Then any axiomatization of this logic is sound and complete for the class of all gossip graphs iff it is sound and complete for the class of reachable gossip graphs.

6.3 Epistemic Logic for Dynamic Gossip Protocols

6.3.1 Syntax and Protocols

We now introduce a language which we will interpret on gossip graphs. Atomic propositional variables $N_a b$ and $S_a b$ stand for "agent a knows the number of agent b" and "agent a knows the secret of agent b" and will be interpreted in the obvious way, using the N and S relations. Definitions 6.3.1 and 6.3.2 are simultaneous, as language construct $K_a^P \varphi$ is with respect to a protocol P.

6.3.1. Definition. We consider the language $\mathcal{L}$ given by

$$\begin{array}{lcl} \varphi & ::= & \top \mid N_a b \mid S_a b \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid K_a^P \varphi \mid [\pi]\varphi \\ \pi & ::= & ?\varphi \mid ab \mid (\pi \ ; \ \pi) \mid (\pi \cup \pi) \mid \pi^* \end{array}$$

where $a, b \in A$ and P is a protocol.

6.3.2. Definition. A *protocol condition* P_{ab} is a family of formulas in the language $\mathcal{L}$, indexed by two agents $a, b \in A$. Given a protocol condition P_{ab}, the corresponding *syntactic protocol* P is a program defined by

$$P := \left(\bigcup_{a \neq b \in A} (?(N_a b \wedge P_{ab}); ab) \right)^* ; ? \bigwedge_{a \neq b \in A} \neg \left(N_a b \wedge P_{ab} \right)$$

We require our protocols to be epistemic and symmetric. A protocol P is *epistemic* iff, for every $a, b \in A$, the protocol condition P_{ab} is equivalent to $K_a^P P_{ab}$ (using the logical semantics of Definition 6.3.8 below). A protocol P is *symmetric* iff, for every permutation J of agents, we have $P_{J(a)J(b)} = J(P_{ab})$, where $J(P_{ab})$ is the natural extension of J to formulas.

Other logical connectives and program constructs are defined by abbreviation as follows. Let $\pi^0 := ?\top$ and for all $n \in \mathbb{N}$ let $\pi^n := \pi^{n-1}; \pi$. Moreover, $N_a bcd$ stands for $N_a b \wedge N_a c \wedge N_a d$, and $N_a B$ for $\bigwedge_{b \in B} N_a b$. Similarly, we use $S_a bcd$ and $S_a B$ as abbreviations. If A is the set of all agents, we also write Ex_a for $S_a A$ to say that agent a is an expert. We write Ex_B for $\bigwedge_{b \in B} Ex_b$ and Ex for Ex_A to say that everyone is an expert. For program modalities, we use the standard definition for diamonds: $\langle \pi \rangle \varphi := \neg [\pi] \neg \varphi$.

Our new operator $K_a^P\varphi$ reads as "given protocol P, agent a knows that φ." Informally, this means that agent a knows that φ on the assumption that the agents have common knowledge that they all use protocol P. The simultaneous induction of formulas and programs in the language definition guarantees that $K_a^P\varphi$ is well-defined. This can be easily explained. Although P is the parameter in $K_a^P\varphi$, this might as well be its protocol condition P_{ab} (as this is the only variable part in the protocol definition), which is of type formula. In other words, the knowledge construct is inductively typed $K_a^\varphi\varphi$. We tend to use either the name of the protocol or the protocol condition to index the knowledge modality. The standard epistemic modality is an abbreviation $K_a\varphi := K_a^{\mathrm{ANY}}\varphi$, where ANY is the "make any call" protocol with protocol condition $\top$. The epistemic dual is defined as $\hat{K}_a^P\varphi := \neg K_a^P\neg\varphi$ and can be read as "given protocol P, agent a considers it possible that φ."

We do not take N_ab to be part of the protocol condition. It is rather a generic condition: a has to know b's number in order to call b, no matter which protocol is used. If N_ab we say that call ab is *possible*.

We do not include, as in other works [Dit+15; AW17], the success condition $?Ex$ in the protocol definition. We can therefore distinguish unsuccessful termination (not all agents know all secrets) from successful termination.

6.3.3. Definition. A terminating protocol execution is *successful* (or *succeeds*) iff afterwards all agents are experts. We say that a protocol P is *strongly successful* on G iff all terminating executions of P succeed: $[P]Ex$. A protocol is *weakly successful* on G iff some terminating execution of P succeeds: $\langle P\rangle Ex$. The protocol is *unsuccessful* on G iff no terminating execution succeeds: $[P]\neg Ex$. A protocol is *strongly successful* iff it is strongly successful on all gossip graphs G, and similarly for weakly successful and unsuccessful.

All our protocols can always be executed. If this is without making any calls, the protocol (extension) is called *empty*. Being empty is different from $[P]\bot$, which never holds.

Strong success implies weak success, but not vice versa. Formally, we have that $[P]\varphi \to \langle P\rangle\varphi$ is valid for all protocols P, but we do not have $\langle P\rangle\varphi \to [P]\varphi$, because our protocols are typically non-deterministic.

Intuitively, a protocol is *epistemic* if callers always know when to make a call, without being given instructions by a central scheduler. If a protocol is *symmetric* the names of the agents are irrelevant and therefore interchangeable. So a symmetric protocol is not allowed to "hard-code" agents to perform certain roles. This means that, for example, we cannot tell agent a to call b, as opposed to c, just because b comes before c in the alphabet. But we can tell a to call b, as opposed to c, on the basis that, say, a knows that b knows five secrets while c only knows two secrets. Epistemic and symmetric protocols capture the distributed peer-to-peer nature of the gossip problem.

6.3.4. Example. The "Learn New Secrets" protocol (LNS) says: You are allowed to call any agent whose secret you do not know yet. This is described by the protocol condition $LNS_{ab} := \neg S_{ab}$ and LNS is therefore the protocol

$$\left(\bigcup_{a \neq b \in A} (?(N_a b \wedge \neg S_a b); ab) \right)^* ; ? \bigwedge_{a \neq b \in A} \neg (N_a b \wedge \neg S_a b)$$

It is easy to see that this protocol is symmetric. We will later explain why $\neg S_{ab}$ is equivalent to $K_a^{\text{LNS}} \neg S_{ab}$, which means that LNS is also epistemic.

We want to discuss strengthenings of gossip protocols in general, and of LNS in particular. As we discussed in the introduction, a strengthening of a protocol helps the agents to make a smart choice among the options left open to them by the protocol. However, which choices are smart often depends on what you expect the other agents to do. A particular choice may, for example, be smart if the other agents are following LNS but not if the other agents use another protocol.

It is therefore important to consider what the agents know about the protocol that the others follow, and in particular, what it means to have *common knowledge* among the agents that a certain protocol is being followed. As such common knowledge is of a given protocol, we will parameterize the epistemic relation in our models with that protocol, and can therefore use that protocol in the knowledge modality.

Which protocol is common knowledge between the agents determines the epistemic relations in our models, that in turn, via the K_a^P operators used in formulas, determine which calls are allowed. To prevent circularity, knowledge is initially interpreted only for *simple* protocols, i.e., protocols without knowledge and call operators, such as LNS above or the protocol ANY (make *any* call), with protocol condition $\top$.

The meaning of $[ab]\varphi$ in our framework is "after call ab, φ holds", without reference to a protocol. We can syntactically enforce protocol P for this call by $[?P_{ab}; ab]\varphi$, for "after the call ab that is permitted according to protocol P, it will be the case that φ." The order of $?P_{ab}; ab$ is crucial here, because already a simple protocol like LNS only makes what we could call "Moore calls": Immediately after making the call ab, it is no longer allowed.

6.3.2 Protocol-Dependent Knowledge

We now define how to interpret our language on gossip graphs. For this we will use the standard tool from epistemic logic, namely Kripke models. The possible worlds of these Kripke models will be pairs of initial gossip graphs and histories. This way of modeling is in fact similar to the usage of type variables in DEMO-S5, as we discussed in Section 3.1.

We assume that the initial gossip graph is common knowledge to all agents and that time is synchronous, meaning that all agents know how many calls happened.

In the continuation we will show that even under these strong assumptions we cannot always strengthen our protocols to guarantee successful termination. Weaker assumptions are quite possible, but make it even harder to guarantee success, see Section 6.7.

Given a set of agents A and an initial gossip graph G, we will define the corresponding *gossip model* for G as a history-based Kripke model consisting of all *gossip states* (G, σ) where G is the initial gossip graph and σ is a sequence of calls possible on G. Epistemic relations between gossip states are parameterized by protocols, and the valuation is determined by the numbers and secrets known by the agents in the gossip state. As the gossip model is uniquely determined by G, we do not use any separate notation for gossip models and it suffices to know the point of evaluation (G, σ).

Our gossip states (G, σ) will always contain the complete call history and G is always an initial graph. We also want to refer to the gossip graph after the calls were made. To each gossip state (G, σ) we therefore associate the *current* gossip graph $G^\sigma = (A, N^\sigma, S^\sigma)$. The current gossip graph alone is not enough to model the knowledge of our agents, because different gossip states may correspond to the same gossip graph: as usual in modal logic, different modal properties may be satisfied by worlds with the same valuation.

The semantics for the novel epistemic operator K_a^P makes the background assumption of a protocol P explicit. With each agent we associate a whole family of epistemic equivalence relations, indexed by protocols. Those protocols in turn refer to the knowledge of our agents. Hence the following Definitions 6.3.5, 6.3.6, 6.3.7 and 6.3.8 are done simultaneously. We note that the $\sim_a^P$ defined below are indeed equivalence relations, which can be seen by an induction on the length of the call sequences.

6.3.5. Definition. If $(G, \sigma) \vDash P_{ab}$ we say that ab is *P-permitted* at (G, σ). A *P-permitted call sequence* consists of P-permitted calls.

6.3.6. Definition. The *epistemic equivalence relation* $\sim_a^P$ over gossip states for agent a, given that protocol P is common knowledge, is defined as:

1. $(G, \epsilon) \sim_a^P (G, \epsilon)$;
2. if $(G, \sigma) \sim_a^P (G, \tau)$, $N_b^\sigma = N_b^\tau$, $S_b^\sigma = S_b^\tau$, and ab is P-permitted at (G, σ) then $(G, \sigma; ab) \sim_a^P (G, \tau; ab)$;
   if $(G, \sigma) \sim_a^P (G, \tau)$, $N_b^\sigma = N_b^\tau$, $S_b^\sigma = S_b^\tau$, and ba is P-permitted at (G, σ) then $(G, \sigma; ba) \sim_a^P (G, \tau; ba)$;
3. if $(G, \sigma) \sim_a^P (G, \tau)$ then for all $c, d, e, f \neq a$ for which cd and ef are P-permitted at (G, σ) and (G, τ) respectively, let $(G, \sigma; cd) \sim_a^P (G, \tau; ef)$.

6.3.7. Definition. Given a set of agents A and an initial gossip graph G, the corresponding *gossip model* for G is the history-based Kripke model consisting of all *gossip states* (G, σ) where G is the initial gossip graph and σ is a sequence of calls possible on G, with equivalence relations $\sim_a^P$ between gossip states. The *execution tree* of a protocol P given G is the submodel of the gossip model restricted to the set of those (G, σ) where σ is P-permitted and to the relation $\sim_a^P$.

6.3.8. DEFINITION. We inductively define the interpretation of a formula $\varphi \in \mathcal{L}$ on a gossip state (G, σ) where $G = (A, N, S)$ is an initial graph, σ a history and $G^\sigma = (A, N^\sigma, S^\sigma)$ the associated current gossip graph.

$$\begin{array}{lll}
G, \sigma \models \top & & \text{always} \\
G, \sigma \models N_a b & \text{iff} & N_a^\sigma b \\
G, \sigma \models S_a b & \text{iff} & S_a^\sigma b \\
G, \sigma \models \neg\varphi & \text{iff} & G, \sigma \not\models \varphi \\
G, \sigma \models \varphi \wedge \psi & \text{iff} & G, \sigma \models \varphi \text{ and } G, \sigma \models \psi \\
G, \sigma \models K_a^P \varphi & \text{iff} & G', \sigma' \models \varphi \text{ whenever } (G', \sigma') \sim_a^P (G, \sigma) \\
G, \sigma \models [\pi]\varphi & \text{iff} & G', \sigma' \models \varphi \text{ whenever } (G', \sigma') \in [\![\pi]\!](G, \sigma)
\end{array}$$

where $[\![\cdot]\!]$ is the following interpretation of programs:

$$\begin{array}{rcl}
[\![?\varphi]\!](G, \sigma) & := & \{(G, \sigma) \mid G, \sigma \models \varphi\} \\
[\![ab]\!](G, \sigma) & := & \{(G, (\sigma; ab)) \mid (G, \sigma) \vDash N_a b\} \\
[\![\pi; \pi']\!](G, \sigma) & := & \bigcup\{[\![\pi']\!](G', \sigma') \mid (G', \sigma') \in [\![\pi]\!](G, \sigma)\} \\
[\![\pi \cup \pi']\!](G, \sigma) & := & [\![\pi]\!](G, \sigma) \cup [\![\pi']\!](G, \sigma) \\
[\![\pi^*]\!](G, \sigma) & := & \bigcup\{[\![\pi^n]\!](G, \sigma) \mid n \in \mathbb{N}\}
\end{array}$$

Let us first explain why the interpretation of knowledge is well-defined. As before, to facilitate the explanation, let ψ be the protocol condition of protocol P. The interpretation of $K_a^P \varphi$ in state (G, σ) is a function of the truth of φ in all (G, τ) accessible via $\sim_a^P$. This is standard. Non-standard is that the equivalence relation $\sim_a^P$ is a function of the truth of ψ in gossip states (G, τ') for strict prefixes τ' of τ. Hence knowledge can never be self-referential.

In our semantics all calls can be evaluated, no matter which protocol is common knowledge. That is, in case agent a does not know the number of agent b, then $[ab]\varphi$ is trivially true for all φ, and if ab is a possible call, i.e., if a knows the number of b, then $[ab]\varphi$ is true if φ is true after the call ab, independently from whether ab is P-permitted or not. Our semantics reflects that agents are free to consider whatever calls they want.

For protocol-dependent knowledge however, the epistemic alternatives given by $\sim_a^P$ are restricted according to protocol P. Hence the relation for P will be empty at states that cannot be reached by it. This leads to a strange but, after some reflection, unsurprising fact that if a call happens that is not permitted according to a protocol P but some agent a still assumes that P is common knowledge, then this agent will turn insane, i.e. believe everything, including contradictions: $\neg P_{ab} \to [ab] K_c^P \bot$.

A direct consequence of our semantics is that agents always know which numbers and secrets they know. Recalling that $K_a^\top \varphi$ equals $K_a \varphi$ (the protocol ANY has condition $\top$), it is elementary that $N_a b \leftrightarrow K_a N_a b$, $\neg N_a b \leftrightarrow K_a \neg N_a b$, $S_a b \leftrightarrow K_a S_a b$, and $\neg S_a b \leftrightarrow K_a \neg S_a b$ are all valid.

We need a little bit more. From the properties of the relation $\sim_a^P$ and the semantics, it follows that $K_a^{\mathrm{ANY}} \varphi \to K_a^P \varphi$ is valid for any protocol P. This expresses that ANY is the weakest of all protocols, see also Lemma 6.4.3 below. Hence we also have $N_a b \to K_a^P N_a b$. It is also easy to see that protocol dependent knowledge has the standard properties of knowledge, such as truthfulness, $K_a^P \varphi \to \varphi$, from which we obtain $N_a b \leftrightarrow K_a^P N_a b$ etc. The validity of $\neg S_a b \leftrightarrow K_a^{\mathrm{LNS}} \neg S_a b$ means that LNS is an epistemic protocol.

These equivalences are a common feature of many gossip settings. We also assume common knowledge of the initial gossip graph, from which individual knowledge is derivable. To illustrate this, in Example 6.1.3, where a knows the number of b and b and c know each other's number, a knows all that prior to having made any call. So $G, \epsilon \models K_a^{\mathrm{LNS}} N_b c$, etc. Of course, knowledge about other agents *not* having a number or a secret is not preserved after calls.

6.3.9. Definition. For any initial gossip graph G and any protocol P we define the *extension of* P *on* G by

$$\begin{array}{lcl} P_0(G) & := & \{\epsilon\} \\ P_{i+1}(G) & := & \{\sigma; ab \mid \sigma \in P_i(G)\ a, b \in A,\ (G, \sigma) \vDash P_{ab}\} \\ P(G) & := & \bigcup_{i<\omega} P_k(G) \end{array}$$

The *extension of* P is then $P(\mathcal{G}) := \bigcup_{G \in \mathcal{G}} P(G)$.

We recall that $\mathcal{G}$ is the set of all initial gossip graphs. For $P(\mathcal{G})$ we often write P unless confusion results. In other words, we tend to identify a protocol with its extension. So, $P \subseteq P'$ means $P(\mathcal{G}) \subseteq P'(\mathcal{G})$, etc. Given a set X of call sequences, sequence $\sigma \in X$ is *terminal* iff for all calls ab, we have $\sigma; ab \notin X$. For the subset of the terminal sequences of X we write $\overline{X}$.

Not all protocols discussed in this work are definable in the logical language. We therefore need the additional notion of a semantic protocol, defined by its extension.

6.3.10. Definition. A *semantic protocol* is a function $P\colon \mathcal{G} \to \mathcal{P}((A \times A)^*)$ mapping initial gossip graphs to sets of call sequences.

We also require that semantic protocols are epistemic and symmetric, adapting the definitions of these two properties as follows. Let J be a permutation. Let $J(\epsilon) = \epsilon$ and $J(\sigma; ab) = J(\sigma); J(a)J(b)$. Then a semantic protocol P is *symmetric* iff $P = J(P)$ (seen as extensions) for any permutation J, where $J(P)$ is the set of all $J(\sigma)$ with $\sigma \in P$. To determine whether P is *epistemic* we replace all conditions "ab is P-permitted at (G, σ)" in Definition 6.3.6 of the epistemic relation by "$\sigma; ab \in P(G)$" and then proceed as before.

6.3.11. EXAMPLE. We continue with Example 6.1.3. The execution tree of LNS on this graph is shown in Figure 6.1. We denote calls with gray arrows and the epistemic equivalence relation with dotted lines. For example, agent a cannot distinguish whether call bc or cb happened. At the end of each branch the termination of LNS is denoted with ✓ if successful, and × if unsuccessful.

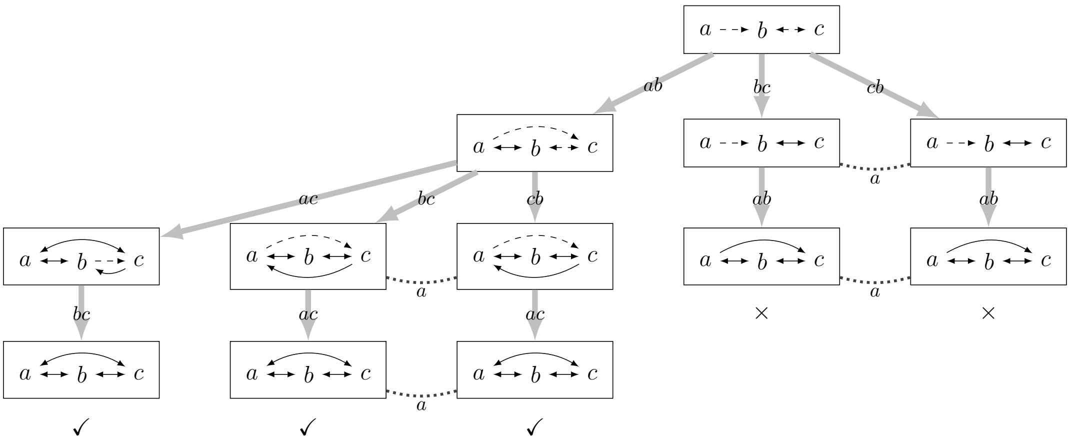


Figure 6.1: Example of an execution tree for LNS.

To illustrate our semantics, for this graph G we have:

- $G, \epsilon \vDash N_a b \wedge \neg S_a b$ — the call ab is LNS-permitted at the start.
- $G, \epsilon \vDash [ab](S_a b \wedge S_b a)$ — after the call ab the agents a and b know each other's secret
- $G, \epsilon \vDash [ab]\langle ac\rangle\top$ — after the call ab the call ac is possible.
- $G, \epsilon \vDash [ab][LNS]Ex$ — after the call ab the LNS protocol will always terminate successfully.
- $G, \epsilon \vDash [bc \cup cb][LNS]\neg Ex$ — after the calls bc or cb the LNS protocol will always terminate unsuccessfully.
- $G, \epsilon \vDash [bc \cup cb]K_a^{LNS}(S_b c \wedge S_c b)$ — after the calls bc or cb, agent a knows that b and c know each others secret.
- $G, ab; bc; ac \vDash \bigwedge_{i \in \{a,b,c\}} K_i^{LNS} Ex$ — after the call sequence $ab; bc; ac$ everyone knows that everyone is an expert.

Note that here we only have epistemic edges for agent a, and that those are between states that are isomorphic. In synchronous gossip with three agents, if you are not involved in a call, you know that the other two agents must have called. You may only be uncertain about the direction of that call. But the direction of the call does not matter for the numbers and secrets being exchanged. Hence all agents always know the current situation. We will see a more interesting epistemic relation later, in Figure 6.4.

In Example 6.3.11 we have three successful and two unsuccessful LNS sequences. To ensure success, agent a has to make the first call. This can be achieved by strengthening LNS. In the next section we will define what it means to strengthen a protocol and then give syntactic and semantic ways to do so.

6.4 Strengthening of Protocols

6.4.1 What is strengthening?

In our semantics it is common knowledge among the agents that they follow a certain protocol, for example LNS. Can they use this information to prevent making "bad" calls that lead to an unsuccessful sequence?

If we look at the execution graph given in Figure 6.1, then it seems easy to fix the protocol. Agents b and c should wait and not make the first call. Agent b should not make a call before he has received a call from a. We cannot say this in our logic as we have no converse modalities to reason over past calls. In this case however, there is a different way to ensure the same result. We can ensure that b and c wait before calling by a strengthening of LNS that only allows a first call from i to j if j does not know the number of i. To determine that a call is not the first call we need another property: after at least one call happened there is an agent who knows another agent's secret.

We can define this new protocol by protocol condition $P_{ij} := \mathrm{LNS}_{ij} \wedge (\neg N_j i \vee \bigvee_{k \neq l} S_k l)$. It is important to observe that this new protocol is again symmetric and epistemic. In particular, agents always know whether $(\neg N_j i \vee \bigvee_{k \neq l} S_k l)$, even though each of the disjuncts alone would not be an epistemic protocol condition. Because of synchronicity, not only the callers but also all other agents know that there are agents k and l such that k knows the secret of l.

This is an ad-hoc solution specific to this initial gossip graph. Could we also give a general definition to improve LNS which works on more or even all initial graphs? The answer to that is: more, yes, but all, no.

We will now discuss different ways to improve protocols by making them more restrictive. Our goal is to rule out unsuccessful sequences while keeping at least some successful ones. Doing this can be difficult because we still require the strengthened protocols to be epistemic and symmetric. Hence we are not allowed to arbitrarily rule out specific calls using the names of agents, for example. Whenever a call is removed from the protocol, we also have to remove all calls to other agents that the caller cannot distinguish: it has to be done *uniformly*. But before we discuss specific ideas for strengthening, let us define it.

6.4.1. DEFINITION. A protocol P' is a *syntactic strengthening* of a protocol P iff $P'_{ab} \to P_{ab}$ is valid for all agents $a \neq b$. A protocol P' is a *semantic strengthening* of a protocol P iff $P' \subseteq P$.

In the case of a syntactic strengthening, P and P' are implicitly required to be syntactic protocols. Vice versa however, syntactic protocols *can* be semantic strengthenings of each other. In fact, we have the following.

6.4.2. PROPOSITION. *Every syntactic strengthening is a semantic strengthening.*

Proof:
Let P' be a syntactic strengthening of a protocol P. Let a gossip graph G be given. We show by induction on the length of σ that $\sigma \in P'(G)$ implies $\sigma \in P(G)$. The base case where $\sigma = \epsilon$ is trivial.

For the induction step, consider any $\sigma = \tau; ab$. As $\tau; ab \in P'(G)$, we also have $\tau \in P'(G)$ and $G, \tau \models P'_{ab}$. From $\tau \in P'(G)$ and the inductive hypothesis, it follows that $\tau \in P(G)$. From $G, \tau \models P'_{ab}$ and the validity of $P'_{ab} \to P_{ab}$ follows $G, \tau \models P_{ab}$. Finally, by Definition 6.3.9, $\tau \in P(G)$ and $G, \tau \models P_{ab}$ imply $\tau; ab \in P(G)$. $\square$

6.4.3. LEMMA. *Suppose P is a strengthening of Q. Then $K^Q\varphi \to K^P\varphi$ and $\hat{K}^P\varphi \to \hat{K}^Q\varphi$ are both valid.*

Proof:
This follows immediately from the semantics of knowledge (Definition 6.3.8). $\square$

Although strengthening is a relation between two protocols P and Q, it is typically the case for a strengthening Q of P that Q is defined by a restricting transformation of P, i.e., $Q = P^\heartsuit$ for some operation $\heartsuit$ as defined in the next sections. We will use $\heartsuit$ to denote arbitrary strengthenings.

6.4.2 Syntactic Strengthening: Look-Ahead and One-Step

We will now present concrete examples of syntactic strengthening.

6.4.4. DEFINITION. Let P be a protocol. We define four kinds of syntactic strengthening of P:

$$\begin{array}{lrcl}
\text{hard look-ahead strengthening :} & P^{\blacksquare}_{ab} & := & P_{ab} \wedge K^P_a[ab]\langle P\rangle Ex \\
\text{soft look-ahead strengthening :} & P^{\blacklozenge}_{ab} & := & P_{ab} \wedge \hat{K}^P_a[ab]\langle P\rangle Ex \\
\text{hard one-step strengthening :} & P^{\Box}_{ab} & := & P_{ab} \wedge K^P_a[ab](Ex \vee \bigvee_{i,j}(N_i j \wedge P_{ij})) \\
\text{soft one-step strengthening :} & P^{\Diamond}_{ab} & := & P_{ab} \wedge \hat{K}^P_a[ab](Ex \vee \bigvee_{i,j}(N_i j \wedge P_{ij}))
\end{array}$$

The *hard* look-ahead strengthening allows agents to make a call iff the call is allowed by the original protocol and moreover they *know* that making this call yields a situation where the original protocol can still succeed.

For example, consider $\text{LNS}^{\blacksquare}$. Informally, its condition is that a is permitted to call b iff a does not have the secret of b and a knows that after making the call to b, it is still possible to follow LNS in such a way that all agents become experts.

The *soft* look-ahead strengthening allows more calls than the hard look-ahead strengthening because it only demands that *a considers it possible* that the protocol can succeed after the call. This can be interpreted as a good faith or lucky draw assumption that the previous calls between other agents have been made "in a good way". Soft look-ahead strengthening allows agents to take a risk.

Both the soft and the hard look-ahead strengthening include a diamond $\langle P \rangle$ with the original protocol, which contains a Kleene star. To evaluate this, we need to compute the execution tree of P for the initial gossip graph G. In practice this can make it hard to check the precondition of the new protocol.

The *one-step* strengthenings, in contrast, only use the protocol condition P_{ij} in their formalization and not the entire protocol P. This means that they provide an easier to compute, but less reliable alternative to full look-ahead, namely by looking only one step ahead. We only demand that agent a knows (or, in the soft version, considers it possible) that after the call, everyone is an expert or the protocol can still go on for at least one more step — though it might be that all continuation sequences will eventually be unsuccessful and thus this next call would already have been excluded by both look-ahead strengthenings.

An obvious question now is, can these or other strengthenings get us from weak to strong success? Do these strengthenings only remove unsuccessful sequences, or will they also remove successful branches, and maybe even return an empty and unsuccessful protocol? In our next example everything still works fine.

6.4.5. Example. Consider Example 6.3.11 again. It is easy to see that both the soft and the hard look-ahead strengthening rule out the two unsuccessful branches in this execution tree and keep the successful ones.

Protocol $\text{LNS}^{\blacksquare}$ only preserves alternatives that are all successful and $\text{LNS}^{\blacklozenge}$ only eliminates alternatives if they are all unsuccessful. In the execution tree in Figure 6.1, the effect is the same for $\text{LNS}^{\blacksquare}$ and $\text{LNS}^{\blacklozenge}$, because at any state the agents always know which calls lead to successful branches.

This is typical for gossip scenarios with three agents: if a call happened, the agent not involved in the call might be unsure about the direction of the call, but it knows who the callers are.

The one-step strengthenings are not enough to rule out the unsuccessful sequences. This is because the unsuccessful sequences are of length 2 but the one-step strengthenings can only remove the last call in a sequence. In this case, the protocols $\text{LNS}^{\Box}$ and $\text{LNS}^{\Diamond}$ both rule out the call ab after bc or cb happened.

6.4.3 Semantic Strengthening: Uniform Backward Defoliation

We now present two semantic strengthenings. They are inspired by the notion of backward induction, a well-known solution concept in decision theory and game theory [OR94]. We will discuss this at greater length when defining the arbitrary iteration of these semantic strengthenings and in Section 6.7.

In backward induction, given a game tree or search tree, a parent node is called *bad* if all its children are loosing or bad nodes. Similarly, in trees with information sets of indistinguishable nodes, a parent node can be called bad if all its children are bad *and if also all children from indistinguishable nodes are bad*. Similar notions were considered in [BSZ09; Per14]. Again, we have a soft and a hard version.

We define *uniform backward defoliation* on the execution trees of dynamic gossip as follows to obtain two semantic strengthenings. We choose the name "defoliation" here because a single application of this strengthening only removes leaves and not whole branches of the execution tree. The iterated versions we present later are then called *uniform backward induction*.

6.4.6. Definition. Suppose we have a protocol P and an initial gossip graph G. We define the *Hard Uniform Backward Defoliation* (HUBD) and *Soft Uniform Backward Defoliation* (SUBD) of P as follows.

$$\begin{array}{rcl} P^{\mathsf{HUBD}}(G) & := & \{\sigma \in P(G) \mid \sigma = \epsilon, \text{ or } \sigma = \tau; ab \text{ and } \forall (G,\tau') \sim_a^P (G,\tau) \\ & & \qquad \text{such that } \tau' \in \overline{P(G)} \text{ implies } (G,\tau';ab) \models Ex\} \\ P^{\mathsf{SUBD}}(G) & := & \{\sigma \in P(G) \mid \sigma = \epsilon, \text{ or } \sigma = \tau; ab \text{ and } \exists (G,\tau') \sim_a^P (G,\tau) \\ & & \qquad \text{such that } \tau' \in \overline{P(G)} \text{ implies } (G,\tau';ab) \models Ex\} \end{array}$$

In this definition, $\forall (G,\tau') \sim_a^P (G,\tau)$ implicitly stands for "for all $\tau' \in P(G)$ such that $(G,\tau') \sim_a^P (G,\tau)$", because for (G,τ') to be in $\sim_a^P$ relation to another gossip state, τ' must be P-permitted; similarly for the existential quantification.

The HUBD strengthening keeps the calls which *must* lead to a non-terminal state or a state where everyone is an expert and SUBD keeps the calls which *might* do so. Equivalently, we can say that HUBD removes calls which may go wrong and SUBD removes those calls which will go wrong — where going wrong means leading to a terminal node where not everyone is an expert.

We can now prove that for any gossip protocol *Hard Uniform Backward Defoliation* is the same as *Hard One-Step Strengthening*, in the sense that their extensions are the same on any gossip graph, and that *Soft Uniform Backward Defoliation* is the same as *Soft One-Step Strengthening*.

6.4.7. Theorem. *$P^{\Box} = P^{\mathsf{HUBD}}$ and $P^{\Diamond} = P^{\mathsf{SUBD}}$*

Proof:
Note that ϵ is an element of both sides of both equations. For any non-empty sequence we have the following chain of equivalences for the hard versions of UBD and one-step strengthening:

$$
\begin{aligned}
& (\sigma; ab) \in P^{\Box}(G) \\
\iff\ & G, \sigma \vDash P^{\Box}_{ab} \\
\iff\ & G, \sigma \vDash P_{ab} \wedge K^P_a[ab]\Big(\bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex\Big) \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } (G, \sigma) \vDash K^P_a[ab]\Big(\bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex\Big) \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } \forall (G, \sigma') \sim^P_a (G, \sigma) : (G, \sigma'; ab) \vDash \bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } \forall (G, \sigma') \sim^P_a (G, \sigma) : \sigma'; ab \notin \overline{P(G)} \text{ or } (G, \sigma'; ab) \vDash Ex \\
\iff\ & (\sigma; ab) \in P^{\mathsf{HUBD}}(G)
\end{aligned}
$$

And we have a similar chain of equivalences for the soft versions:

$$
\begin{aligned}
& (\sigma; ab) \in P^{\Diamond}(G) \\
\iff\ & G, \sigma \vDash P^{\Diamond}_{ab} \\
\iff\ & G, \sigma \vDash P_{ab} \wedge \hat{K}^P_a[ab]\Big(\bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex\Big) \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } (G, \sigma) \vDash \hat{K}^P_a[ab]\Big(\bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex\Big) \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } \exists (G, \sigma') \sim^P_a (G, \sigma) : (G, \sigma'; ab) \vDash \bigvee\nolimits_{i,j}(N_i j \wedge P_{ij}) \vee Ex \\
\iff\ & (\sigma; ab) \in P(G) \text{ and } \exists (G, \sigma') \sim^P_a (G, \sigma) : \sigma'; ab \notin \overline{P(G)} \text{ or } (G, \sigma'; ab) \vDash Ex \\
\iff\ & (\sigma; ab) \in P^{\mathsf{SUBD}}(G)
\end{aligned}
$$

□

Similarly to backward induction in perfect information games, uniform backward defoliation is *rational*. If you know that a move (call) that you could make is losing (unsuccessful) then it is clearly irrational to make it. So a rational agent should rule out those moves. This yields SUBD. The strengthening HUBD is even stricter: If you consider it possible that the move/call might be losing, then do not make it.

6.4.4 Iterated Strengthenings

The syntactic strengthenings we looked at are all defined in terms of the original protocol. In $P^{\blacksquare}_{ab} := P_{ab} \wedge K^P_a[ab]\langle P \rangle Ex$ the given P occurs in three places. Firstly, in the protocol condition P_{ab} requiring that the call is permitted according to the old protocol P — this ensures that the new protocol is a strengthening of the original P. Secondly, as a parameter to the knowledge operator, in K^P_a, which means that agent a knows that everyone followed P (and that this is common knowledge). Thirdly, in the part $\langle P \rangle$ assuming that after the considered call everyone will continue to follow protocol P in the future.

Hence we have strengthened the protocol that the agents use and thereby their behavior, but not their assumptions about what protocol other agents follow. For example, when $P = \text{LNS}$, all agents now act according to $\text{LNS}^{\blacksquare}$, on the assumption that all other agents act according to LNS. This does not mean that agents cannot determine what they know if $\text{LNS}^{\blacksquare}$ were common knowledge: each agent a can check that knowledge using $K_a^{\text{LNS}^{\blacksquare}} \varphi$. But this $K_a^{\text{LNS}^{\blacksquare}}$ modality is not part of the protocol $\text{LNS}^{\blacksquare}$. The agents do not use this knowledge to determine whether to make calls.

But why should our agents stop their reasoning here? It is natural to iterate strengthenings and determine whether we can further improve our protocols by also updating the knowledge of the agents.

For example, consider repeated hard one-step strengthening:

$$(P^{\Box})^{\Box}_{ab} = P^{\Box}_{ab} \wedge \hat{K}_a^{P^{\Box}} [ab](Ex \vee \bigvee_{i,j} (N_i j \wedge P^{\Box}_{ij}))$$

In this section we investigate iterations and combinations of protocol strengthenings. In particular we investigate various combinations of hard and soft one-step and look-ahead strengthening, in order to determine how they relate to each other.

6.4.8. Definition. Let P be a syntactic protocol. For any of the four syntactic strengthenings $\heartsuit \in \{\blacksquare, \blacklozenge, \Box, \Diamond\}$, we define its iteration by adjusting the protocol condition as follows, which implies $P^{\heartsuit 1} = P^{\heartsuit}$:

$$\begin{array}{lcl} P^{\heartsuit 0}_{ab} & := & P_{ab} \\ P^{\heartsuit (k+1)}_{ab} & := & (P^{\heartsuit k})^{\heartsuit}_{ab} \end{array}$$

Let now P be a semantic protocol, and let $\heartsuit \in \{\mathsf{HUBD}, \mathsf{SUBD}\}$. We define their iteration, for all gossip graphs G, by:

$$\begin{array}{lcl} P^{\heartsuit 0}(G) & := & P(G) \\ P^{\heartsuit (k+1)}(G) & := & (P^{\heartsuit k})^{\heartsuit}(G) \end{array}$$

It is easy to check that Theorem 6.4.7 generalizes to the iterated strengthenings as follows.

6.4.9. Corollary. *For any $k \in \mathbb{N}$, we have:*

$$P^{\Box k} = P^{\mathsf{HUBD} k} \text{ and } P^{\Diamond k} = P^{\mathsf{SUBD} k}$$

Proof:
By induction using Theorem 6.4.7. □

6.4.10. Example. We reconsider Examples 6.3.11 and 6.4.5, and we recall that $\mathrm{LNS}^{\Box}$ and $\mathrm{LNS}^{\Diamond}$ both rule out the call ab after bc or cb happened. To eliminate bc and cb as the first call, we have to iterate one-step strengthening: $(\mathrm{LNS}^{\Box})^{\Box}$ is strongly successful on this graph, as well as $(\mathrm{LNS}^{\Diamond})^{\Diamond}$, $(\mathrm{LNS}^{\Box})^{\Diamond}$ and $(\mathrm{LNS}^{\Diamond})^{\Box}$.

6.4.11. Example. We consider the "N"-shaped gossip graph shown in Figure 6.2.

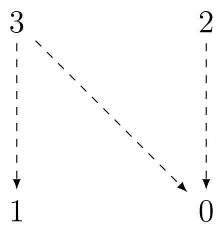


Figure 6.2: The "N" Graph.

There are 21 LNS sequences for this graph, of which 4 are successful (✓) and 17 are unsuccessful (×):

20; 30; 01; 31	×	30; 20; 01; 31; 21	✓	30; 31; 20; 21; 01	×
20; 30; 31; 01	×	30; 20; 21; 01; 31	✓	31; 10; 20; 30	×
20; 31; 10; 30	×	30; 20; 21; 31; 01	✓	31; 10; 30; 20	×
20; 31; 30; 10	×	30; 20; 31; 01; 21	×	31; 20; 10; 30	×
30; 01; 20; 31	×	30; 20; 31; 21; 01	×	31; 20; 30; 10	×
30; 01; 31; 20	×	30; 31; 01; 20	×	31; 30; 10; 20	×
30; 20; 01; 21; 31	✓	30; 31; 20; 01; 21	×	31; 30; 20; 10	×

We can show the call sequences in a more compact way if we only distinguish call sequences up to the moment when it is decided whether LNS will succeed. Formally, consider the set of minimal $\sigma \in \mathrm{LNS}(G)$ such that for all two terminal LNS-sequences $\tau, \tau' \in \overline{\mathrm{LNS}(G)}$ extending σ, we have $G, \tau \vDash Ex$ iff $G, \tau' \vDash Ex$. We will use this shortening convention throughout the chapter.

20	×
30; 01	×
30; 20; 01	✓
30; 20; 21	✓
30; 20; 31	×
30; 31	×
31	×

It is pretty obvious what the agents should do here: Agent 2 should not make the first call but let 3 call 0 first. The soft look-ahead strengthening works well on this graph: It disallows all unsuccessful sequences and keeps all successful ones.

For example, after call 30, agent 2 considers it possible that call 30 happened and in this case the call 20 can lead to success. Hence the protocol condition of LNS$^{\blacklozenge}$ is fulfilled. The strengthening LNS$^{\blacklozenge}$ is strongly successful on this graph.

But note that 2 does not *know* that 20 is safe, because the first call could have been 31 as well and for agent 2 this would be indistinguishable from 30. Therefore the hard look-ahead strengthening is too restrictive here. In fact, the only call which $LNS^{\blacksquare}$ still allows is 30 at the beginning. After that no more calls are allowed by the hard look-ahead strengthening.

A full list showing which call sequences are allowed by which strengthenings of LNS for this example is provided in Table 6.4. "Full" means that we continue iterating the strengthening until $P^{\heartsuit k}(G) = P\heartsuit(k+1)(G)$ for the given graph G. Such fixpoints of protocol strengthening will be formally introduced in the next section.

The hard look-ahead strengthening restricts the set of allowed calls based on a full analysis of the whole execution tree. One might thus expect, that applying hard look-ahead more than once would not make a difference. However, we have the following negative results on iterating hard look-ahead strengthening and the combination of hard look-ahead and hard one-step strengthening.

6.4.12. FACT. Hard look-ahead strengthening is not idempotent and does not always yield a fixpoint of hard one-step strengthening:

(i) There is a protocol P for which $P^{\blacksquare} \neq (P^{\blacksquare})^{\blacksquare}$.

(ii) There is a protocol P for which $(P^{\blacksquare})^{\square} \neq P^{\blacksquare}$.

Proof:

(i) Let G be the "N" graph from Example 6.4.11 and consider the protocol $P =$ LNS. Applying hard look-ahead strengthening once only allows the first call 30 and nothing after that call. If we now apply hard look-ahead strengthening again we get the empty set: $P^{\blacksquare}(G) \neq (P^{\blacksquare})^{\blacksquare}(G) = \varnothing$. See also Table 6.4.

(ii) The "diamond" graph that we will present in Example 6.4.6 can serve as an example here. We can show that the inequality holds for this graph by exhaustive search, using our Haskell implementation described in Section 6.6.1. Plain LNS has 48 successful and 44 unsuccessful sequences on this graph. Of these, LNS$^{\blacksquare}$ still includes 8 successful and 8 unsuccessful sequences. If we now apply hard one-step strengthening, we get (LNS$^{\blacksquare}$)$^{\square}$ where 4 of the unsuccessful sequences are removed. See also Table 6.3. □

Similarly, we can ask whether two soft strengthenings are related to each other, analogous to Fact 6.4.12. We do not know whether there is a protocol P for which $(P^{\blacklozenge})^{\lozenge} \neq P^{\blacklozenge}$ and leave this as an open question.

Another interesting property that strengthenings can have is *monotonicity*. Intuitively, a strengthening is monotone iff it preserves the inclusion relation between extensions of protocols. This property is useful to study the fixpoint behavior of strengthenings. We will now define monotonicity formally and then obtain some results for it.

6.4.13. Definition. A strengthening $(\cdot)^{\heartsuit}$ is called *monotone* iff for all protocols Q and P such that $Q \subseteq P$, we also have $Q^{\heartsuit} \subseteq P^{\heartsuit}$.

6.4.14. Proposition. *Soft one-step strengthening is monotone. More formally, let P be a protocol and Q be an arbitrary strengthening of P, i.e. $Q \subseteq P$. Then we also have $Q^{\lozenge} \subseteq P^{\lozenge}$.*

Proof:
As Q is a strengthening of P, the formula $Q_{ab} \to P_{ab}$ is valid. We want to show that $Q^{\lozenge}_{ab} \to P^{\lozenge}_{ab}$. Suppose that $G, \sigma \vDash Q^{\lozenge}_{ab}$, i.e.:

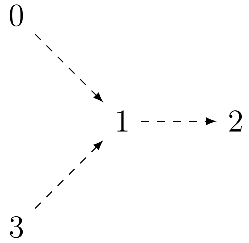
$$G, \sigma \vDash Q_{ab} \text{ and } G, \sigma \vDash \hat{K}^{Q}_{a}[ab](Ex \vee \bigvee_{i,j}(N_i j \wedge Q_{ij}))$$

From the first part and the validity of $Q_{ab} \to P_{ab}$, we get $G, \sigma \vDash P_{ab}$. The second part and the validity of $Q_{ij} \to P_{ij}$ give us $G, \sigma \vDash \hat{K}^{Q}_{a}[ab](Ex \vee \bigvee_{i,j}(N_i j \wedge P_{ij}))$. From that and Lemma 6.4.3 it follows that $G, \sigma \vDash \hat{K}^{P}_{a}[ab](Ex \vee \bigvee_{i,j}(N_i j \wedge P_{ij}))$. Combining these, it follows by definition of soft one-step strengthening that we have $G, \sigma \vDash P^{\lozenge}_{ab}$. □

6.4.15. Proposition. *Both hard strengthenings are not monotone: Let P and Q be protocols. If $Q \subseteq P$, then (i) $Q^{\blacksquare} \subseteq P^{\blacksquare}$ may not hold, and also (ii) $Q^{\square} \subseteq P^{\square}$ may not hold.*

Proof:
(i) *Hard one-step strengthening is not monotone*:

Consider the "spaceship" graph below with four agents 0, 1, 2 and 3 where 0 and 3 know 1's number, 1 knows 2's number, and 2 knows no numbers.



On this graph the LNS sequences up to decision point are:

$01;02$	$\times$	$01;31;12$	✓	$31;01;02$	✓	$31;12$	$\times$
$01;12$	$\times$	$01;31;32$	✓	$31;01;12$	✓	$31;32$	$\times$
$01;31;02$	$\times$	12	$\times$	$31;01;32$	$\times$		

Note that

$$\mathrm{LNS}^{\blacklozenge}(G) = \left\{ \begin{array}{l} (01;31;12;02;32), (01;31;12;32;02), (01;31;32;02;12), \\ (01;31;32;12;02), (31;01;02;12;32), (31;01;02;32;12), \\ (31;01;12;02;32), (31;01;12;32;02) \end{array} \right\}$$

is strongly successful and therefore hard one-step strengthening does not change it — we have $(\mathrm{LNS}^{\blacklozenge})^{\square}(G) = \mathrm{LNS}^{\blacklozenge}(G)$. On the other hand, consider

$$\mathrm{LNS}^{\square}(G) = \left\{ \begin{array}{l} (01;02;12), (01;12;02), (01;31;02;12), (01;31;02;32), \\ (01;31;12;32;02), (01;31;32;12;02), (12;01), (12;31), \\ (31;01;02;12;32), (31;01;12;02;32), (31;01;32;02), \\ (31;01;32;12), (31;12;32), (31;32;12) \end{array} \right\}$$

and note that this is not a superset of $(\mathrm{LNS}^{\blacklozenge})^{\square}(G) = \mathrm{LNS}^{\blacklozenge}(G)$, because we have $(01;31;12;02;32) \in (\mathrm{LNS}^{\blacklozenge})^{\square}(G) = \mathrm{LNS}^{\blacklozenge}(G)$ but $(01;31;12;02;32) \notin \mathrm{LNS}^{\square}(G)$.

Together, we have $\mathrm{LNS}^{\blacklozenge}(G) \subseteq \mathrm{LNS}(G)$ but $(\mathrm{LNS}^{\blacklozenge})^{\square}(G) \not\subseteq \mathrm{LNS}^{\square}(G)$.

Hence $Q = \mathrm{LNS}^{\blacklozenge} \subseteq \mathrm{LNS} = P$ is a counterexample and $(\cdot)^{\square}$ is not monotone.

(ii) *Hard look-ahead strengthening is not monotone*:

For hard look-ahead strengthening we can use the same example. Because $\mathrm{LNS}^{\blacklozenge}$ is strongly successful, hard look-ahead strengthening does not change it: $(\mathrm{LNS}^{\blacklozenge})^{\blacksquare}(G) = \mathrm{LNS}^{\blacklozenge}(G)$. Moreover, we have $\mathrm{LNS}^{\blacksquare}(G) = \{(01), (31)\}$ which is not a superset of $(\mathrm{LNS}^{\blacklozenge})^{\blacksquare}(G) = \mathrm{LNS}^{\blacklozenge}(G)$.

Together we have $\mathrm{LNS}^{\blacklozenge}(G) \subseteq \mathrm{LNS}(G)$ but $(\mathrm{LNS}^{\blacklozenge})^{\blacksquare}(G) \not\subseteq \mathrm{LNS}^{\blacksquare}(G)$, hence hard look-ahead strengthening is not monotone either. $\square$

This result is relevant for our pursuit to pin down what it means to commonly know a protocol. It shows that hard look-ahead strengthening is not rational, as follows.

We consider again the "spaceship" graph in the proof of Proposition 6.4.15. Let us, using the language of game theory, define a *losing move* as a call after which no successful continuation is possible. The initial call could be 12, but that is a losing move. All successful, now also called winning, LNS sequences on this graph start with $01;31$ or $31;01$.

Let us place ourselves in the position of agent 3 after one call has been made. As far as 3 can tell (if the only background common knowledge is that everyone follows LNS), the first call may have been 12, at which point no agent can make a winning move (no continuation is successful). In particular, the second call 31 is then losing. So 3 will not call 1, because it is possible that the call 31 is losing, and we are following hard look-ahead.

Symmetrically, the same reasoning is made by agent 0: even if the first call is 31, it could also have been 12, after which any continuation is unsuccessful, and therefore 0 will not call 1, which again seems irrational.

So nobody will make a call. The extension of $\text{LNS}^{\blacksquare}$ on this graph is empty.

But as all agents know that 12 is losing, agent 1 knows this in particular, and as agent 1 is rational herself, she would therefore not have made that move. And agents 3 and 0 can draw that conclusion too. It therefore seems after all irrational for 3 not to call 1, or for 0 not to call 1.

This shows that hard look-ahead strengthening is not rational. In particular, it ignores the rationality of other agents.

6.4.5 Limits and Fixpoints of Strengthenings

Given the iteration of strengthenings we discussed in the previous section, it is natural to consider limits and fixpoints of strengthening procedures. In this subsection we discuss them and give some small results. A detailed investigation is deferred to future research.

Note that the protocol conditions of all four basic syntactic strengthenings are conjunctions with the original protocol condition as a conjunct. Therefore, all these four strengthenings are *decreasing*: For all $\heartsuit \in \{\blacksquare, \blacklozenge, \Box, \Diamond\}$ and all protocols P, we have $P^{\heartsuit} \subseteq P$. The same holds, by definition, for semantic strengthenings. This implies that if, on any gossip graph, we start with a protocol that only allows finite call sequences, such as LNS, then applying strengthening repeatedly will eventually lead to a fixpoint. This fixpoint might be the empty set, or a non-empty set and thereby provide a new protocol.

For other protocols that allow infinite call sequences, such as ANY, we do not know if this procedure leads to a unique fixpoint and whether fixpoints are always reached. We therefore distinguish fixpoints from limits.

6.4.16. Definition. Consider any strengthening $(\cdot)^{\heartsuit}$. The $\heartsuit$-*limit* of a given protocol P is the semantic protocol $P^{\heartsuit *}$ defined as $\bigcap_k P^{\heartsuit k}$. A given protocol P is a *fixpoint* of a strengthening $(\cdot)^{\heartsuit}$ iff $P = P^{\heartsuit}$.

Note that limit protocols $P^{\heartsuit *}$ are *not* in the logical language, unlike their constituents $P^{\heartsuit k}$. We now define $P^{\Box *}$ as *Hard Uniform Backward Induction*, and $P^{\Diamond *}$ as *Soft Uniform Backward Induction*. Again using induction on Theorem 6.4.7, it follows that Uniform Backward Induction is the same as arbitrarily often iterated Uniform Backward Defoliation.

6.4.17. Corollary.

$$P^{\Box *} = P^{\mathsf{HUBD}*} \text{ and } P^{\Diamond *} = P^{\mathsf{SUBD}*}.$$

6.4.18. Example. Consider P = LNS. The number of LNS calls between n agents is bounded by $\binom{n}{2} = n(n-1)/2$. The limit $\text{LNS}^{\heartsuit *}$ is therefore reached after a finite number of iterations, and expressible in the gossip protocol language: $\text{LNS}^{\heartsuit n(n-1)/2} = \text{LNS}^{\heartsuit *}$.

As a further observation, the look-ahead strengthenings are not always the limits of one-step strengthenings. In other words, we do *not* have for all G that $P^{\Box *}(G) = P^{\blacksquare}(G)$ or that $P^{\Diamond *}(G) = P^{\blacklozenge}(G)$. Counterexamples are the "N" graph from Example 6.4.11 and the extension of various strengthenings relating to the example in the upcoming Section 6.4.6, as shown in Table 6.3.

However, we know by the Knaster-Tarski theorem that on any gossip graph soft one-step strengthening $(\cdot)^{\Diamond}$ has a unique greatest fixpoint, because $(\cdot)^{\Diamond}$ is monotone and the lattice we are working in is the powerset of the set of all call sequences and thereby complete. We leave a detailed analysis of infinite protocols with such algebraic methods for another occasion.

6.4.6 Detailed Example: the Diamond Gossip Graph

Consider the initial "diamond" gossip graph in Figure 6.3.

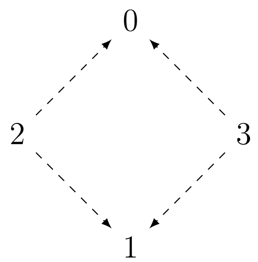


Figure 6.3: The "diamond" example for four agents.

There are 92 different terminating sequences of LNS calls for this initial graph of which 48 are successful and 44 are unsuccessful. Table 6.1 gives an overview of all sequences. For brevity we only list them in the compact way, up to the call after which success has been decided.

$20;01$	$\times$	$21;10$	$\times$	$30;01$	$\times$	$31;10$	$\times$
$20;21$	$\times$	$21;20$	$\times$	$30;20;01$	✓	$31;20$	✓
$20;30;01$	✓	$21;30$	✓	$30;20;21$	✓	$31;21;10$	✓
$20;30;21$	$\times$	$21;31;10$	✓	$30;20;31$	$\times$	$31;21;20$	✓
$20;30;31$	✓	$21;31;20$	$\times$	$30;21$	✓	$31;21;30$	$\times$
$20;31$	✓	$21;31;30$	✓	$30;31$	$\times$	$31;30$	$\times$

Table 6.1: All LNS sequences for the diamond example up to decision moments.

Table 6.2 shows how many sequences are still allowed by the different strengthenings. Both one-step strengthenings and the soft look-ahead strengthening rule out some, but not all, unsuccessful sequences while keeping all successful sequences. The hard look-ahead strengthening however, also removes some successful sequences while still keeping the same number of unsuccessful sequences as the soft strengthening. Interestingly, those are not the same sequences, which is not visible in Table 6.2.

Protocol	# successful	# unsuccessful
LNS	48	44
$\text{LNS}^{\blacksquare}$	8	8
$\text{LNS}^{\blacksquare 2}$	0	4
$\text{LNS}^{\blacksquare 3}$	0	0
$\text{LNS}^{\blacklozenge}$	48	8
$\text{LNS}^{\blacklozenge 2}$	48	8
$\text{LNS}^{\blacklozenge 3}$	48	8
$\text{LNS}^{\Box}$	24	36
$\text{LNS}^{\Box 2}$	8	16
$\text{LNS}^{\Box 3}$	8	4
$\text{LNS}^{\Box 4}$	0	4
$\text{LNS}^{\Box 5}$	0	0
$\text{LNS}^{\Diamond}$	48	36
$\text{LNS}^{\Diamond 2}$	48	32
$\text{LNS}^{\Diamond 3}$	48	32
$(\text{LNS}^{\Diamond})^{\Box 3}$	16	0
$((\text{LNS}^{\Diamond})^{\Box})^{\blacksquare}$	16	0

Table 6.2: Statistics for the diamond example.

Only looking at these statistics can be misleading: If a strengthening does not change the *number of* successful and unsuccessful sequences, it might still have *shortened* some sequences. Table 6.3 lists individual sequences, showing for example that $\text{LNS}^{\Diamond 2}$ and $\text{LNS}^{\Diamond 3}$ not only both have 48 successful and 32 unsuccessful sequences on the diamond graph, but that we also have an extensional identity between them. This is therefore a fixpoint of $(\cdot)^{\Diamond}$ on that graph.

Recall that we can identify the one-step strengthening with uniform backward defoliation and thereby the limit of one-step strengthening with uniform backward induction — see Theorem 6.4.7 and Corollary 6.4.17. Table 6.2 serves well to show the difference between the look-ahead strengthenings and the one-step/defoliation strengthenings. Although on this "diamond" graph, the hard strengthenings $\text{LNS}^{\blacksquare k}$ and $\text{LNS}^{\Box k}$ have the same empty extension for all $k \geq 4$, the soft strengthenings $\text{LNS}^{\blacklozenge k}$ and $\text{LNS}^{\Diamond k}$ have different fixpoints. Both are reached at $k = 2$.

We now discuss some strengthenings that are strongly successful on this graph (only successfully terminating call sequences remain).

First, consider the protocol $(\mathrm{LNS}^{\Diamond})^{\Box 3}$. Its extension is as follows, see also Tables 6.2 and 6.3.

$20;30;01;31;21$	$21;30;01;31;20$	$30;20;01;21;31$	$31;20;01;21;30$
$20;30;31;01;21$	$21;30;31;01;20$	$30;20;21;01;31$	$31;20;21;01;30$
$20;31;10;30;21$	$21;31;10;30;20$	$30;21;10;20;31$	$31;21;10;20;30$
$20;31;30;10;21$	$21;31;30;10;20$	$30;21;20;10;31$	$31;21;20;10;30$

Unlike the next strongly successful strengthening its extension has no short sequences with only four calls. Instead, there are redundant second-to-last calls, for example 10 in $20;31;30;10;21$.

Second, we present another protocol that is strongly successful on this graph, that preserves more sequences than the previous protocol $(\mathrm{LNS}^{\Diamond})^{\Box 3}$, but that does not correspond to iteration of the soft or hard one-step protocols discussed up to now. We first describe it as a semantic protocol, liberally referring to call histories in our description (which cannot be done in our logical language) and only then give a formalization using the syntax of our protocol logic. Consider the following protocol:

(1) The left and right agents 2 and 3 both make one call, in any order (say, first 2, then 3).

(2) If they called the same agent (say, 0), then that agent calls the remaining agent (in this case 1), and then 2 and 3 call 1, in the same order as they made the first two calls.

(3) If they called different agents (say 2 called 0 and 3 called 1), then they both call the other one, in the opposite order as the first two calls (so in this case 3 calls 0 and then 2 calls 1).

We need synchronicity to make sure that step 1 is finished before step 2 or 3 is begun.

Moreover, note that only one of 2 and 3, namely the one making the second call, will learn whether (2) or (3) should be done. The protocol is still epistemic, because the agent making the first call simply does the same in both (2) and (3): wait one round and then if they called 0 first, call 1, or vice versa. This can also be seen in the call sequences of this protocol (with the number behind the sequence to indicate which part of the protocol is used):

$20;30;01;21;31$	(2)	$30;20;01;31;21$	(2)
$21;31;10;20;30$	(2)	$31;21;10;30;20$	(2)
$20;31;30;21$	(3)	$30;21;20;31$	(3)
$21;30;31;20$	(3)	$31;20;21;30$	(3)

Finally, we can see that all of these sequences are also LNS sequences, as shown in Table 6.1. Hence this is indeed a semantic strengthening of LNS.

This protocol can also be defined syntactically as follows, though it is rather ugly and we cannot guarantee the exact order of calls.

We can define “no calls have been made” quite easily: $\varphi_0 := \bigwedge_i \bigwedge_{j \neq i} \neg S_i j$. Defining “one call has been made” is a bit harder, but we can do it because after the first call there are two agents that know each other’s secrets, while the remaining agents know only their own: $\varphi_1 := \bigvee_{i,j}(S_i j \wedge S_j i \wedge \bigwedge_{k \notin \{i,j\}} \bigwedge_{l \neq k} \neg S_k l)$.

So the calls allowed by clause (1) are: if you only know your own secret and φ_0 or φ_1 holds, then you may make a call (if you know the number, of course.) Formally, this means that we add a disjunct $\bigwedge_{k \neq i} \neg S_i k \wedge (\varphi_0 \vee \varphi_1)$ to P_{ij}.

Now, consider the calls that have to be made for clause (2): in our language, we cannot distinguish between 0 (the callee) and 3 (the last person to call 0). But that does not matter, since both are supposed to call 1. We define that if you know three secrets, you are allowed to call the final person who’s secret you do not know. This gives us a disjunct $\bigvee_{k,l \notin \{i,j\}} S_i kl$.

This leaves us with agent 2, which first called 0. This agent 2 must make a call to agent 1 (the only agent that 2 can call) after 0 or 3 has made their call. So the call must be made after at least three other calls have been placed. We can formulate a condition φ_3 which holds if at least three calls have been made, but that formula is huge. So it is more convenient to take a shortcut: you are allowed to call someone if you consider it possible that this person is an expert. This yields a disjunct $\hat{K}_i Ex_j$.

Now, consider clause (3). Here agents 3 and 2 need to call 0 and 1, respectively. First, consider the call by 3. This needs to take place if the two agents called different agents. Agent 3 will know that this is the case. So we can create a disjunct that says that you are allowed to make a call if you know that all four agents know two secrets:

$$\bigvee_{i \neq j,\ k \neq l} \begin{pmatrix} & S_i ij \wedge S_j ij \wedge S_k kl \wedge S_l kl \\ \wedge & \neg S_i k \wedge \neg S_j k \wedge \neg S_i l \wedge \neg S_j l \\ \wedge & \neg S_k i \wedge \neg S_l i \wedge \neg S_k j \wedge \neg S_l j \end{pmatrix}$$

Note that not only agent 3 is allowed to make a call based on this, but also agent 1. That is fine, because the extra call does not prevent success.

Finally, agent 2 needs to make a call. That happens through the same disjunct that we discussed in clause (2), namely $\hat{K}_i Ex_j$.

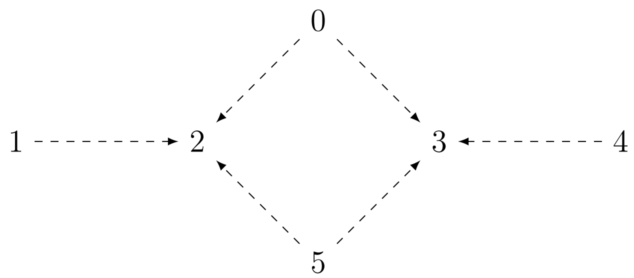
All in all, this gives us the protocol that we need. Admittedly, the manual verification of this protocol is tedious. We therefore also checked that it is strongly successful using the implementation that will describe later in Section 6.6.1.

6.5 Impossibility Result on Strengthening LNS

In this section we will show that there are graphs where (i) LNS is weakly successful and (ii) no epistemic symmetric strengthening of LNS is strongly successful. Recall that we assume that the system is synchronous and that the initial gossip graph is common knowledge. Without such assumptions it is even easier to obtain such an impossibility result, a matter that we will address in Section 6.7.

6.5.1. THEOREM. *There is no epistemic symmetric protocol that is a strongly successful strengthening of* LNS *on all graphs.*

Proof:
Consider the following "candy" graph G:



LNS is weakly successful on G, but there is no epistemic symmetric protocol P that is a strengthening of LNS and that is strongly successful on G.

In [Dit+15], it was shown that LNS is weakly successful on any graph that is neither a "bush" nor a "double bush". Since this graph G is neither a bush nor a double bush, LNS is weakly successful on it. For example, the sequence

$$02; 12; 53; 43; 13; 03; 23; 52; 42$$

is a successful LNS sequence which makes everyone an expert. LNS is not strongly successful on this graph, however. For example,

$$02; 12; 53; 43; 13; 03; 52; 42$$

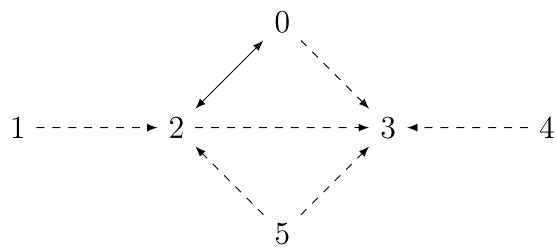
is an unsuccessful LNS sequence, because 5 does neither learn the number nor the secret of 4 and no further calls are allowed.

Now, suppose towards a contradiction that P is an epistemic symmetric strengthening of LNS, and that P is strongly successful on G. Before we look at specific calls made by P, we consider a general fact. Recall that knowing a *pure number* means knowing the number of an agent without knowing their secret. For any gossip graph and any agent a, if no one has a's pure number, then no call sequence will result in anyone learning a's pure number. After all, in order to learn a's number, one would have to call or be called by someone who already knows that number, but in such a call one would also learn a's secret.

In LNS, you are only allowed to call an agent if you have the number but not the secret of that agent, i.e., if you have their pure number. It follows that if, in a given gossip graph, no one has a's pure number, then no LNS sequence on that graph will contain any calls where a is the receiver.

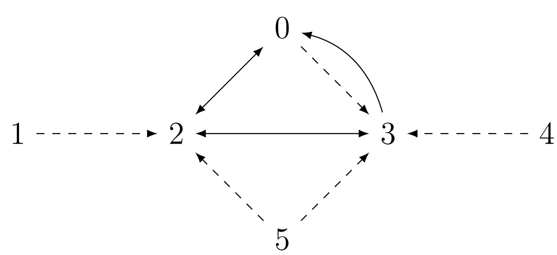
In the gossip graph G under consideration, agents 0, 1, 4 and 5 are in the situation that no one else knows their number. So in particular, no one knows the pure number of any of these agents. It follows that 2 and 3 are the only possible targets for LNS calls in this graph.

Now, let us consider the first call according to P. This call must target 2 or 3. The calls 12 and 43 are losing moves, since they would result in 1 (resp. 4) being unable to make calls or be called, while still not being an expert. This means that either 0 or 5 must make the first call. By symmetry, we can assume without loss of generality that the first call is 02. This yields the following situation.



Now, let us look at the next call.

- The sequence $02; 43$ is losing, because afterwards 4 cannot become an expert.
- Because of the symmetry of P, the initial call could have been 03 instead of 02. The sequence $03; 12$ is losing, since 1 cannot become an expert, so $03; 12$ is not allowed by the strongly successful protocol P. Moreover, agent 1 cannot tell the difference between 03 and 02, so from the fact that $03; 12$ is disallowed and that P is epistemic, it follows that $02; 12$ is also disallowed.
- The sequence $02; 03$ is losing, since 0 will not be able to make any call afterwards. As 0 can never be called, this implies that 0 will never become an expert.
- Consider then the sequence $02; 23$. This results in the following diagram.



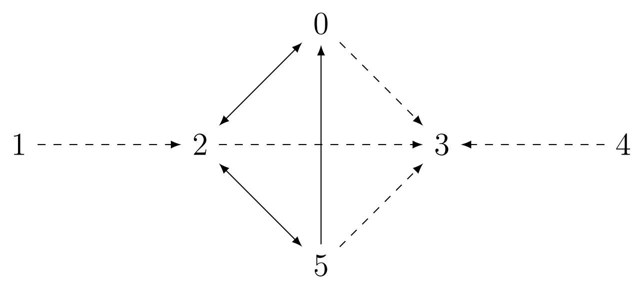
This graph has the following property: it is impossible (in any LNS sequence) for any agent to get to learn a new pure number. That is, nobody can learn a new number without also getting to know the secret of that agent: agents 1, 0, and 4 each know only one pure number, so they cannot teach anyone a new number, and agent 5 knows two pure numbers (2 and 3), but those agents already know each other's secrets.

As a result, any call that will become allowed by LNS in the future is already allowed now. There are 5 such calls that are currently allowed, namely 12, 52, 53, 03 and 43. Furthermore, of those calls 52 and 53 are mutually exclusive, since calling 2 will teach 5 the secret of 3, and calling 3 will teach 5 the secret of 2.

So any continuation of 02; 23 allowed by LNS can only contain (in any order) 12, 03, 43 and either 52 or 53. Since P is a strengthening of LNS, the same holds for P. But using only those calls, there is no way to teach 3 the secret of 1: secret 1 can reach agent 2 using the call 12, but in order for the secret to travel any further we need the call 52. After that call only 03 and 43 are still allowed (in particular, 53 is ruled out), so the knowledge of secret 1 remains limited to agents 1, 2 and 5.

Since 02;13 cannot be extended to a successful LNS sequence, 02;13 must be disallowed.

- Consider the call sequence 02; 52. This gives the following diagram.



Note that in this situation, it is impossible for agents 3 and 4 to learn any new number without also learning the secrets corresponding to those numbers: there is no agent that knows the number of agent 3 and that also knows another pure number, and this will remain the case whatever other calls happen.

This means that agent 3 cannot make any calls, and that agent 4 can make exactly one call, to agent 3.

Suppose now that 02; 52 is extended to a successful LNS sequence. This sequence has to contain the call 43 at some point. This will be the only call by agent 4, so in order for the sequence to be successful, agent 3 already has to know secret 1 by the time 43 takes place.

In particular, this means that the call 12 has already happened, and that either agent 1 or agent 2 has then called agent 3 to transmit this secret. Whichever agent among 1 and 2 makes this call, afterwards they are unable to make any more calls. Furthermore, this takes place before the call 43, so whatever agent $x \in \{1, 2\}$ informs 3 of secret 1 does not learn secret 4. Since this agent x can neither make another call nor be called, it follows that x does not become an expert.

So $02; 52$ is not allowed by P which we assumed to be strongly successful.

- Finally, consider the call sequence $02; 53$. By symmetry, 03 could have been the first call as opposed to 02. Furthermore, the same reasoning that showed 02;52 to be unsuccessful above can, with an appropriate permutation of agents, be used to show that 03;53 is unsuccessful. Agent 5 cannot distinguish between the first call 02 and 03 before making the call 53, so if $03; 53$ is disallowed then so is $02; 53$ because P is epistemic.

Remember that 02 is, without loss of generality, the only initial call that can lead to success. We have shown that all of the LNS-permitted calls following the initial call 02 (namely, the calls 43, 12, 03, 23, 52 and 53) are disallowed by P. This contradicts P being a strongly successful strengthening of LNS. □

Given this impossibility result, it is natural to wonder what would happen if we use the syntactic strengthenings from Definition 6.4.4, or their iterations, on the "candy" graph G.

All second calls are eliminated by $\text{LNS}^{\blacksquare}$, because for any two agents a and b we have $G, 02 \vDash \neg K_a^{\text{LNS}}[ab]\langle\text{LNS}\rangle Ex$. By symmetry this also holds for the three other possible first calls, hence $\text{LNS}^{\blacksquare}$ is unsuccessful on G. However, the first calls *are* still allowed according to $\text{LNS}^{\blacksquare}$.

There are 9468 LNS-sequences on this graph of which 840 are successful. With the implementation discussed Section 6.6.1, we found out that $\text{LNS}^{\blacklozenge}$, the soft look-ahead strengthening of LNS, is weakly successful on this graph and allows 840 successful and 112 unsuccessful sequences.

6.6 Model Checking for Dynamic Gossip

Analyzing examples of gossip graphs and execution trees by hand is tedious. In this section we will explore different ways to automate the analysis of gossip graphs, calls and protocols. We start with an implementation of explicit state model checking. Then we discuss how to represent gossip in standard DEL and give a new symbolic representation using our framework from Chapters 2 and 3.

6.6.1 An Explicit Implementation

To help us find and check all the examples in the previous sections we wrote an explicit model checker. Like SMCDEL, it is written in Haskell. The sources can be found at `https://github.com/m4lvin/gossip`.

Our program can show and randomly generate gossip graphs, execute the protocols we discussed and draw the resulting execution trees with epistemic edges. The program also includes an epistemic model checker for the formal language we introduced, similar to DEMO-S5, but tailor-made for dynamic gossip. Another similar implementation, though only for static gossip, is the *EGP* tool from [Att+15] and [Att15]. It was only recently made public at `https://github.com/mdk333/EGPTool` and we leave a comparison between this tool and our implementations as future work.

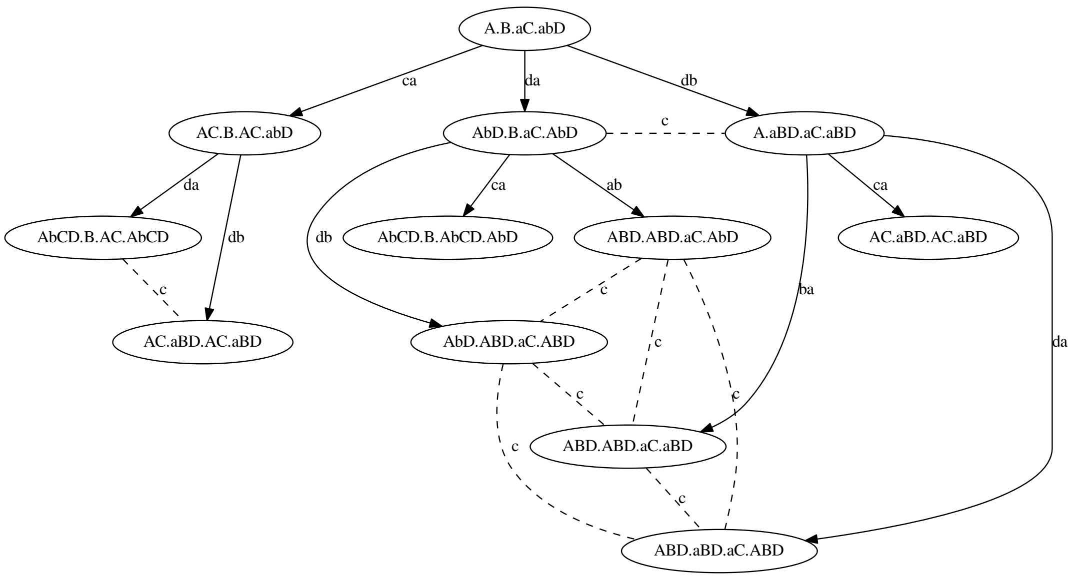


Figure 6.4: `dispTreeWith [2] 2 1 lns (tree lns (nExample,[]))`.

Figure 6.4 is an example output of the implementation, showing the execution tree for Example 6.4.11 up to two calls, together with the epistemic relations for agent 2, here called c. Note that we use a more compact way to denote gossip graphs: lower case stands for a pure number and capital letters for knowing the number and secret.

We observe that after da and db, agent c considers two alternatives, but this indistinguishability disappears when c makes the second call. This is because c will either learn the secret of a or the secret of b and thereby learn with whom of these c talked before. This shows that there is no “No Learning” axiom in gossip, because agents do learn something about the past when they make calls. Similarly, “No Miracles” is not valid in gossip either.

The implementation also includes an automated test module, to check all examples we used in the previous sections. Similar to the tests for SMCDEL described in Section 3.10, it is based on QuickCheck and Hspec.

Our implementation can run different protocols on a given graph and output a LaTeX table showing and comparing the extension of those protocols. Tables 6.4 and 6.3 have been generated in this way. They provide details how various strengthenings behave on the gossip graphs from Example 6.4.11 and 6.4.6.

	LNS	$.\blacksquare$	$(.\blacksquare)^{\Box}$	$.\blacklozenge$	$.\Box$	$.\Box 2$	$.\Box 3$	$.\Box 4$	$.\Diamond$	$.\Diamond 2$	$.\Diamond 3$	$(.\Diamond)^{\Box 3}$
ϵ								×				
01						×						
01;21					×				×	×	×	
01;21;30	×											
01;21;31	×											
01;30					×							
01;30;21	×								×	×	×	
01;31					×							
01;31;21	×								×	×	×	
21						×						
21;01					×				×	×	×	
21;01;30	×											
21;01;31	×											
21;30										×	×	
21;30;01					×				×			
21;30;01;31	×											
21;30;31					×				×			
21;30;31;01	×											
21;31					×							
21;31;01	×								×	×	×	
30			×				×					
30;01		×				×						
30;01;21;31	✓			✓					✓	✓	✓	
30;01;31;21	✓			✓	✓				✓	✓	✓	✓
30;21;01					×							
30;21;01;31	×			×					×	×	×	
30;21;31					×							
30;21;31;01	×			×					×	×	×	
30;31		×				×						
30;31;01;21	✓			✓	✓				✓	✓	✓	✓
30;31;21;01	✓			✓					✓	✓	✓	
31;01;21;30	✓			✓					✓	✓	✓	
31;01;30;21	✓			✓	✓				✓	✓	✓	
31;10;21;30	✓			✓					✓	✓	✓	
31;10;30;21	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
31;21;01;30	✓			✓					✓	✓	✓	
31;21;30	✓			✓	✓				✓	✓	✓	
31;30;10;21	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
31;30;21	✓			✓					✓	✓	✓	

Table 6.3: Diamond Example 6.4.6: Extensions of strengthenings, after 20.

	LNS	$.\blacksquare$	$.\blacklozenge$	$.\Box$	$.\Box 2$	$.\Box 3$	$.\Box 4$	$.\Diamond$	$.\Diamond 2$	$.\Diamond 3$	$.\Diamond 4$	$.\Diamond 5$
ϵ							×					
20						×				×		
20;30					×				×			
20;30;01				×				×				
20;30;01;31	×											
20;30;31				×				×				
20;30;31;01	×											
20;31					×				×			
20;31;10				×				×				
20;31;10;30	×											
20;31;30				×				×				
20;31;30;10	×											
30		×				×						
30;01					×							
30;01;20				×								
30;01;20;31	×							×	×	×	×	×
30;01;31				×				×	×	×	×	×
30;01;31;20	×											
30;20;01					×							
30;20;01;21;31	✓		✓	✓				✓	✓	✓	✓	✓
30;20;01;31;21	✓		✓					✓	✓	✓	✓	✓
30;20;21					×							
30;20;21;01;31	✓		✓	✓				✓	✓	✓	✓	✓
30;20;21;31;01	✓		✓					✓	✓	✓	✓	✓
30;20;31;01				×								
30;20;31;01;21	×							×	×	×	×	×
30;20;31;21				×								
30;20;31;21;01	×							×	×	×	×	×
30;31					×							
30;31;01				×				×				
30;31;01;20	×											
30;31;20									×	×	×	×
30;31;20;01				×				×				
30;31;20;01;21	×											
30;31;20;21				×				×				
30;31;20;21;01	×											
31						×						
31;10					×							
31;10;20				×				×	×	×	×	×
31;10;20;30	×											
31;10;30				×				×				
31;10;30;20	×											
31;20					×				×	×	×	×
31;20;10				×				×				
31;20;10;30	×											
31;20;30				×				×				
31;20;30;10	×											
31;30					×							
31;30;10				×				×				
31;30;10;20	×											
31;30;20				×				×	×	×	×	×
31;30;20;10	×											

Table 6.4: N Example 6.4.11: Extensions of strengthenings.

6.6.2 Gossip in Standard DEL

In the previous sections we used a special-purpose language and semantics to analyze the gossip problem. And while our explicit model checker is similar to DEMO-S5, it is not compatible with the symbolic methods we developed in Chapters 2 and 3. Hence we would also like to see which parts of the gossip problem we can model in standard DEL, based on propositional logic and interpreted in Kripke models or knowledge structures.

An analysis of gossip in DEL with action models is [Att+14]. In Section 6.6.4 we will look at the action models provided there. In Section 6.6.5 we then define a symbolic abstraction of these models to speed up model checking tasks.

In contrast to the previous sections, we now restrict ourselves to "static" gossip where N is the total relation and only secrets and no numbers are exchanged.

When a call happens, agents can learn various facts: Who is in the call? Which secrets are exchanged? What is their value? The answers differ, depending on which observations we allow agents who do not participate in the call to make. Moreover, the number of different possible events depends on how secrets and the knowledge thereof are represented.

The question who is in the call gives us a factor of $n(n-1)$: all possible calls are different events. In static gossip we can identify calls ab and ba, which reduces the factor to $\frac{n \cdot (n-1)}{2} = \binom{n}{2}$.

Before we consider the other two questions, we need to be more precise about the knowledge of secrets, which is also a good occasion to explain the differences between our gossip models and [Att+14].

6.6.3 Knowing-whether, Knowing-that, Atomic-knowing

We can distinguish at least three ways of modeling gossip. First, what we have done in the previous sections is to model that a knows the secret of b as an atomic proposition $S_a b$. We can call this the "atomic-knowing" modeling. Second, the "knowing-whether" approach in [Att+14] models secrets themselves as atomic propositions which can be true or false.

In between the two is a third approach, which we now call "knowing-that" where secrets are only exchanged if they are true. This also comes natural, because some gossip is only interesting if it is true: "Did you know that he does *not* have a cat?!" is not a very exciting thing to say.

The three languages are compared in Table 6.5. Clearly, atomic-knowing is more concise, but we can only use it if we do not care about the value of secrets.

The reader will also notice that these different models of the static gossip problem are similar to the different representations for knowledge of numeric variables discussed in Chapter 5. In particular, we can compare Table 6.5 to Table 5.3: Just like PIL is an abstraction that ignores the values of numeric variables, atomic-knowing gossip is an abstraction that ignores the values of secrets. And again, this abstraction trades expressivity for succinctness. In the next sections we will see that this is not only the case for formulas, but also for the size of action models and transformers to describe events.

	a knows b's secret	c knows that a knows b's secret
knowing-whether	$K_a p_b \vee K_a \neg p_b$	$K_c(K_a p_b \vee K_a \neg p_b)$
knowing-that	$K_a p_b$	$K_c(p \to K_a p_b)$
atomic-knowing	$S_a b$	$K_c S_a b$

Table 6.5: Three formal languages for gossip.

6.6.4 Action Models for Gossip

In [Att+14] an analysis of the static gossip problem is given using action models. In contrast to our models, knowledge of secrets there is modeled using standard propositions and knowing-whether. Moreover, it is not assumed that there is a bijection between secrets and agents and that each agent knows only their own secret in the beginning. The authors discuss three sorts of gossip calls, depending on what agents that are not in the call can observe. The resulting action models differ in their specific size, but are all in the same order of magnitude, as Table 6.6 shows.

Symbol	Name	Number of actions
ab^-	observable	$\mathcal{O}(2^{4n})$
ab^0	synchronous	$\mathcal{O}(2^{4n}) \cdot \binom{n}{2} = \mathcal{O}(2^{4n})$
ab^+	skip-async	$\mathcal{O}(2^{4n}) \cdot \binom{n}{2} + 1 = \mathcal{O}(2^{4n})$

Table 6.6: Size of action models for gossip calls.

A call ab corresponds to multiple events in these action models. Each event not only describes who is in the call, but also which information is exchanged. This is achieved with preconditions of the form $\delta(Q_a^+, Q_a^-, Q_b^+, Q_b^-)$ with four parameters saying which secrets are known to be true or false by a or b. For example, Q_a^+ consists of those variables which a knows to be true before the call. Each parameter can be any subset of the set of secrets, hence for n secrets we get 2^{4n} different preconditions.

In these action models, two events are indistinguishable for an agent iff either (i) the agent is not in both calls or (ii) the agent is in both calls AND observes the same, i.e. there is the same other agent in both calls and both agents in the call knew the same set of secrets before.

Can we simplify the action models by saying that secrets are only exchanged if they are true, i.e. by moving from knowing-whether to knowing-what gossip?

The answer is positive, but not satisfactory: The preconditions would then be of the form $\delta(Q_a^+, Q_b^+)$ which reduces the size of the action models, but they are still exponential. In the next section we therefore go back to the atomic-knowing modeling.

6.6.5 Symbolic Gossip

We now discuss how the "atomic-knowing" version of gossip can be modeled symbolically using knowledge transformers with factual change as presented in Definition 2.8.5. There are two key ideas that lead us to a very compact modeling.

First, in atomic-knowing gossip, *exchange of secrets is factual change.* Hence, instead of multiple events with different preconditions, we can use a single event with postconditions that describe which secrets are exchanged. In an action model the call ab is then represented by one event with the postconditions $\mathsf{post}_{ab}(S_a b) := \top$ and $\mathsf{post}_{ab}(S_a c) := S_a c \lor S_b c$ etc.

Second, *postconditions are common knowledge, but not what they evaluate to.* This means that we can ensure that agents in the call observe more than agents who do not participate, without needing multiple events for the same call. Instead, the uncertainty about which call happens induces uncertainty about what other agents learn. Note that here we use our assumption that the initial graph is common knowledge. Therefore, all uncertainty about what agents in other calls are exchanging comes from uncertainty about previous calls.

More formally, suppose some agent is not in a call. In knowing-whether gossip it will consider only those preconditions that are true at some possible world it considers before the call. Each world can only fulfill one of the mutually exclusive preconditions. Similarly, in atomic-knowing gossip we have one event which applies to all those possible worlds, and the postcondition will be evaluated differently but deterministically in each one. New uncertainty is always about who-called-whom, and not about what they exchanged.

The initial situation for atomic-knowing gossip with n agents is given by the knowledge structure

$$\mathcal{F}_{\text{init}} = (V = \{S_i j \mid i, j \in I, i \neq j\},\ \theta = \bigwedge_{i \neq j} \neg S_{i,j},\ O_i = \varnothing)$$

and the actual state $\varnothing$.

We now model calls using a knowledge transformer with factual change. As we are dealing with static total-graph gossip for now, we can identify the calls (a, b) and (b, a). The atoms in the following event vocabulary V^+ describe which call happens: $q_{i,j}$ is an element of the actual event iff the call (i, j) or (j, i) happens. For each agent k, let $\varphi_k := \bigvee(\{q_{i,k} \mid k \in I, i < k\} \cup \{q_{k,j} \mid j \in I, k < j\})$. This abbreviation says that k participates in a call.

We show the call transformer $\mathcal{X}_{\text{call}}$ in Figure 6.5.

The event law θ^{+} ensures that some call happens, but excludes that two calls happen at the same time.

The factual change encoded by θ_{-} says that after a call, i has the secret of j iff either i already knew it, or i and j are both in the call or i is in the call and there is some k in the call who knew j. While this is quite a complex boolean formula, in the implementation it will be a BDD of reasonable size. For example, if we consider four agents, the BDD of $\theta_(S_0 1)$ has 21 non-terminal nodes.

$$\left(\begin{array}{lcl} V^{+} & = & \{q_{i,j} \mid i,j \in I, i < j\} \\ \theta^{+} & = & \bigvee_{i<j} q_{i,j} \land \bigwedge\{\neg(q_{i,j} \land q_{k,l}) \mid i,j,k,l \in I, i<j, k<l, (i,j) \neq (j,k)\} \\ V_{-} & = & V \\ \theta_{-} & : & S_{i,j} \mapsto S_{i,j} \lor (\varphi_i \land \varphi_j) \lor (\varphi_i \land \bigvee_k (\varphi_k \land S_{k,j})) \\ O^{+} & = & \{q_{i,k}\} \cup \{q_{k,j}\} \end{array}\right)$$

Figure 6.5: The transformer $\mathcal{X}_{\text{call}}$.

Figure 6.6 shows how the initial knowledge structure for gossip can be implemented in SMCDEL and Figure 6.7 shows the implementation of the transformer $\mathcal{X}_{\text{ab}}$ together with some helper function to simplify its usage. We also include short comments to highlight the connection to the mathematical definitions in Figure 6.5 and our explanations of θ^{+} and θ_{-}.

One might also consider describing the events with less propositions, namely just q_i to say that i participates in the call. But someone in the actual call also observes which other person participates. Hence only using propositions q_i makes it impossible to encode the event observations with a simple set of observational variables and forces us to use belief structures, even though the relations here are all equivalences.

```
gossipers :: Int -> [Int]
gossipers n = [0..(n-1)]

hasSof :: Int -> Int -> Int -> Prp
hasSof n a b | a == b    = error "Let's not even talk about that."
             | otherwise = toEnum (n * a + b)

gossipInit :: Int -> KnowScene
gossipInit n = (KnS vocab law obs, actual) where
  vocab  = [ hasSof n i j | i <- gossipers n, j <- gossipers n, i /= j ]
  law    = boolBddOf $ Conj [ Neg $ PrpF $ hasSof n i j
                            | i <- gossipers n, j <- gossipers n, i /= j]
  obs    = [ (show i, []) | i <- gossipers n ]
  actual = [ ]
```

Figure 6.6: $\mathcal{F}_{\text{init}}$ in `SMCDEL.Examples.GossipS5`.

```haskell
thisCallProp :: (Int,Int) -> Prp
thisCallProp (i,j) | i < j     = P (100 + 10*i + j)
                   | otherwise = error $ "wrong call: " ++ show (i,j)

call :: Int -> (Int,Int) -> Event
call n (a,b) = (callTrf n, [thisCallProp (a,b)])

callTrf :: Int -> KnowChange
callTrf n = CTrf eventprops eventlaw changeprops changelaws eventobs where
  thisCallHappens (i,j) = PrpF $ thisCallProp (i,j)
  isInCallForm k = Disj $
    [ thisCallHappens (i,k) | i <- gossipers n \\ [k], i < k ] ++
    [ thisCallHappens (k,j) | j <- gossipers n \\ [k], k < j ]
  allCalls = [ (i,j) | i <- gossipers n, j <- gossipers n, i < j ]
  eventprops = map thisCallProp allCalls
  eventlaw = simplify $
    Conj [ Disj (map thisCallHappens allCalls)
         -- some call must happen, but never two at the same time:
         , Neg $ Disj [ Conj [thisCallHappens c1, thisCallHappens c2]
                      | c1 <- allCalls, c2 <- allCalls \\ [c1] ] ]
  callPropsWith k = [ thisCallProp (i,k) | i <- gossipers n, i < k ]
                 ++ [ thisCallProp (k,j) | j <- gossipers n, k < j ]
  eventobs = fromList [(show k, callPropsWith k) | k <- gossipers n]
  changeprops = keys changelaws
  changelaws = fromList
    [(hasSof n i j, boolBddOf $             -- after call, i has secret j
        Disj [ has n i j                    -- iff i already knew j, or
             , Conj (map isInCallForm [i,j]) -- i and j are both in call or
             , Conj [ isInCallForm i         -- i is in call and some k who
                    , Disj [ Conj [ isInCallForm k, has n k j ] -- knew j
                           | k <- gossipers n \\ [j] ] ]
             ])
    | i <- gossipers n, j <- gossipers n, i /= j ]

doCall :: KnowScene -> (Int,Int) -> KnowScene
doCall start (a,b) = knowChange start (call (length $ agentsOf start) (a,b))

after :: Int -> [(Int,Int)] -> KnowScene
after n = foldl doCall (gossipInit n)
```

Figure 6.7: $\mathcal{X}_{ab}$ in SMCDEL.Examples.GossipS5.

6.7 Conclusion and Future Work

We modeled common knowledge of protocols in the setting of distributed dynamic gossip. A crucial role is played by the novel notion of protocol-dependent knowledge. This knowledge is interpreted using an equivalence relation over states in the execution tree of a gossip protocol in a given gossip graph. As the execution tree consists of gossip states resulting from calls permitted by the protocol, this requires a careful semantic framework. We described various syntactically or semantically definable strengthenings of gossip protocols, and investigated the combination and iteration of such strengthenings, in view of strengthening a weakly successful protocol into one that is strongly successful on all graphs. In the setting of gossip, a novel notion we used in such strengthenings is that of uniform backward induction, as a variation on backward induction in search trees and game trees. Finally, we proved that for the LNS protocol, in which agents are only allowed to call other agents if they do not know their secrets, it is impossible to define a strengthening that is strongly successful on all graphs.

As already described at length in the introductory section, our work builds upon prior work on dynamic distributed gossip [Dit+15; Dit+17], which itself has a prior history both in the networks community [HLL99; Kar+00; Hae15] and in the logic community [Att+14; AGH15]. Many aspects of gossip may or may not be common knowledge among agents: how many agents there are, the time of a global clock, the gossip graph, etc. The point of our result is that even under the strongest such assumptions, one can still not guarantee that a gossip protocol always terminates successfully. How common knowledge of agents is affected by gossip protocol execution is investigated in [AW17]: for example, the authors demonstrate how sender-receiver subgroup common knowledge is obtained (and lost) during calls. However, they do not study common knowledge of gossip protocols. We do not know of other work on that topic. Outside the area of gossip, protocol knowledge has been well investigated in the epistemic logic community [Hos09; Wan10; Dit+14].

While the concept of backward induction is well-known in game theory, it is only used in perfect-information settings, where all agents know what the real world or the actual state is. Our definition of *uniform* backward induction is a generalization of backward induction to the dynamic gossip setting, where only partial observability is assumed. A concept akin to uniform backward induction has been proposed in [Per14] (rooted in [BS02]), under the name of *common belief in future rationality*, with an accompanying recursive elimination procedure called *backward dominance*.[2] As in our approach, this models a decision rule faced with uncertainty over indistinguishable moves.

In [Per14], the players are utility maximizers with probabilistic beliefs, which in our setting would correspond to *randomizing* over all indistinguishable moves/calls. As a decision rule this is also known as the *insufficient reason* (or *Laplace*) criterion: all outcomes are considered equiprobable. Seeing uniform backward induction as the combination of backward induction and a decision rule immediately clarifies the picture. Soft uniform backward induction applies the *minimax regret* criterion for the decision whom to call, minimizing the maximum utility loss. In contrast, hard uniform backward induction applies the *maximin utility* criterion, maximizing the minimum utility (also known as risk-averse, pessimistic, or Wald criterion). In the gossip scenario, the unique minimum value is unsuccessful termination, and the unique maximum value is successful termination. Minimax prescribes that as long as the agent considers it possible that a call leads to successful termination, the agent is allowed to make the call (as long as the minimum of the maximum

[2] We kindly thank Andrés Perea for his interactions.

is success, go for it): the soft version. Maximin prescribes that, as long as the agent considers it possible that a call lead to unsuccessful termination, the agent should not make the call (as long as the maximum of the minimum is failure, avoid it): the hard version. Such decision criteria over uncertainty also crop up in areas overlapping with social software and social choice, e.g. [BSZ09; CWX11; PTW13; Mei15]. In [BSZ09] a somewhat similar concept has been called "common knowledge of stable belief in rationality". However, there it applies to a weaker epistemic notion, namely belief.

The impossibility result for LNS is for dynamic gossip wherein agents exchange both secrets and numbers, and where the network expands. Also in the non-dynamic setting we can quite easily find a graph where static LNS is weakly successful but cannot be (epistemically and symmetrically) strengthened to a strongly successful protocol. Consider again the "diamond" graph of Section 6.4.6, for which we described various strongly successful strengthenings. Also in "static" gossip LNS is weakly successful on this graph, since $01; 30; 20; 31$ is successful. All four possible first calls are symmetric. After 21, the remaining possible calls are 20, 31 and 30. But 20 is losing, since 2 will never learn secret 3 that way. Also 31 is losing, since agent 1 will never learn the secret of 0. The call 30 is winning, but by epistemic symmetry it cannot be allowed while 31 is disallowed. Therefore, it is impossible to strengthen LNS on "diamond" such that it becomes strongly successful. We can thus expect a completely different picture for strengthening "static" gossip protocols in similar fashion as we did here, for dynamic gossip.

We assumed synchronicity (a global clock) and common knowledge of the initial gossip graph. These strong assumptions were made on purpose, because without them agents will have even less information available and will therefore not be able to coordinate any better. Such and other parameters for gossip problems are discussed in [Dit+16]. It is unclear what results still can be obtained under fully distributed conditions, where agents only know their own history of calls and who their neighbors are.

We wish to determine the logic of protocol-dependent knowledge K_a^P, and also on fully distributed gossip protocols, without a global clock, and to further generalize this beyond the setting of gossip.

Our explicit state implementation covers both static and dynamic gossip, but the implementation of gossip in SMCDEL so far only covers static gossip and misses many features of the explicit checker. In the future we also hope to symbolically verify dynamic gossip protocols.

Another recent formalization of dynamic gossip which could help with automated checking of protocols is [Wag17]. It gives a translation of various properties of gossip graphs and the success of the LNS protocol to *NetKAT*, a network programming language based on Kleene algebras with tests. This paves another way for an automated analysis of gossip protocols, and its performance should be compared with our explicit and symbolic approaches.

Conclusion

> Het mooie van logica, van wetenschap in het algemeen, is dat het groter is dan jezelf: je kunt je er altijd in blijven ontwikkelen.
>
> Jan van Eijck

Coming to the end of this thesis, we give a summary of our work. How did we answer our research questions? Which new concepts did we introduce? Which results did we show? Which open questions remain?

Summary

We started our investigations with an overview of the standard frameworks for Epistemic Logic, Public Announcement Logic, Dynamic Epistemic Logic, Temporal Logic and symbolic model checking in Chapter 1. It can serve as a new introduction text to both Dynamic Epistemic Logic and symbolic representation.

The next three chapters revolved around our first three research questions:

Can we find symbolic model checking methods for DEL?
How can symbolic model checking for DEL be implemented?
How good is the performance of symbolic methods for DEL?

We achieved our goal of putting a new engine into DEL with symbolic equivalents for all its components: Kripke models are encoded in knowledge and belief structures; action models are represented using transformers. Besides these modeling parts of the DEL framework, we also presented syntax, semantics and reduction axioms for a symbolic language with dynamic operators based on transformers.

The take-home message of Chapter 2 is "Everything is boolean!", because the key feature of our symbolic version of DEL is that all formulas have boolean equivalents with respect to a given structure. Besides being the crucial ingredient to the symbolic model checking procedure, this translation is also useful to study other properties of our structures. For example, we gave a symbolic analysis of bisimulations as boolean formulas in Section 2.11 and we compared different forms of redundancy and corresponding optimization methods in Section 2.12.

Our first research question can thus be answered with a clear yes.

In Chapter 3 we tackled the second question by moving from mathematics to programming, implementing our ideas in Haskell. Our main engineering challenge here was to implement the two tricks of symbolic representation: First, the set of possible worlds is replaced with the powerset of our vocabulary, restricted by a boolean formula. Second, explicit relations are replaced with subsets of the vocabulary, the so-called observational variables, or, if they are not equivalences, formulas over a double vocabulary. The management of double and quadruple vocabularies is mathematically simple, but not easy to keep track of manually. We therefore paid extra attention to lift this management of vocabularies to the type level, so that we no longer have to worry about it.

To make the implementation truly symbolic and efficient, instead of boolean formulas we used Binary Decision Diagrams wherever possible. While the worst-case complexity of BDDs is not better than that of other representations of boolean functions like truth tables or formulas in conjunctive normal form, they perform much better on real-world examples.

The main answer to our second question is thus given by BDDs, which just like for temporal logics, are an excellent data type for the boolean reasoning tasks underpinning our symbolic version of DEL. Additionally, we think that functional programming is the natural choice for implementing logics in general and model checking in particular.

The benchmark results and further examples in Chapter 4 give two answers to our third question: Compared with the explicit DEL model checkers DEMO and DEMO-S5, our new SMCDEL is clearly faster. Compared with temporal model checkers, the answer depends on concrete examples. If the scenario at hand is of the kind characterized in [Ben+09] and [DHR13], then modeling it in DEL instead of a temporal logic is usually simpler and more efficient. That is, if we have synchronicity, perfect-recall and no miracles, and individual time steps are not important, then DEL can compete with epistemic temporal logics. That we now have a symbolic model checker for DEL means we no longer have to choose between simplicity of logics and performance of implementations, but can have both at the same time.

For the next research question we analyzed a specific kind of knowledge, namely "knowing the value" or numeric knowledge.

How can we model knowledge of variables and values?

In Chapter 5 we first summarized two existing approaches, the binary encoding and register models. Our main contribution then is Public Inspection Logic (PIL), a basic dynamic epistemic logic of "knowing the value" and "inspecting the value". Analogous to the public announcement in PAL, the public inspection in PIL is a new dynamic operator that updates the agents' knowledge about values. We provided a sound and strongly complete axiomatization for the single and multi-agent case, making use of the well-known Armstrong axioms for dependencies in databases.

The second application of DEL we studied are dynamic gossip protocols. We began Chapter 6 with a new result that intuitively says that "anything might happen" in dynamic gossip, provided there are enough agents. The first research question then brought us to the study of gossip with the tools of epistemic logic.

Can we improve gossip protocols using epistemic logic?

Chapter 6 gives multiple positive answers to this. We introduced the new K_i^P operator for protocol-dependent knowledge and then used it to define four different ways to strengthen gossip protocols. Examples showed that these strengthenings indeed improve the previously studied "Learn New Secrets" protocol on many gossip graphs. However, we also proved a negative result: There is no perfect strengthening of "Learn New Secrets" that works on all graphs.

This leaves us with our last research question.

Can we use model checking for DEL to analyze gossip protocols?

We saw that model checking dynamic gossip explicitly is feasible for small numbers of agents and this is informative. In fact, our implementation of explicit state model checking was a perfect tool to find and check the results from the previous sections. Finally, combining topics from the beginning and the end of this thesis, we also showed how symbolic model checking methods can be used to model the gossip problem.

Overall, we see that the plurality of Dynamic Epistemic Logics is their biggest strength and weakness at the same time. Tailor-made languages like PIL models or gossip graphs are more intuitive to grasp and often they perform well enough on small examples, as shown in Section 6.6. An extreme case of this special-purpose modeling is the generalized quantifier approach to the Muddy Children puzzle from [GS11] which we discussed in Section 4.1.

Already a small deviation from the propositional standard DEL in syntax and semantics can make it hard or impossible to apply general purpose methods for symbolic model checking. To make this concrete, we can say that the DEMO-S5 "type variable trick", which we discussed in Section 3.1, trades efficiency for usability and generality. SMCDEL is our start to overcome this dichotomy between efficient methods that only work for the propositional standard framework on one side and convenient explicit methods that work with various kinds of models on the other side. We hope to extend SMCDEL with non-propositional variables in the future, similar to their usage in temporal logic model checkers.

Open Questions

There is obviously more to be explored, about DEL, knowing values and gossip in particular and the theory of symbolic representation for modal and dynamic logics in general. We now conclude with only a few points for future research, ranging from concrete questions to fluffy speculation.

As mentioned before, restricting postconditions to boolean formulas does not limit expressivity. The authors of [DK08] in fact prove the stronger result that postconditions can be restricted to $\top$ and $\bot$. Hence one can also model postconditions as functions of the type $A \to \mathcal{P}(V)$ as done in [Bol14]. What could be symbolic versions of such more compact postconditions and would they improve the usability and performance of our transformers?

In Section 2.13 we already mentioned the open question of how to extend our symbolic methods to non-normal logics for evidence and belief which are usually interpreted on neighborhood models.

Yet another variant of DEL introduces an "arbitrary announcement" operator which says that there is an announcement leading to a model where the given formula is true. Similarly, we can study "arbitrary arrow update" and "arbitrary action model" operators [Hal13]. All of these are very much dependent on the language. How can such "arbitrary" operators be interpreted on symbolic structures? What is the logic of "arbitrary transformation"?

Transformers also motivate a new notion of action equivalence. This might help to solve a problem with action models for which bisimulation had to be replaced with the more complicated notion of action emulation [ERS12]. Is there an easy way to determine whether two given transformers encode action models that emulate each other?

Perhaps the deepest issue that we see emerging in our approach is the following. Standard logical approaches to information flow assume a sharp distinction between syntax and semantic models. The BDD-oriented approach suggests the existence of a third intermediate level: We introduced state and observation laws as boolean formulas, but in practice we only need the boolean function they represent. Our structures are therefore not a compromise, but rather a both-and solution combining syntax and semantics. Also from the viewpoints of computational complexity and cognition, this might be the right level. We leave the exploration of this grander program to another occasion.

Bibliography

[AGH15] Krzysztof R. Apt, Davide Grossi, and Wiebe van der Hoek. "Epistemic Protocols for Distributed Gossiping". In: *Proceedings of the 15th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Ramanujam. TARK 2015. 2015. DOI: 10.4204/EPTCS.215.5 (cited on page 197).

[AGM85] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. "On the logic of theory change: Partial meet contraction and revision functions". In: *Journal of Symbolic Logic* 50.2 (1985), pages 510–530. DOI: 10.2307/2274239 (cited on page 57).

[All+17] Alice Allen et al. "Engineering Academic Software (Dagstuhl Perspectives Workshop 16252)". In: *Dagstuhl Manifestos* 6.1 (2017), pages 1–20. ISSN: 2193-2433. DOI: 10.4230/DagMan.6.1.1 (cited on page 4).

[AM18] Christopher Allen and Julie Moronuki. *Haskell Programming from first principles*. forthcoming. 2018. URL: http://haskellbook.com/ (cited on page 89).

[Arm74] William Ward Armstrong. "Dependency Structures of Data Base Relationships". In: *IFIP congress*. Volume 74. Geneva, Switzerland, 1974, pages 580–583. URL: https://is.gd/armstrong1974dependency (cited on pages 138, 139, 143, 145, 147).

[AS13] Guillaume Aucher and François Schwarzentruber. "On the Complexity of Dynamic Epistemic Logic". In: *Proceedings of the 14th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Burkhard C. Schipper. TARK 2013. 2013, pages 19–28. URL: https://is.gd/ComplexityDEL (cited on page 111).

[Att+14] Maduka Attamah, Hans van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. "Knowledge and Gossip". In: *Proceedings of the Twenty-first European Conference on Artificial Intelligence*. Frontiers in Artificial Intelligence and Applications. 2014, pages 21–26. ISBN: 978-1-61499-418-3. DOI: 10.3233/978-1-61499-419-0-21 (cited on pages 192, 193, 197).

[Att+15] Maduka Attamah, Hans van Ditmarsch, Davide Grossi, and Wiebe van der Hoek. "A Framework for Epistemic Gossip Protocols". In: *12th European Conference on Multi-Agent Systems EUMAS 2014*. Edited by Nils Bulling. 2015, pages 193–209. ISBN: 978-3-319-17130-2. DOI: 10.1007/978-3-319-17130-2_13 (cited on page 189).

[Att12] Maduka Attamah. "Visualising Dynamic Epistemic Logic". Master's thesis. University of Liverpool, 2012. URL: https://cgi.csc.liv.ac.uk/~byear/index_files/Maduka_MScDissertation.pdf (cited on page 87).

[Att15] Maduka Attamah. "Epistemic Gossip Protocols". PhD thesis. 2015. URL: https://livrepository.liverpool.ac.uk/3001317/ (cited on page 189).

[AW17] Krzysztof R. Apt and Dominik Wojtczak. "Common Knowledge in a Logic of Gossips". In: *Proceedings of the 16th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Jérôme Lang. TARK 2017. 2017. DOI: 10.4204/EPTCS.251.2 (cited on pages 164, 197).

[BA11] Thomas Bolander and Mikkel Birkegaard Andersen. "Epistemic planning for single- and multi-agent systems". In: *Journal of Applied Non-Classical Logics* 21.1 (2011), pages 9–34. DOI: 10.3166/jancl.21.9-34 (cited on page 128).

[Bai13] Justin Bailey. *The Haskell Cheatsheet*. July 22, 2013. URL: https://cheatsheet.codeslower.com/ (cited on page 89).

[Bai17] Leemon Baird. *The Swirlds Hashgraph Consensus Algorithm: Fair, Fast, Byzantine Fault Tolerance*. 2017. URL: https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf (cited on pages 3, 156).

[Bal16] Alexandru Baltag. "To Know is to Know the Value of a Variable". In: *Advances in Modal Logic*. Edited by Lev Beklemishev, Stéphane Demri, and András Máté. Volume 11. 2016, pages 135–155. URL: http://www.aiml.net/volumes/volume11/Baltag.pdf (cited on pages 3, 139, 153).

[Bar+17] Chitta Baral, Thomas Bolander, Hans van Ditmarsch, and Sheila McIlrath. “Epistemic Planning (Dagstuhl Seminar 17231)”. In: *Dagstuhl Reports* 7.6 (2017), pages 1–47. ISSN: 2192-5283. DOI: `10.4230/DagRep.7.6.1` (cited on page 130).

[BEK06] Johan van Benthem, Jan van Eijck, and Barteld Kooi. “Logics of communication and change”. In: *Information and computation* 204.11 (2006), pages 1620–1662. DOI: `10.1016/j.ic.2006.04.006` (cited on pages 19, 20, 63).

[Ben+09] Johan van Benthem, Jelle Gerbrandy, Tomohiro Hoshi, and Eric Pacuit. “Merging frameworks for interaction”. In: *Journal of Philosophical Logic* 38.5 (2009), pages 491–526. DOI: `10.1007/s10992-008-9099-x` (cited on pages 26, 27, 117, 200).

[Ben+15] Johan van Benthem, Jan van Eijck, Malvin Gattinger, and Kaile Su. “Symbolic Model Checking for Dynamic Epistemic Logic”. In: *Proceedings of the 5th International Workshop on Logic, Rationality, and Interaction (LORI 2015)*. Edited by Wiebe van der Hoek, Wesley H. Holliday, and Wen-fang Wang. 2015, pages 366–378. ISBN: 978-3-662-48561-3. DOI: `10.1007/978-3-662-48561-3_30` (cited on page 7).

[Ben+17] Johan van Benthem, Jan van Eijck, Malvin Gattinger, and Kaile Su. “Symbolic Model Checking for Dynamic Epistemic Logic – S5 and Beyond”. In: *Journal of Logic and Computation (JLC)* (2017). DOI: `10.1093/logcom/exx038`. URL: `https://is.gd/DELBDD` (cited on page 7).

[BFP14] Johan van Benthem, David Fernández-Duque, and Eric Pacuit. “Evidence and plausibility in neighborhood structures”. In: *Annals of Pure and Applied Logic* 165.1 (2014), pages 106–133. ISSN: 0168-0072. DOI: `10.1016/j.apal.2013.07.007` (cited on page 83).

[BFS17] Kai Brünnler, Dandolo Flumini, and Thomas Studer. “A Logic of Blockchain Updates”. In: *CoRR* abs/1707.01766 (2017). URL: `https://arxiv.org/abs/1707.01766` (cited on page 156).

[BLF85] Simon Baron-Cohen, Alan M. Leslie, and Uta Frith. “Does the autistic child have a “theory of mind”?” In: *Cognition* 21.1 (1985), pages 37–46. DOI: `10.1016/0010-0277(85)90022-8` (cited on page 126).

[BMS98] Alexandru Baltag, Lawrence S. Moss, and Slawomir Solecki. “The logic of public announcements, common knowledge, and private suspicions”. In: *Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Itzhak Gilboa. TARK 1998. 1998, pages 43–56. URL: `https://dl.acm.org/citation.cfm?id=645876.671885` (cited on page 20).

[Bol14] Thomas Bolander. "Seeing is Believing: Formalising False-Belief Tasks in Dynamic Epistemic Logic." In: *Proceedings of the European Conference on Social Intelligence (ECSI-2014)*. Edited by Andreas Herzig and Emiliano Lorini. 2014, pages 246–263. URL: `http://ceur-ws.org/Vol-1283/paper_14.pdf` (cited on pages 127, 202).

[BRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science 53. Cambridge University Press, 2001. ISBN: 978-0-521-52714-9 (cited on pages 13, 17, 48, 78).

[Bry86] Randal E. Bryant. "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transaction on Computers* C-35.8 (1986), pages 677–691. DOI: `10.1109/TC.1986.1676819` (cited on pages 28, 34, 36, 51, 125).

[BS02] Pierpaolo Battigalli and Marciano Siniscalchi. "Strong Belief and Forward Induction Reasoning". In: *Journal of Economic Theory* 106.2 (2002), pages 356–391. DOI: `10.1006/jeth.2001.2942` (cited on page 197).

[BS08] Alexandru Baltag and Sonja Smets. "A qualitative theory of dynamic interactive belief revision". In: *Logic and the Foundations of Game and Decision Theory (LOFT 7)*. Volume 3. Texts in logic and games. Amsterdam University Press, 2008, pages 9–58. ISBN: 9789089640260. URL: `https://hdl.handle.net/11245/1.300554` (cited on page 57).

[BS15] Johan van Benthem and Sonja Smets. "Dynamic Logics of Belief Change". In: *Handbook of Epistemic Logic*. Edited by Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek, and Barteld Kooi. College Publications, 2015, pages 313–393. ISBN: 978-1-84890-158-2. URL: `https://is.gd/BenSme2015DLBC` (cited on page 27).

[BS72] Brenda Baker and Robert Shostak. "Gossips and telephones". In: *Discrete Mathematics* 2.3 (1972), pages 191–193. DOI: `10.1016/0012-365X(72)90001-5` (cited on page 155).

[BSZ09] Alexandru Baltag, Sonja Smets, and Jonathan Alexander Zvesper. "Keep 'hoping' for rationality: a solution to the backward induction paradox". In: *Synthese* 169.2 (2009), pages 301–333. DOI: `10.1007/s11229-009-9559-z` (cited on pages 173, 198).

[Bur+90] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and Lain-Jinn Hwang. "Symbolic model checking: 10^{20} states and beyond". In: *Fifth Annual IEEE Symposium on Logic in Computer Science*. 1990, pages 428–439. DOI: `10.1109/lics.1990.113767` (cited on pages 2, 28, 31).

[CF10] Henry Corrigan-Gibbs and Bryan Ford. “Dissent: Accountable Anonymous Group Messaging”. In: *Proceedings of the 17th ACM Conference on Computer and Communications Security*. CCS 2010. 2010, pages 340–350. ISBN: 978-1-4503-0245-6. DOI: `10.1145/1866307.1866346` (cited on page 130).

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E Long. “Model checking and abstraction”. In: *ACM transactions on Programming Languages and Systems* 16.5 (1994), pages 1512–1542. DOI: `10.1145/186025.186051` (cited on pages 28, 32).

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. Cambridge, Massachusetts, USA: The MIT Press, 1999. ISBN: 9780262032704 (cited on pages 25, 28, 32, 86, 134, 136).

[CH00] Koen Claessen and John Hughes. “QuickCheck: a lightweight tool for random testing of Haskell programs”. In: *Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*. Edited by Martin Odersky and Philip Wadler. 2000, pages 268–279. DOI: `10.1145/351240.351266` (cited on page 105).

[Cha85] David Chaum. “Security without identification: transaction systems to make big brother obsolete”. In: *Communications of the ACM* 28.10 (1985), pages 1030–1044. DOI: `10.1145/4372.4373` (cited on page 117).

[Cha88] David Chaum. “The dining cryptographers problem: Unconditional sender and recipient untraceability”. In: *Journal of Cryptology* 1.1 (1988), pages 65–75. ISSN: 0933-2790. DOI: `10.1007/BF00206326` (cited on pages 112, 116).

[Cia16] Ivano Ciardelli. “Questions in Logic”. PhD thesis. University of Amsterdam, 2016. URL: `https://www.illc.uva.nl/Research/Publications/Dissertations/DS-2016-01.text.pdf` (cited on page 153).

[Cim+02] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. “NuSMV 2: An OpenSource Tool for Symbolic Model Checking”. In: *Computer Aided Verification (CAV)*. Edited by Ed Brinksma and Kim Guldstrand Larsen. 2002, pages 359–364. DOI: `10.1007/3-540-45657-0_29` (cited on page 86).

[CKS09] Jacques Carette, Oleg Kiselyov, and Chung-chieh Shan. “Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages”. In: *Journal of Functional Programming* 19.5 (2009), pages 509–543. DOI: `10.1017/S0956796809007205` (cited on page 107).

[Cla+01] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. "Bounded Model Checking Using Satisfiability Solving". In: *Formal Methods in System Design* 19.1 (July 1, 2001), pages 7–34. ISSN: 1572-8102. DOI: 10.1023/A:1011276507260 (cited on page 107).

[Cla+18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Forthcoming. Springer International Publishing, Mar. 20, 2018. ISBN: 978-3-319-10574-1. DOI: 10.1007/978-3-319-10575-8 (cited on page 28).

[Com18] Commercial Haskell Special Interest Group. *The Haskell Tool Stack*. Version 1.6.5. Feb. 19, 2018. URL: https://haskellstack.org (cited on page 109).

[Cor+15] Andrés Cordón-Franco, Hans van Ditmarsch, David Fernández-Duque, and Fernando Soler-Toscano. "A geometric protocol for cryptography with cards". In: *Designs, Codes and Cryptography* 74.1 (2015), pages 113–125. ISSN: 0925-1022. DOI: 10.1007/s10623-013-9855-y (cited on page 120).

[CR15] Ivano Ciardelli and Floris Roelofsen. "Inquisitive dynamic epistemic logic". In: *Synthese* 192.6 (2015), pages 1643–1687. DOI: 10.1007/s11229-014-0404-7 (cited on page 153).

[CS15] Tristan Charrier and François Schwarzentruber. "Arbitrary Public Announcement Logic with Mental Programs". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. IFAAMAS. 2015, pages 1471–1479. URL: https://dl.acm.org/citation.cfm?id=2772879.2773340 (cited on page 107).

[CS17] Tristan Charrier and François Schwarzentruber. "A Succinct Language for Dynamic Epistemic Logic". In: *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems*. AAMAS '17. São Paulo, Brazil: International Foundation for Autonomous Agents and Multiagent Systems, 2017, pages 123–131. URL: http://www.aamas2017.org/proceedings/pdfs/p123.pdf (cited on pages 38, 107).

[CW96] Edmund M. Clarke and Jeannette M. Wing. "Formal Methods: State of the Art and Future Directions". In: *ACM Computing Surveys* 28.4 (1996), pages 626–643. ISSN: 0360-0300. DOI: 10.1145/242223.242257 (cited on page 2).

[CWX11] Vincent Conitzer, Toby Walsh, and Lirong Xia. "Dominating Manipulations in Voting with Partial Information". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011, pages 638–643. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3706 (cited on page 198).

[DE12] Kees Doets and Jan van Eijck. *The Haskell Road to Logic, Maths and Programming*. 2012. ISBN: 0-9543006-9-6 (cited on page 89).

[DEW10] Hans van Ditmarsch, Jan van Eijck, and William Wu. "One Hundred Prisoners and a Lightbulb — Logic and Computation". In: *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning*. 2010, pages 90–100. ISBN: 978-1-5773-545-12. URL: `https://www.aaai.org/ocs/index.php/KR/KR2010/paper/view/1234` (cited on page 130).

[DHK07] Hans van Ditmarsch, Wiebe van der Hoek, and Barteld Kooi. *Dynamic Epistemic Logic*. Springer, 2007. ISBN: 978-1-4020-5838-7. DOI: `10.1007/978-1-4020-5839-4` (cited on pages 14, 19, 23, 37, 45, 122).

[DHR13] Hans van Ditmarsch, Wiebe van der Hoek, and Ji Ruan. "Connecting dynamic epistemic and temporal epistemic logics". In: *Logic Journal of IGPL* 21.3 (2013), pages 380–403. DOI: `10.1093/jigpal/jzr038` (cited on pages 27, 117, 118, 200).

[Din16] Yifeng Ding. "Epistemic Logic with Functional Dependency Operator". In: *Studies in Logic* 9.4 (2016), pages 55–84. URL: `https://arxiv.org/abs/1706.02048` (cited on page 154).

[Dit+06] Hans van Ditmarsch, Wiebe van der Hoek, Ron van der Meyden, and Ji Ruan. "Model Checking Russian Cards". In: *Electronic Notes in Theoretical Computer Science* 149.2 (2006), pages 105–123. DOI: `10.1016/j.entcs.2005.07.029` (cited on pages 27, 38, 87, 120, 121).

[Dit+12] Hans van Ditmarsch, Jan van Eijck, Ignacio Hernández-Antón, Floor Sietsma, Sunil Simon, and Fernando Soler-Toscano. "Modelling Cryptographic Keys in Dynamic Epistemic Logic with DEMO". In: *Highlights on Practical Applications of Agents and Multi-Agent Systems*. Edited by Javier Bajo Pérez et al. 2012, pages 155–162. ISBN: 978-3-642-28762-6 (cited on page 87).

[Dit+13] Hans van Ditmarsch, Andreas Herzig, Emiliano Lorini, and François Schwarzentruber. "Listen to Me! Public Announcements to Agents That Pay Attention — or Not". In: *Logic, Rationality, and Interaction*. Edited by Davide Grossi, Olivier Roy, and Huaxin Huang. 2013, pages 96–109. ISBN: 978-3-642-40948-6 (cited on page 137).

[Dit+14] Hans van Ditmarsch, Sujata Ghosh, Rineke Verbrugge, and Yanjing Wang. "Hidden protocols: Modifying our expectations in an evolving world". In: *Artificial Intelligence* 208 (2014), pages 18–40. ISSN: 0004-3702. DOI: `10.1016/j.artint.2013.12.001` (cited on page 197).

[Dit+15] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. "Dynamic Gossip". In: *Preprint* (2015). URL: https://arxiv.org/abs/1511.00867 (cited on pages 155, 157, 159, 164, 185, 197).

[Dit+16] Hans van Ditmarsch, Davide Grossi, Andreas Herzig, Wiebe van der Hoek, and Louwe B. Kuijer. "Parameters for Epistemic Gossip Problems". In: *Proceedings of LOFT 2016*. 2016. URL: https://sites.google.com/site/lbkuijer/LOFT_Gossip_Revised.pdf (cited on page 198).

[Dit+17] Hans van Ditmarsch, Jan van Eijck, Pere Pardo, Rahim Ramezanian, and François Schwarzentruber. "Epistemic protocols for dynamic gossip". In: *Journal of Applied Logic* 20 (2017), pages 1–31. DOI: 10.1016/j.jal.2016.12.001 (cited on pages 155, 157, 197).

[Dit+18] Hans van Ditmarsch, Malvin Gattinger, Louwe B. Kuijer, and Pere Pardo. *How Come You Don't Call Me? Common Knowledge of Gossip Protocols*. Submitted. 2018 (cited on page 8).

[Dit03] Hans van Ditmarsch. "The Russian Cards problem". In: *Studia Logica* 75.1 (2003), pages 31–62. DOI: 10.1023/A:1026168632319 (cited on pages 120, 121).

[DK08] Hans van Ditmarsch and Barteld Kooi. "Semantic Results for Ontic and Epistemic Change". In: *Logic and the Foundations of Game and Decision Theory (LOFT 7)*. Volume 3. Texts in logic and games. Amsterdam University Press, 2008, pages 87–117. ISBN: 9789089640260. URL: https://arxiv.org/abs/cs/0610093 (cited on pages 20, 23, 202).

[DKS17] Hans van Ditmarsch, Ioannis Kokkinis, and Anders Stockmarr. "Reachability and Expectation in Gossiping". In: *Proceedings of PRIMA 2017*. 2017. URL: https://sites.google.com/site/ykokkinis/prima17.pdf (cited on page 157).

[DM17] Chris Dornan and Simon Marlow. *Alex: A lexical analyser generator for Haskell*. Version 3.2.3. Sept. 8, 2017. URL: https://www.haskell.org/alex (cited on pages 95, 104).

[DRV08] Hans P. van Ditmarsch, Ji Ruan, and Rineke Verbrugge. "Sum and Product in Dynamic Epistemic Logic". In: *Journal of Logic and Computation* 18.4 (2008), pages 563–588. DOI: 10.1093/logcom/exm081 (cited on pages 122, 124).

[EG15] Jan van Eijck and Malvin Gattinger. "Elements of Epistemic Crypto Logic". In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. AAMAS '15. Istanbul, Turkey: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pages 1795–1796. ISBN: 978-1-4503-3413-6. URL: `https://dl.acm.org/citation.cfm?id=2773441` (cited on pages 8, 134, 135, 137).

[EGW17] Jan van Eijck, Malvin Gattinger, and Yanjing Wang. "Knowing Values and Public Inspection". In: *Seventh Indian Conference on Logic and Its Applications: ICLA 2017, Kanpur, India*. Edited by Sujata Ghosh and Sanjiva Prasad. 2017, pages 77–90. ISBN: 978-3-662-54069-5. DOI: `10.1007/978-3-662-54069-5_7`. URL: `https://arxiv.org/abs/1609.03338` (cited on pages 8, 151).

[Eij07] Jan van Eijck. "DEMO—a demo of epistemic modelling". In: *Interactive Logic. Selected Papers from the 7th Augustus de Morgan Workshop, London*. Volume 1. 2007, pages 303–362. URL: `https://homepages.cwi.nl/~jve/papers/07/pdfs/DEMO_IL.pdf` (cited on pages 37, 87).

[Eij11] Jan van Eijck. *Demo Light*. Technical report. CWI, 2011. URL: `https://homepages.cwi.nl/~jve/software/demolight0/` (cited on page 87).

[Eij13] Jan van Eijck. *PRODEMO: Implementation of Model Checking for EPL*. Technical report. CWI, 2013. URL: `https://homepages.cwi.nl/~jve/software/prodemo/PRODEMO.pdf` (cited on page 87).

[Eij14a] Jan van Eijck. *DEMO-S5*. Technical report. CWI, 2014. URL: `https://homepages.cwi.nl/~jve/software/demo_s5` (cited on pages 31, 37, 87, 104, 109, 122).

[Eij14b] Jan van Eijck. "Dynamic epistemic logics". In: *Johan van Benthem on Logic and Information Dynamics*. Edited by Alexandru Baltag and Sonja Smets. 2014, pages 175–202. ISBN: 978-3-319-06025-5. DOI: `10.1007/978-3-319-06025-5_7` (cited on page 63).

[Eij14c] Jan van Eijck. *Relations, Equivalences, Partitions*. Technical report. CWI, 2014. URL: `https://homepages.cwi.nl/~jve/software/demo_s5/EREL.pdf` (cited on page 30).

[Ell+04] John Ellson, Emden R Gansner, Eleftherios Koutsofios, Stephen C North, and Gordon Woodhull. "Graphviz and dynagraph — static and dynamic graph drawing tools". In: *Graph drawing software* (2004), pages 127–148 (cited on pages 87, 104).

[Eme08] E. Allen Emerson. "The Beginning of Model Checking: A Personal Perspective". In: *25 Years of Model Checking: History, Achievements, Perspectives.* Edited by Orna Grumberg and Helmut Veith. 2008, pages 27–45. ISBN: 978-3-540-69850-0. DOI: `10.1007/978-3-540-69850-0_2` (cited on page 1).

[Eng+15] Thorsten Engesser, Thomas Bolander, Robert Mattmüller, and Bernhard Nebel. "Cooperative Epistemic Multi-Agent Planning With Implicit Coordination". In: *ICAPS Proceedings of the 3rd Workshop on Distributed and Multi-Agent Planning (DMAP-2015)* (2015), pages 68–76. URL: `https://is.gd/DMAP2015_p68` (cited on pages 121, 128).

[EO07] Jan van Eijck and Simona Orzan. "Epistemic Verification of Anonymity". In: *Electronic Notes in Theoretical Computer Science* 168 (2007). Second International Workshop on Views on Designing Complex Architectures (VODCA 2006), pages 159–174. ISSN: 1571-0661. DOI: `10.1016/j.entcs.2006.08.026` (cited on page 116).

[ERS12] Jan van Eijck, Ji Ruan, and Tomasz Sadzik. "Action emulation". In: *Synthese* 185.1 (2012), pages 131–151. DOI: `10.1007/s11229-012-0083-1` (cited on pages 55, 202).

[Eug+04] Patrick Th. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. "Epidemic Information Dissemination in Distributed Systems". In: *IEEE Computer* 37.5 (2004), pages 60–67. DOI: `10.1109/MC.2004.1297243` (cited on page 155).

[Fag+95] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about knowledge.* Volume 4. MIT press Cambridge, 1995. ISBN: 9780262061629 (cited on pages 13, 14, 45).

[Fan17] Jeremy Fantl. "Knowledge How". In: *The Stanford Encyclopedia of Philosophy.* Edited by Edward N. Zalta. Fall 2017. Metaphysics Research Lab, Stanford University, 2017. URL: `https://plato.stanford.edu/archives/fall2017/entries/knowledge-how/` (cited on page 131).

[FG16] David Fernández-Duque and Valentin Goranko. "Secure aggregation of distributed information: how a team of agents can safely share secrets in front of a spy". In: *Discrete Applied Mathematics* 198 (2016), pages 118–135. DOI: `10.1016/j.dam.2015.06.022` (cited on pages 120, 130).

[Fit03] Melvin Fitting. "Bisimulations and Boolean Vectors". In: *Advances in Modal Logic.* Volume 4. King's College Publications, 2003, pages 97–126. URL: `http://www.aiml.net/volumes/volume4/Fitting.ps` (cited on page 62).

[Fre69] Hans Freudenthal. "Formulering van het 'som-en-product'-probleem". In: *Nieuw Archief voor Wiskunde* 17 (1969), page 152 (cited on page 122).

[Gat14] Malvin Gattinger. "Dynamic Epistemic Logic for Guessing Games and Cryptographic Protocols". See also `https://w4eg.de/malvin/illc/thesis/`. Master's thesis. ILLC, University of Amsterdam, 2014. URL: `https://eprints.illc.uva.nl/934/` (cited on pages 87, 134, 135, 136, 137).

[Gat16] Malvin Gattinger. "A Model Checker for the Hardest Logic Puzzle Ever". In: *PhDs in Logic VIII, Darmstadt*. 2016 (cited on pages 8, 129).

[Gat17a] Malvin Gattinger. *HasCacBDD*. Version 0.1.0.0. Mar. 9, 2017. URL: `https://github.com/m4lvin/HasCacBDD` (cited on page 92).

[Gat17b] Malvin Gattinger. "Towards Symbolic Factual Change in DEL". In: *Proceedings of the ESSLLI 2017 Student Session*. Edited by Karoliina Lohiniva and Johannes Wahle. 2017, pages 14–24. URL: `https://is.gd/symbolicfactualchange` (cited on page 7).

[Gat18] Malvin Gattinger. *SMCDEL — An Implementation of Symbolic Model Checking for Dynamic Epistemic Logic with Binary Decision Diagrams*. Version 1.0.0. Feb. 26, 2018. DOI: `10.5281/zenodo.1184686`. URL: `https://github.com/jrclogic/SMCDEL` (cited on pages 7, 92, 110, 112).

[GJ04] Philippe Golle and Ari Juels. "Dining Cryptographers Revisited". In: *Advances in Cryptology - EUROCRYPT 2004*. Edited by Christian Cachin and Jan L. Camenisch. 2004, pages 456–473. ISBN: 978-3-540-24676-3. DOI: `10.1007/978-3-540-24676-3_27` (cited on page 130).

[GM04] Peter Gammie and Ron van der Meyden. "MCK: Model Checking the Logic of Knowledge". In: *Computer Aided Verification*. 2004, pages 479–483. DOI: `10.1007/978-3-540-27813-9_41` (cited on page 86).

[GM17] Andy Gill and Simon Marlow. *Happy: the parser generator for Haskell*. Version 1.19.8. Oct. 12, 2017. URL: `https://www.haskell.org/happy` (cited on pages 95, 104).

[GR02] Nikos Gorogiannis and Mark D. Ryan. "Implementation of Belief Change Operators Using BDDs". In: *Studia Logica* 70.1 (2002), pages 131–156. ISSN: 0039-3215. DOI: `10.1023/A:1014610426691` (cited on pages 32, 33, 82).

[GS11] Nina Gierasimczuk and Jakub Szymanik. "A note on a generalization of the Muddy Children puzzle". In: *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Krzysztof R. Apt. TARK 2011. 2011, pages 257–264. ISBN: 978-1-4503-0707-9. DOI: 10.1145/2000378.2000409 (cited on pages 104, 110, 111, 201).

[GT17] Tetsuji Goto and Satoshi Tojo. *DEMO+A*. Version 0.0.1.0. Sept. 5, 2017. URL: http://cirrus.jaist.ac.jp:8080/soft/demo_plus_a/ (cited on page 87).

[GV17] Jan Friso Groote and Erik P. de Vink. "Problem Solving Using Process Algebra Considered Insightful". In: *ModelEd, TestEd, TrustEd: Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*. Edited by Joost-Pieter Katoen, Rom Langerak, and Arend Rensink. 2017, pages 48–63. ISBN: 978-3-319-68270-9. DOI: 10.1007/978-3-319-68270-9_3 (cited on page 130).

[GW16] Tau Gu and Yanjing Wang. ""Knowing value" logic as a normal modal logic". In: edited by Lev Beklemishev, Stéphane Demri, and András Máté. Volume 11. 2016, pages 362–381. URL: http://www.aiml.net/volumes/volume11/Gu-Wang.pdf (cited on pages 3, 138, 139).

[Hae+16] Bernhard Haeupler, Gopal Pandurangan, David Peleg, Rajmohan Rajaraman, and Zhifeng Sun. "Discovery Through Gossip". In: *Random Structures & Algorithms* 48.3 (2016), pages 565–587. ISSN: 1098-2418. DOI: 10.1002/rsa.20621 (cited on page 155).

[Hae15] Bernhard Haeupler. "Simple, Fast and Deterministic Gossip and Rumor Spreading". In: *Journal of the ACM* 62.6 (2015). DOI: 10.1145/2767126 (cited on page 197).

[Hal13] James Hales. "Arbitrary Action Model Logic and Action Model Synthesis". In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 2013. DOI: 10.1109/lics.2013.31 (cited on pages 24, 202).

[Hen17] Simon Hengel. *Hspec: A Testing Framework for Haskell*. Version 2.4.4. June 16, 2017. URL: https://hspec.github.io/ (cited on page 106).

[HHL88] Sandra M Hedetniemi, Stephen T Hedetniemi, and Arthur L Liestman. "A survey of gossiping and broadcasting in communication networks". In: *Networks* 18.4 (1988), pages 319–349. DOI: 10.1002/net.3230180406 (cited on pages 155, 156).

[Hil13] David Hilbert. *David Hilbert's Lectures on the Foundations of Arithmetic and Logic 1917-1933*. Edited by Michael Hallett, Ulrich Majer, and Dirk Schlimm. Volume 3. 2013. ISBN: 978-3-540-20578-4. DOI: 10.1007/978-3-540-69444-1 (cited on page 97).

[HLL99] Mor Harchol-Balter, Frank Thomson Leighton, and Daniel Lewin. "Resource Discovery in Distributed Networks". In: *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing (PODC)*. 1999, pages 229–237. DOI: `10.1145/301308.301362` (cited on pages 156, 197).

[HLM15] Andreas Herzig, Emiliano Lorini, and Faustine Maffre. "A Poor Man's Epistemic Logic Based on Propositional Assignment and Higher-Order Observation". In: *LORI 2015: Logic, Rationality, and Interaction (LORI) 5th International Workshop, Taipei, Taiwan, October 28-30*. Edited by Wiebe van der Hoek, Wesley H. Holliday, and Wen-fang Wang. 2015, pages 156–168. DOI: `10.1007/978-3-662-48561-3_13` (cited on page 38).

[HM17] Andreas Herzig and Faustine Maffre. "How to share knowledge by gossiping". In: *AI Communications* 30.1 (2017), pages 1–17. DOI: `10.3233/aic-170723` (cited on pages 38, 157).

[HN16] Kristine Harjes and Pavel Naumov. "Functional Dependence in Strategic Games". In: *Notre Dame Journal Formal Logic* 57.3 (2016), pages 341–353. DOI: `10.1215/00294527-3479096` (cited on page 154).

[Hos09] Tomohiro Hoshi. "Epistemic Dynamics and Protocol Information". PhD thesis. Amsterdam University, 2009. URL: `https://www.illc.uva.nl/cms/Research/Publications/Dissertations/DS-2009-08.text.pdf` (cited on page 197).

[HP17] Joseph Y Halpern and Rafael Pass. "A Knowledge-Based Analysis of the Blockchain". In: *Proceedings of the 16th Conference on Theoretical Aspects of Rationality and Knowledge*. Edited by Jérôme Lang. TARK 2017. 2017. DOI: `10.4204/EPTCS.251.22` (cited on page 156).

[HS15] Vincent Hendricks and John Symons. "Epistemic Logic". In: *The Stanford Encyclopedia of Philosophy*. Edited by Edward N. Zalta. Fall 2015. Metaphysics Research Lab, Stanford University, 2015. URL: `https://plato.stanford.edu/archives/fall2015/entries/logic-epistemic/` (cited on page 15).

[HST15] Ryo Hatano, Katsuhiko Sano, and Satoshi Tojo. "Linear Algebraic Semantics for Multi-agent Communication". In: *Proceedings of the International Conference on Agents and Artificial Intelligence*. SCITEPRESS Science and Technology Publications, 2015. DOI: `10.5220/0005219001740181` (cited on page 62).

[Hur00] C. A. J. Hurkens. "Spreading gossip efficiently". In: *Nieuw Archief voor Wiskunde* 5.1 (2000), pages 208–210. URL: `http://www.nieuwarchief.nl/serie5/pdf/naw5-2000-01-2-208.pdf` (cited on page 156).

[HV91] Joseph Y. Halpern and Moshe Y. Vardi. "Model Checking vs. Theorem Proving: A Manifesto". In: *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*. Edited by Vladimir Lifschitz. 1991, pages 151–176. DOI: `10.1016/b978-0-12-450010-5.50015-3` (cited on page 29).

[Irv16] Mix Irving. *Gossiping Securely is the new Email.* Feb. 13, 2016. URL: `https://is.gd/IrvingGossipingSecurely` (cited on pages 3, 155).

[Kar+00] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. "Randomized Rumor Spreading". In: *41st Annual Symposium on Foundations of Computer Science, FOCS*. 2000, pages 565–574. DOI: `10.1109/SFCS.2000.892324` (cited on pages 156, 197).

[Kle17] Rana Klein. "The Logical Dynamics of Gossip: an analysis in Dynamic Epistemic Logic". Master's thesis. ILLC, University of Amsterdam, 2017. URL: `https://eprints.illc.uva.nl/1567/` (cited on page 155).

[Kme16] Edward A. Kmett. *tagged: Haskell 98 phantom types to avoid unsafely passing dummy arguments.* Version 0.8.5. July 23, 2016. URL: `https://hackage.haskell.org/package/tagged-0.8.5` (cited on page 99).

[Knu08] Donald E. Knuth. *Fun With Binary Decision Diagrams (BDDs)*. June 2008. URL: `https://youtu.be/SQE21efsf7Y` (cited on pages 34, 36).

[Knu11] Donald E. Knuth. *The Art of Computer Programming. Combinatorial Algorithms, Part 1.* Volume 4A. Addison-Wesley Professional, 2011. URL: `https://cs.stanford.edu/~uno/taocp.html` (cited on page 36).

[Knu84] Donald E. Knuth. "Literate Programming". In: *The Computer Journal* 27.2 (1984), pages 97–111. DOI: `10.1093/comjnl/27.2.97` (cited on page 85).

[Kov11] Daniel L. Kovacs. *BNF definition of PDDL 3.1.* 2011. URL: `https://is.gd/PDDL2011` (cited on page 128).

[KR11a] Barteld P. Kooi and Bryan Renne. "Generalized arrow update logic". In: *Proceedings of the 13th Conference on Theoretical Aspects of Rationality and Knowledge.* Edited by Krzysztof R. Apt. TARK 2011. 2011, pages 205–211. DOI: `10.1145/2000378.2000403` (cited on page 25).

[KR11b] Barteld Kooi and Bryan Renne. "Arrow Update Logic". In: *The Review of Symbolic Logic* 4.4 (2011), pages 536–559. DOI: `10.1017/S1755020311000189` (cited on pages 24, 71, 73).

[Kur+95] Ágnes Kurucz, István Németi, Ildikó Sain, and András Simon. "Decidable and undecidable logics with a binary modality". In: *Journal of Logic, Language and Information* 4.3 (Sept. 1995), pages 191–206. ISSN: 1572-9583. DOI: 10.1007/BF01049412 (cited on page 82).

[LF17] Esteban Landerreche and David Fernández-Duque. "A case study in almost-perfect security for unconditionally secure communication". In: *Designs, Codes and Cryptography* 83.1 (2017), pages 145–168. ISSN: 1573-7586. DOI: 10.1007/s10623-016-0210-y (cited on page 120).

[Lip11] Miran Lipovača. *Learn You a Haskell for Great Good!* 2011. ISBN: 978-1-59327-283-8. URL: http://learnyouahaskell.com/ (cited on page 89).

[Lit53] J.E. Littlewood. *A Mathematician's Miscellany*. London: Methuen, 1953. URL: https://archive.org/details/mathematiciansmi033496mbp (cited on page 44).

[LMR00] Alessio R. Lomuscio, Ron van der Meyden, and Mark Ryan. "Knowledge in Multiagent Systems: Initial Configurations and Broadcast". In: *Transactions on Computational Logic (TOCL)* 1.2 (2000), pages 247–284. ISSN: 1529-3785. DOI: 10.1145/359496.359527 (cited on page 38).

[LP15] Alessio Lomuscio and Wojciech Penczek. "Model Checking Temporal Epistemic Logic". In: *Handbook of Epistemic Logic*. Edited by Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek, and Barteld Kooi. College Publications, 2015, pages 397–441. ISBN: 978-1-84890-158-2. URL: https://is.gd/LomPen2015MCTEL (cited on page 25).

[LQR15] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. "MCMAS: an open-source model checker for the verification of multi-agent systems". In: *International Journal on Software Tools for Technology Transfer* (2015), pages 1–22. ISSN: 1433-2779. DOI: 10.1007/s10009-015-0378-x (cited on pages 31, 37, 86, 112, 117, 118, 119).

[LR06] Alessio Lomuscio and Franco Raimondi. "MCMAS: A model checker for multi-agent systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS 2006. 2006, pages 450–454. DOI: 10.1007/11691372_31 (cited on page 86).

[LSX13] Guanfeng Lv, Kaile Su, and Yanyan Xu. "CacBDD: A BDD Package with Dynamic Cache Management". In: *Proceedings of the 25th International Conference on Computer Aided Verification*. CAV'13. 2013, pages 229–234. ISBN: 978-3-642-39798-1. DOI: 10.1007/978-3-642-39799-8_15 (cited on pages 92, 110).

[LT93] Nancy G. Leveson and Clark S. Turner. "An investigation of the Therac-25 accidents". In: *Computer* 26.7 (1993), pages 18–41. DOI: 10.1109/MC.1993.274940 (cited on page 1).

[Luo+08] Xiangyu Luo, Kaile Su, Abdul Sattar, and Yan Chen. "Solving Sum and Product Riddle via BDD-Based Model Checking". In: *2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE, 2008, pages 630–633. DOI: 10.1109/WIIAT.2008.277 (cited on pages 31, 38, 123).

[Lut06] Carsten Lutz. "Complexity and Succinctness of Public Announcement Logic". In: *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*. AAMAS '06. Hakodate, Japan: ACM, 2006, pages 137–143. ISBN: 1-59593-303-4. DOI: 10.1145/1160633.1160657 (cited on pages 19, 43).

[LW13] Fenrong Liu and Yanjing Wang. "Reasoning about agent types and the hardest logic puzzle ever". In: *Minds and Machines* 23.1 (2013), pages 123–161. DOI: 10.1007/s11023-012-9287-x (cited on page 129).

[McD+98] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. *PDDL – The Planning Domain Definition Language – Version 1.2*. Technical report. CVC TR-98-003. Yale Center for Computational Vision and Control, 1998. URL: https://is.gd/PDDL1988 (cited on page 128).

[McM93] Kenneth L. McMillan. "Symbolic Model Checking: an Approach to the State Explosion Problem". PhD thesis. Carnegie Mellon University, 1993 (cited on page 86).

[Mei15] Reshef Meir. "Plurality Voting Under Uncertainty". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI 2015. 2015, pages 2103–2109. ISBN: 0-262-51129-0. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9784 (cited on page 198).

[MN10] Sara Miner More and Pavel Naumov. "An Independence Relation for Sets of Secrets". In: *Studia Logica* 94.1 (2010), pages 73–85. DOI: 10.1007/s11225-010-9223-0 (cited on page 154).

[MS04] Ron van der Meyden and Kaile Su. "Symbolic Model Checking the Knowledge of the Dining Cryptographers". In: *17th IEEE Computer Security Foundations Workshop*. 2004, pages 280–291. ISBN: 0-7695-2169-X. DOI: 10.1109/CSFW.2004.1310747 (cited on pages 31, 38).

[Nau12] Pavel Naumov. “Independence in Information Spaces”. In: *Studia Logica* 100.5 (2012), pages 953–973. DOI: `10.1007/s11225-012-9435-6` (cited on page 154).

[NN14] Pavel Naumov and Brittany Nicholls. “Rationally Functional Dependence”. In: *Journal of Philosophical Logic* 43.2-3 (2014), pages 603–616. DOI: `10.1007/s10992-013-9283-5` (cited on page 154).

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. Electronic edition, freely accessible at `https://books.osborne.economics.utoronto.ca/`. The MIT Press, 1994. ISBN: 0-262-65040-1 (cited on page 173).

[OSG08] Bryan O’Sullivan, Don Stewart, and John Goerzen. *Real World Haskell*. O’Reilly Media, 2008. ISBN: 978-0596514983. URL: `http://book.realworldhaskell.org/` (cited on page 89).

[OSu16] Bryan O’Sullivan. *Criterion*. 2016. URL: `http://www.serpentine.com/criterion` (cited on pages 110, 125).

[Per14] Andrés Perea. “Belief in the opponents’ future rationality”. In: *Games and Economic Behavior* 83.Supplement C (2014), pages 231–254. ISSN: 0899-8256. DOI: `10.1016/j.geb.2013.11.008` (cited on pages 173, 197).

[Pla07] Jan Plaza. “Logics of public communications”. In: *Synthese* 158.2 (2007). Originally published in 1989., pages 165–179. ISSN: 1573-0964. DOI: `10.1007/s11229-007-9168-7` (cited on pages 17, 138, 139, 153).

[Pol15] Iris van de Pol. “How Difficult is it to Think that you Think that I Think that...? A DEL-based Computational-level Model of Theory of Mind and its Complexity”. Master’s thesis. ILLC, University of Amsterdam, 2015. URL: `https://eprints.illc.uva.nl/955/` (cited on page 126).

[PRS15] Iris van de Pol, Iris van Rooij, and Jakub Szymanik. “Parameterized Complexity Results for a Model of Theory of Mind Based on Dynamic Epistemic Logic”. In: *Proceedings of the 15th Conference on Theoretical Aspects of Rationality and Knowledge*. TARK 2015. 2015, pages 246–263. DOI: `10.4204/EPTCS.215.18` (cited on page 126).

[PTW13] Rohit Parikh, Çağil Taşdemir, and Andreas Witzel. “The Power of Knowledge in Games”. In: *International Game Theory Review* 15.4 (2013). DOI: `10.1142/S0219198913400306` (cited on page 198).

[SLZ04] Kaile Su, Guanfeng Lv, and Yan Zhang. "Reasoning about Knowledge by Variable Forgetting". In: (2004). Edited by Didier Dubois, Christopher Welty, and Mary-Anne Williams, pages 576–586. URL: https://www.aaai.org/Papers/KR/2004/KR04-060.pdf (cited on page 38).

[SLZ16] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. *SPECTRE: A Fast and Scalable Cryptocurrency Protocol.* Cryptology ePrint Archive, Report 2016/1159. 2016. URL: https://eprint.iacr.org/2016/1159 (cited on pages 3, 156).

[Som12] Fabio Somenzi. *CUDD: CU Decision Diagram Package Release 2.5.0.* Technical report. Department of Electrical, Computer, and Energy Engineering, University of Colorado at Boulder, 2012. URL: http://vlsi.colorado.edu/~fabio/CUDD/ (cited on pages 93, 110).

[Spi11] SpikedMath.com. *Three logicians walk into a bar.* Creative Commons BY-NC-SA. Sept. 22, 2011. URL: http://spikedmath.com/445.html (cited on page 114).

[SSL07] Kaile Su, Abdul Sattar, and Xiangyu Luo. "Model Checking Temporal Logics of Knowledge Via OBDDs". In: *The Computer Journal* 50.4 (2007), pages 403–420. DOI: 10.1093/comjnl/bxm009 (cited on pages 31, 37, 38, 86, 118).

[Swe15] Latanya Sweeney. *Only You, Your Doctor, and Many Others May Know.* 2015. URL: https://techscience.org/a/2015092903/ (cited on page 138).

[Tij71] Robert Tijdeman. "On a telephone problem". In: *Nieuw Archief voor Wiskunde* 3.19 (1971), pages 188–192 (cited on pages 155, 156).

[Vää07] Jouko Väänänen. *Dependence Logic: A New Approach to Independence Friendly Logic.* London Mathematical Society Student Texts. Cambridge: Cambridge University Press, 2007. ISBN: 9780521876599 (cited on page 153).

[Var95] Moshe Y. Vardi. "On the Complexity of Bounded-variable Queries (Extended Abstract)". In: *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* Edited by Mihalis Yannakakis. 1995, pages 266–276. ISBN: 0-89791-730-8. DOI: 10.1145/212433.212474 (cited on page 79).

[VR07] Hans Van Ditmarsch and Ji Ruan. "Model Checking Logic Puzzles". In: *Annales du Lamsade* 8 (2007). URL: https://hal.archives-ouvertes.fr/hal-00188953 (cited on pages 87, 130).

[Wag17] Jana Wagemaker. "Gossip in NetKAT". Master's thesis. ILLC, University of Amsterdam, 2017. URL: https://eprints.illc.uva.nl/1552/ (cited on page 198).

[Wal15] Adam Walker. *cudd: Bindings to the CUDD binary decision diagrams library.* Version 0.1.0.0 for CUDD 2.5.0. 2015. URL: `https://hackage.haskell.org/package/cudd-0.1.0.0` (cited on page 93).

[Wan10] Yanjing Wang. "Epistemic Modelling and Protocol Dynamics". PhD thesis. Amsterdam University, 2010. URL: `https://www.illc.uva.nl/cms/Research/Publications/Dissertations/DS-2010-06.text.pdf` (cited on page 197).

[Wan18] Yanjing Wang. "Beyond Knowing That: A New Generation of Epistemic Logics". In: *Jaakko Hintikka on Knowledge and Game-Theoretical Semantics*. Edited by Hans van Ditmarsch and Gabriel Sandu. Springer International Publishing, 2018, pages 499–533. ISBN: 978-3-319-62864-6. DOI: `10.1007/978-3-319-62864-6_21` (cited on pages 3, 131).

[WC13] Yanjing Wang and Qinxiang Cao. "On axiomatizations of public announcement logic". In: *Synthese* 190.1 (2013), pages 103–134. ISSN: 1573-0964. DOI: `10.1007/s11229-012-0233-5` (cited on page 18).

[WF13] Yanjing Wang and Jie Fan. "Knowing That, Knowing What, and Public Communication: Public Announcement Logic with Kv Operators". In: *Twenty-Third International Joint Conference on Artificial Intelligence – IJCAI '13*. 2013, pages 1147–1154. URL: `https://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6742` (cited on pages 138, 139).

[WF14] Yanjing Wang and Jie Fan. "Conditionally Knowing What". In: *Advances in Modal Logic*. Edited by Rajeev Goré, Barteld Kooi, and Agi Kurucz. Volume 10. 2014, pages 569–587. URL: `http://www.aiml.net/volumes/volume10/Wang-Fan.pdf` (cited on pages 138, 139).

[WL12] Yanjing Wang and Yanjun Li. "Y.: Not all those who wander are lost: Dynamic epistemic reasoning in navigation". In: *Advances in Modal Logic*. Edited by Thomas Bolander, Torben Braüner, Silvio Ghilardi, and Lawrence Moss. Volume 9. 2012, pages 559–580. ISBN: 978-1-84890-068-4. URL: `http://www.aiml.net/volumes/volume9/Wang-Li.pdf` (cited on pages 128, 130).

Samenvatting

Dynamische Epistemische Logica (DEL) kan complexe informatiescenario's modelleren op een intuïtieve manier. Bestaande implementaties zijn echter gebaseerd op expliciete representaties die alleen voor kleine modellen werken. Daarom weten we niet hoe bruikbaar DEL is voor grotere modellen en echte problemen.

Voor temporele logica's daarentegen bestaan geavanceerde symbolische methodes voor modelverificatie (*model checking*) die met succes worden toegepast, bijvoorbeeld voor protocol- en hardwareverificatie. Symbolische modelverificatie voor temporele logica is efficiënt en kan met zeer grote modellen werken.

In dit proefschrift slaan we een brug: nieuwe representaties van Kripke-modellen als zogenaamde kennis- en geloofsstructuren (*knowledge and belief structures*) die geschikt zijn voor symbolische methodes. Voor complexe epistemische gebeurtenissen en feitelijke verandering introduceren we kennis- en geloof-veranderende structuren (*knowledge and belief transformers*), een symbolische vervanging voor actiemodellen (*action models*).

Naast een gedetailleerde uitleg van de theorie presenteert het proefschrift SMCDEL, een Haskell-implementatie van symbolische modelverificatie voor DEL gebaseerd op binaire beslissingsdiagrammen.

Onze nieuwe methodes kunnen bekende epistemische problemen en puzzels sneller oplossen dan bestaande implementaties van DEL. We vergelijken de prestatie van onze nieuwe methodes ook met die van bestaande implementaties van temporele logica. De resultaten tonen aan dat DEL de strijd met de concurrentie aankan.

We kijken verder naar twee specifieke varianten van DEL voor concrete toepassingen. Ten eerste introduceren we Public Inspection Logic (PIL), een nieuwe logica voor de kennis van variabelen en de dynamiek ervan. Ten tweede bestuderen we het dynamische roddelprobleem en we analyseren het in epistemische logica. We laten zien dat bestaande roddelprotocollen kunnen worden verbeterd, maar we bewijzen ook dat het "Learn New Secrets"-protocol niet zodanig kan worden versterkt dat het altijd succesvol uitgevoerd kan worden.

Dit onderzoek toont aan dat DEL een in de praktijk bruikbare logica is en efficiënt geïmplementeerd kan worden. Het opent de deur zowel naar toepassingen als naar verdere ontwikkeling van de theorie van symbolische representatie.

Abstract

Dynamic Epistemic Logic (DEL) can model complex information scenarios in a way that appeals to logicians. However, its existing implementations are based on explicit model checking which can only deal with small models, so we do not know how DEL performs for larger and real-world problems.

For temporal logics, in contrast, symbolic model checking has been developed and successfully applied, for example in protocol and hardware verification. Symbolic model checkers for temporal logics are very efficient and can deal with very large models.

In this thesis we build a bridge: new faithful representations of DEL models as so-called knowledge and belief structures that allow for symbolic model checking. For complex epistemic and factual change we introduce knowledge and belief transformers, a symbolic replacement for action models.

Besides a detailed explanation of the theory, the thesis presents SMCDEL: a Haskell implementation of symbolic model checking for DEL using Binary Decision Diagrams.

Our new methods can solve well-known benchmark problems in epistemic scenarios much faster than existing methods for DEL. We also compare the performance of the implementation to existing model checkers for temporal logics and show that DEL can compete with the established frameworks.

We zoom in on two specific variants of DEL for concrete applications. First, we introduce Public Inspection Logic (PIL), a new framework for the knowledge of variables and its dynamics. Second, we study the dynamic gossip problem and how it can be analyzed with epistemic logic. We show that existing gossip protocols can be improved, but also prove that no perfect strengthening of the "Learn New Secrets" protocol exists.

This research allows DEL to join the club of efficiently implemented and applicable logics. It opens up new directions, both towards real-world applications and further development of the theory of symbolic representation.

Titles in the ILLC Dissertation Series:

ILLC DS-2009-01: **Jakub Szymanik**
Quantifiers in TIME and SPACE. Computational Complexity of Generalized Quantifiers in Natural Language

ILLC DS-2009-02: **Hartmut Fitz**
Neural Syntax

ILLC DS-2009-03: **Brian Thomas Semmes**
A Game for the Borel Functions

ILLC DS-2009-04: **Sara L. Uckelman**
Modalities in Medieval Logic

ILLC DS-2009-05: **Andreas Witzel**
Knowledge and Games: Theory and Implementation

ILLC DS-2009-06: **Chantal Bax**
Subjectivity after Wittgenstein. Wittgenstein's embodied and embedded subject and the debate about the death of man.

ILLC DS-2009-07: **Kata Balogh**
Theme with Variations. A Context-based Analysis of Focus

ILLC DS-2009-08: **Tomohiro Hoshi**
Epistemic Dynamics and Protocol Information

ILLC DS-2009-09: **Olivia Ladinig**
Temporal expectations and their violations

ILLC DS-2009-10: **Tikitu de Jager**
"Now that you mention it, I wonder...": Awareness, Attention, Assumption

ILLC DS-2009-11: **Michael Franke**
Signal to Act: Game Theory in Pragmatics

ILLC DS-2009-12: **Joel Uckelman**
More Than the Sum of Its Parts: Compact Preference Representation Over Combinatorial Domains

ILLC DS-2009-13: **Stefan Bold**
Cardinals as Ultrapowers. A Canonical Measure Analysis under the Axiom of Determinacy.

ILLC DS-2010-01: **Reut Tsarfaty**
Relational-Realizational Parsing

ILLC DS-2010-02: **Jonathan Zvesper**
Playing with Information

ILLC DS-2010-03: **Cédric Dégremont**
The Temporal Mind. Observations on the logic of belief change in interactive systems

ILLC DS-2010-04: **Daisuke Ikegami**
Games in Set Theory and Logic

ILLC DS-2010-05: **Jarmo Kontinen**
Coherence and Complexity in Fragments of Dependence Logic

ILLC DS-2010-06: **Yanjing Wang**
Epistemic Modelling and Protocol Dynamics

ILLC DS-2010-07: **Marc Staudacher**
Use theories of meaning between conventions and social norms

ILLC DS-2010-08: **Amélie Gheerbrant**
Fixed-Point Logics on Trees

ILLC DS-2010-09: **Gaëlle Fontaine**
Modal Fixpoint Logic: Some Model Theoretic Questions

ILLC DS-2010-10: **Jacob Vosmaer**
Logic, Algebra and Topology. Investigations into canonical extensions, duality theory and point-free topology.

ILLC DS-2010-11: **Nina Gierasimczuk**
Knowing One's Limits. Logical Analysis of Inductive Inference

ILLC DS-2010-12: **Martin Mose Bentzen**
Stit, Iit, and Deontic Logic for Action Types

ILLC DS-2011-01: **Wouter M. Koolen**
Combining Strategies Efficiently: High-Quality Decisions from Conflicting Advice

ILLC DS-2011-02: **Fernando Raymundo Velazquez-Quesada**
Small steps in dynamics of information

ILLC DS-2011-03: **Marijn Koolen**
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

ILLC DS-2011-04: **Junte Zhang**
System Evaluation of Archival Description and Access

ILLC DS-2011-05: **Lauri Keskinen**
Characterizing All Models in Infinite Cardinalities

ILLC DS-2011-06: **Rianne Kaptein**
Effective Focused Retrieval by Exploiting Query Context and Document Structure

ILLC DS-2011-07: **Jop Briët**
Grothendieck Inequalities, Nonlocal Games and Optimization

ILLC DS-2011-08: **Stefan Minica**
Dynamic Logic of Questions

ILLC DS-2011-09: **Raul Andres Leal**
Modalities Through the Looking Glass: A study on coalgebraic modal logic and their applications

ILLC DS-2011-10: **Lena Kurzen**
Complexity in Interaction

ILLC DS-2011-11: **Gideon Borensztajn**
The neural basis of structure in language

ILLC DS-2012-01: **Federico Sangati**
Decomposing and Regenerating Syntactic Trees

ILLC DS-2012-02: **Markos Mylonakis**
Learning the Latent Structure of Translation

ILLC DS-2012-03: **Edgar José Andrade Lotero**
Models of Language: Towards a practice-based account of information in natural language

ILLC DS-2012-04: **Yurii Khomskii**
Regularity Properties and Definability in the Real Number Continuum: idealized forcing, polarized partitions, Hausdorff gaps and mad families in the projective hierarchy.

ILLC DS-2012-05: **David García Soriano**
Query-Efficient Computation in Property Testing and Learning Theory

ILLC DS-2012-06: **Dimitris Gakis**
Contextual Metaphilosophy - The Case of Wittgenstein

ILLC DS-2012-07: **Pietro Galliani**
The Dynamics of Imperfect Information

ILLC DS-2012-08: **Umberto Grandi**
Binary Aggregation with Integrity Constraints

ILLC DS-2012-09: **Wesley Halcrow Holliday**
Knowing What Follows: Epistemic Closure and Epistemic Logic

ILLC DS-2012-10: **Jeremy Meyers**
Locations, Bodies, and Sets: A model theoretic investigation into nominalistic mereologies

ILLC DS-2012-11: **Floor Sietsma**
Logics of Communication and Knowledge

ILLC DS-2012-12: **Joris Dormans**
Engineering emergence: applied theory for game design

ILLC DS-2013-01: **Simon Pauw**
Size Matters: Grounding Quantifiers in Spatial Perception

ILLC DS-2013-02: **Virginie Fiutek**
Playing with Knowledge and Belief

ILLC DS-2013-03: **Giannicola Scarpa**
Quantum entanglement in non-local games, graph parameters and zero-error information theory

ILLC DS-2014-01: **Machiel Keestra**
Sculpting the Space of Actions. Explaining Human Action by Integrating Intentions and Mechanisms

ILLC DS-2014-02: **Thomas Icard**
The Algorithmic Mind: A Study of Inference in Action

ILLC DS-2014-03: **Harald A. Bastiaanse**
Very, Many, Small, Penguins

ILLC DS-2014-04: **Ben Rodenhäuser**
A Matter of Trust: Dynamic Attitudes in Epistemic Logic

ILLC DS-2015-01: **María Inés Crespo**
Affecting Meaning. Subjectivity and evaluativity in gradable adjectives.

ILLC DS-2015-02: **Mathias Winther Madsen**
The Kid, the Clerk, and the Gambler - Critical Studies in Statistics and Cognitive Science

ILLC DS-2015-03: **Shengyang Zhong**
Orthogonality and Quantum Geometry: Towards a Relational Reconstruction of Quantum Theory

ILLC DS-2015-04: **Sumit Sourabh**
Correspondence and Canonicity in Non-Classical Logic

ILLC DS-2015-05: **Facundo Carreiro**
Fragments of Fixpoint Logics: Automata and Expressiveness

ILLC DS-2016-01: **Ivano A. Ciardelli**
Questions in Logic

ILLC DS-2016-02: **Zoé Christoff**
Dynamic Logics of Networks: Information Flow and the Spread of Opinion

ILLC DS-2016-03: **Fleur Leonie Bouwer**
What do we need to hear a beat? The influence of attention, musical abilities, and accents on the perception of metrical rhythm

ILLC DS-2016-04: **Johannes Marti**
Interpreting Linguistic Behavior with Possible World Models

ILLC DS-2016-05: **Phong Lê**
Learning Vector Representations for Sentences - The Recursive Deep Learning Approach

ILLC DS-2016-06: **Gideon Maillette de Buy Wenniger**
Aligning the Foundations of Hierarchical Statistical Machine Translation

ILLC DS-2016-07: **Andreas van Cranenburgh**
Rich Statistical Parsing and Literary Language

ILLC DS-2016-08: **Florian Speelman**
Position-based Quantum Cryptography and Catalytic Computation

ILLC DS-2016-09: **Teresa Piovesan**
Quantum entanglement: insights via graph parameters and conic optimization

ILLC DS-2016-10: **Paula Henk**
Nonstandard Provability for Peano Arithmetic. A Modal Perspective

ILLC DS-2017-01: **Paolo Galeazzi**
Play Without Regret

ILLC DS-2017-02: **Riccardo Pinosio**
The Logic of Kant's Temporal Continuum

ILLC DS-2017-03: **Matthijs Westera**
Exhaustivity and intonation: a unified theory

ILLC DS-2017-04: **Giovanni Cinà**
Categories for the working modal logician

ILLC DS-2017-05: **Shane Noah Steinert-Threlkeld**
Communication and Computation: New Questions About Compositionality

ILLC DS-2017-06: **Peter Hawke**
The Problem of Epistemic Relevance

ILLC DS-2017-07: **Aybüke Özgün**
Evidence in Epistemic Logic: A Topological Perspective

ILLC DS-2017-08: **Raquel Garrido Alhama**
Computational Modelling of Artificial Language Learning: Retention, Recognition & Recurrence

ILLC DS-2017-09: **Miloš Stanojević**
Permutation Forests for Modeling Word Order in Machine Translation

ILLC DS-2018-01: **Berit Janssen**
Retained or Lost in Transmission? Analyzing and Predicting Stability in Dutch Folk Songs

ILLC DS-2018-02: **Hugo Huurdeman**
Supporting the Complex Dynamics of the Information Seeking Process

ILLC DS-2018-03: **Corina Koolen**
Reading beyond the female: The relationship between perception of author gender and literary quality

ILLC DS-2018-04: **Jelle Bruineberg**
Anticipating Affordances: Intentionality in self-organizing brain-body-environment systems

ILLC DS-2018-05: **Joachim Daiber**
Typologically Robust Statistical Machine Translation: Understanding and Exploiting Differences and Similarities Between Languages in Machine Translation

ILLC DS-2018-06: **Thomas Brochhagen**
Signaling under Uncertainty

ILLC DS-2018-07: **Julian Schlöder**
Assertion and Rejection

ILLC DS-2018-08: **Srinivasan Arunachalam**
Quantum Algorithms and Learning Theory

ILLC DS-2018-09: **Hugo de Holanda Cunha Nobrega**
Games for functions: Baire classes, Weihrauch degrees, transfinite computations, and ranks

ILLC DS-2018-10: **Chenwei Shi**
Reason to Believe

ILLC DS-2018-11: **Malvin Gattinger**
New Directions in Model Checking Dynamic Epistemic Logic