Suitable logics: provability, temporal laws, and formalization

Ana de Almeida Gabriel Vieira Borges

UNIVERSITAT DE BARCELONA

Suitable logics: provability, temporal laws, and formalization

Ana de Almeida Gabriel Vieira Borges

Thesis to obtain the Doctorate Degree in

Mathematics and Computer Science

Supervisor: Dr. Joost J. Joosten

July 2023

Logic takes care of itself; all we have to do is to look and see how it does it.

Ludwig Wittgenstein

Foreword

Dear reader,

The story of this thesis starts with a call for an industrial PhD position in Barcelona, a kind of position I was not aware of at the time. This position was a collaboration between the University of Barcelona and Formal Vindications S.L., with the following stated goals: development and verification of legal software in Coq, formalization of parts of logic/mathematics, and research on topics related with ordinal analysis via modal logic and reflection principles.

Even though my understanding of Coq at the time consisted of "it's a proof assistant built on the Curry-Howard isomorphism," and my familiarity with the other research topics amounted to "this looks like proof theory," it did seem fascinating, and so I applied and was accepted for the position.

What followed were several years vaguely guided by the above goals, which led to research in vastly different subjects. I was right that it would be fascinating, and also weird and frustrating and exciting. Verification of software requires a clear picture of what the software is supposed to do, which is surprisingly hard to obtain with legal software in particular. This led to the formal analysis of some European regulations, which is one of the subjects of this thesis. On the other hand, and quite independently, I spent a significant amount of time doing proof theoretical research, which focused mostly on provability logic. Finally, I did end up formalizing both legal software and mathematics, and found it very satisfying.

Thus I present to you a thesis that is simultaneously about provability logic, formal analysis of European regulations, and formalization. I hope you find as much joy in it as I did.

Ana Borges

Acknowledgements

First and foremost, I would like to thank my supervisor Joost J. Joosten for his support during the completion of this thesis. I thank him for his enthusiasm in suggesting projects and discussing ideas, for pushing me to focus on the things that most interested me, for providing ample opportunities to discuss my work at seminars, workshops, and conferences, and for introducing me to several topics and people. I also thank him for his many worthwhile suggestions on my academic writing and on this thesis in particular. He greatly encouraged my growth as a researcher.

Second, I would like to thank Guillermo Errezil for creating the law formalization project, for sharing his excitement and ideas, and for supporting both the overall project and this thesis in particular, even the parts further from his heart. As I believe he would say, life is complicated and one should make the most of that.

I would like to thank the many logicians who formed part of my academic social network and who shared their ideas or gave me advice, in particular Lev Beklemishev, Bruno Dinis, David Fernández-Duque, Fernando Ferreira, Rosalie Iemhoff, Fedor Pakhomov, and Andreas Weiermann.

I could not have learned Coq and became a passable Coq programmer without the help of the Coq community. The teachers at the MathComp Winter School 2017 and Cyril Cohen in particular were fundamental for this, and I am very grateful to them. A special thanks to the Coq developers who guided my first steps into Coq development: Ali Caglayan, Emilio Jesús Gallego Arias, Guillaume Melquiond, and Pierre Roux. Thank you also to Théo Zimmermann for sponsoring my attendance to the Federated Logic Conference 2022 and to Yves and Janet Bertot for very kindly solving a worrying logistical problem I had during a Coq Users and Developers Workshop. Finally, I would like to thank everyone who hangs out at the Coq Zulip chat, who helped me debug a number of problems and in general created a welcoming environment. Besides the people already mentioned in this paragraph and among many others, this includes: Yannick Forster, Paolo Giarrusso, Jason Gross, Hugo Herbelin, Jasper Hugunin, Dominik Kirst, Dominique Larchey-Wendling, and Karl Palmskog. Without their help, I might still be frowning at a piece of Coq code right now.

I am grateful to my PhD colleagues all over the world, who I enjoyed meeting or emailing with more or less regularity. Together we commiserated on common woes and exulted in each other's successes. We discussed open questions and shared LaTeX tips. My PhD experience was enriched by them. Thank you: Pedro Filipe, Iris van der Giessen, Eduardo Hermo, Luka Mikec, Konstantinos Papafilippou, Raúl Penaguião, Nika Pona, Sofía Santiago, and Ian Shillito.

I am very thankful to my Formal Vindications S.L. colleagues as well. Thank you to Eric Sancho and Aleix Solé for being enthusiastic students, for taking up the most soul-killing responsibilities (and surviving the ordeal), for helping me with logistics on numerous occasions, and for welcoming me as a friend. Gracias también a Aleix por ayudarme a practicar mi español. Thank you to my fellow coders Quim Casals, Juan Conejero, Mireia González, and Eduardo Hermo: coding with them was a pleasure. Thank you in particular to Juan for keeping me virtual company during the pandemic, it was (and is) a delight to share my struggles and wins with him. Thank you also to Mireia for always having my back, her competence and friendship are incredibly important to me.

A special thanks to everyone who directly or indirectly proofread parts of this thesis, namely: Carolina, David, Eduardo, Eric, Joost, Juan, Mireia, Quim, Sofía, Yannick, the anonymous reviewers of my publications, my godmother, and my parents. There will always be things to improve, but without their help there would be even more!

Thank you to my Portuguese friends. Even though I moved away, and even though we spent the pandemic years stuck at home, they were there for me from afar, and we managed to find opportunities to be together. Thank you “Verdadeiros”: Inês Antunes, Diogo e Tiago Borges, Carolina Caldeirinha, Maria do Mar da Câmara Pereira, and Jorge Sacadura. Thank you also to Daniel Sousa for teaching me many valuable things. Finally, a heartfelt thank you to Raquel Gonçalves, who was and is both a friend and a coach whenever needed.

I am thankful to my family, for their unconditional love and support throughout my life. Even though they are half-convinced I wrote my thesis in Kryptonian, everyone still offered to help and indeed helped with their support, long conversations, laughs, shared meals, and care. Obrigada queridos avós Lúcia, Cândida, e César, madrinha Gui e padrinho Quim, tios João, Sílvia, Ricardo, e Cândida, prima Candidinha, e primos Diogo, Tiago, Francisco, Vasco, e Clara.

It is difficult to put into words my appreciation for the most fundamental people in my life. My brother Zé is one of my closest friends. We coexist perfectly, whether in extended silence or in excited conversation. He complements my personality and has taught me many, many things. I deeply thank him for his open and lively presence in my life. My father takes “acceptance of thy daughter” to the next level. He is completely chill about any decision I make, which makes me feel empowered and safe. With him I learned I could always trust myself, as he always trusted in me. As an academic he is a wonderful role model of passionate and organized work. Obrigada Papá. My mother was always here to help me, always available, always engaged. Even with a hectic schedule, she has always made time for me and my interests. She read everything I wrote and shared with her, she stimulated my thought, marveled at my development, and nurtured and accepted my uniqueness, even when she didn’t agree with me. She takes it as her job to support me in any way she possibly can. She’s an extremely energetic and generous mother. Words are too weak to describe her foundational role in my life. Obrigada Mamã.

A significant portion of this thesis was developed as part of the project RTC-2017-6740-7 funded by the Ministry of Science, Innovation and Universities, the State Agency for Research and the European Regional Development Fund. Thank you.

Abstract

This thesis explores various aspects of logic, encompassing provability logic, logical analysis of law, and formalization of both mathematics and software. We are particularly interested in the aspect of suitability, both in the sense of finding expressible and tractable fragments of certain logical systems, and in the sense of finding appropriate formal systems for describing certain legal texts.

In the domain of provability logic, we introduce two novel calculi: the Worm Calculus (WC) and the Quantified Reflection Calculus with one modality (QRC_1). Both WC and QRC_1 are inspired by the Reflection Calculus (RC), introduced by Dashkov in 2012. All three logics are strictly positive provability logics, with WC and RC being propositional and polymodal, while QRC_1 features a quantified language with a single modality.

Worms are words in a numerical alphabet that have many possible readings, particularly as iterated consistency statements. Although the language of worms is very simple, it is remarkably expressive, and known to fully describe the closed fragment of RC. The Worm Calculus is a calculus in the language of worms that illustrates this power: RC is shown to be conservative over WC.

Vardanyan showed in 1986 that the quantified provability logic of Peano Arithmetic ($\mathsf{QPL}(\mathsf{PA})$) is Π^0_2-complete, and in particular not recursively axiomatizable. We investigate the strictly positive fragment of $\mathsf{QPL}(\mathsf{PA})$, showing that QRC_1 is a decidable axiomatization of that fragment. In the process, we prove soundness and completeness of QRC_1 with respect to Kripke semantics and two flavors of arithmetical semantics. We also see that QRC_1 is the strictly positive fragment of $\mathsf{QK4}$ and of QGL, and of any logic in between those.

In the realm of law and inspired by a collaboration with industry, we focus on two European transport regulations, examining certain articles with curious mathematical properties. Then we identify fragments of Monadic Second-order Logic capable of expressing specific segments of one of these regulations, and show how other fragments are less suitable. This effort illustrates some issues with the way the current regulations specify algorithms, and it hints at the role of model checking as a useful legal tool.

Lastly, we delve into the formal verification of both mathematics and software using Coq, presenting two case studies. The first case study involves the formalization of modal logical results on QRC_1. We describe the overall formalization strategy, focusing on the difficulties and adaptations of both definitions and proofs helpful for the formalization. Due to this process, we were able to identify a small number of improvements to the original proofs.

The formalization of software is somewhat different from the formalization of mathematics, although they share many particularities. We use a general framework for software formalization that starts by writing a basic specification for each function, which is then iteratively refined with better algorithms, data structures, and error handling, culminating in extraction to OCaml. Our case study is the formalization of UTC calendars, particularly functions to translate between human-readable times and timestamps, as well as functions to perform time arithmetic. This is a first step towards the formalization of laws that depend on time keeping, of which there are many, including the ones studied here.

Keywords

Modal logic, provability logic, strictly positive logic, quantified modal logic, feasible fragments, monadic second order logic, formal analysis of law, formal verification, Coq

Resumen

Esta tesis en lógica formal abarca la lógica de demostrabilidad, el análisis lógico del derecho, y la formalización de las matemáticas y del software.

Se introducen dos nuevos cálculos de demostrabilidad estrictamente positivos inspirados en el Reflection Calculus (RC): el Worm Calculus (WC) y el Quantified Reflection Calculus con una modalidad (QRC_1). Tanto RC como WC son lógicas proposicionales y polimodales, mientras que QRC_1 tiene un lenguaje cuantificado con una sola modalidad.

Los “worms” son palabras en un alfabeto numérico con múltiples interpretaciones, en particular como enunciados de consistencia iterada. El Worm Calculus es un cálculo en el lenguaje de los “worms” que describe la totalidad del fragmento cerrado de RC.

Vardanyan demostró que la lógica cuantificada de demostrabilidad de la Aritmética de Peano ($\mathsf{QPL}(\mathsf{PA})$) es Π^0_2-completa y, en particular, no axiomatizable de manera recursiva. Esta tesis investiga el fragmento estrictamente positivo de $\mathsf{QPL}(\mathsf{PA})$ y presenta QRC_1 como una axiomatización decidible de ese fragmento. Se demuestra la corrección y completitud de QRC_1 mediante semántica de Kripke y dos semánticas aritméticas, y que QRC_1 es el fragmento estrictamente positivo de las lógicas entre $\mathsf{QK4}$ y QGL, inclusive.

Asimismo, la tesis incluye un estudio aplicado al ámbito del derecho en el que se examinan dos regulaciones de transporte europeas con remarcables propiedades matemáticas. Se identifican fragmentos de Lógica Monádica de Segundo Orden capaces de expresar segmentos específicos, y otros menos adecuados, sugiriendo que la verificación de modelos puede ser una herramienta legal útil.

Se estudia la verificación formal de matemáticas y software utilizando Coq a través de dos ejemplos. Se describe la formalización de resultados lógicos modales sobre QRC_1, abordando los desafíos y adaptaciones en definiciones y pruebas, e identificando mejoras a las pruebas originales.

Para la formalización del software se emplea un marco general comenzando con una especificación básica de funciones, refinándola de manera iterativa con mejores algoritmos, estructuras de datos y manejo de errores, culminando con extracción a OCaml. Se formaliza el calendario UTC, incluyendo funciones de traducción entre tiempos y timestamps junto con aritmética del tiempo. Este es un primer paso para la formalización de leyes dependientes del tiempo.

Palabras clave

Lógica modal, lógica de demostrabilidad, lógica estrictamente positiva, lógica modal cuantificada, fragmentos factibles, lógica monádica de segundo orden, análisis formal de leyes, verificación formal, Coq

Contents

1 Introduction

Contents

1.1 Preamble

This thesis is the product of several years of curiosity and research in the broad field of logic. A common theme is suitability: which logic, or formalism, or strategy is best for a given purpose? Or if not provably best, then at least better than the previously available ones? In particular, we consider (more specific versions of) the following questions:

1. Is it possible to axiomatize a fragment of a logic within that fragment? (Chapter 3)
2. Is it possible to restrict the language of a given intractable logic and obtain a weaker but still useful and tractable version? (Chapters 4 and 5)
3. Is it possible to express certain legal texts within certain logical systems? (Chapter 8)
4. What is a good way of translating certain systems to a computer-readable and checkable format? (Chapters 10 and 11)

We have not explored the above questions in their full generality. Instead, we chose specific examples of logics, regulations, and systems that seemed particularly interesting, relevant, or amenable to study. In the process, we branched out into some related questions or tasks, and this document is the end result.

This thesis is split into three parts, which mimic three clusters suggested by the questions above. Part I is dedicated to provability logics, and presents our positive answers to the first two questions. Part II is inspired by certain European road transport regulations, where we explore some logical shortcomings of those regulations (Chapter 7), as well as how certain articles are either expressible or not (reasonably) expressible in fragments of Monadic Second-order Logic (Chapter 8). Finally, Part III describes Coq formalizations of one of the two systems described in Part I (Chapter 10), as well as of UTC calendars with leap seconds (Chapter 11), which are a pre-requisite for formalizing the regulations studied in Part II. Each part includes an introductory chapter describing the relevant state of the art as well as some background knowledge (Chapters 2, 6, and 9, respectively).

The remainder of this general introduction focuses on each part by itself, ending with a list of references of our contributions.

1.2 Provability

The concept of “provable in a theory T” can be defined as a predicate $\Box_T$ within sufficiently strong theories of arithmetic such as Peano Arithmetic (PA). This provability predicate enjoys certain provable structural properties, such as distributivity:

$$\Box_T(\varphi \to \psi) \to (\Box_T\varphi \to \Box_T\psi).$$

This distributivity property expresses *modus ponens* within T: if $\varphi \to \psi$ is provable in T and φ is also provable in T then ψ is provable in T. Such structural properties can be studied via modal logic by abstracting the definition of $\Box_T$ to the modality $\Box$ and endowing it with relevant properties via modal logical axioms. The standard provability logic is GL [49], which has well-studied fragments as well as extensions. One such extension is GLP [139], which instead of a single $\Box$ modality includes several, written $[\alpha]$ for some ordinal α. These different modalities represent different provability strengths, such as provability under oracles.

The Reflection Calculus (RC) [78] is the strictly positive fragment of GLP, whose language is made up of $\top$, propositional symbols, conjunctions, and diamonds $\langle\alpha\rangle$. The diamonds can be read as consistency statements, where $\langle\alpha\rangle\varphi$ is stronger than $\langle\beta\rangle\varphi$ when $\alpha > \beta$.

We focus on the closed fragment of RC, denoted by RC^0. In particular, we are interested in studying worms, which are the conjunction-free formulas of RC^0, i.e., the formulas built only from diamonds and $\top$. The interplay between the several diamonds of RC^0 leads to remarkable expressivity, and in fact every closed formula of RC is equivalent to a worm [136]. This motivates our first question:

1. Is it possible to axiomatize the worm fragment of RC^0 using worms?

The answer is yes. We introduce the Worm Calculus (WC) in Chapter 3 and show that it has the same expressive power as RC^0. We also investigate universal models for WC, and see that a large class of such models inherits much of the complexity of Ignatiev's model [135], a well-known universal model for GLP^0.

The next topic tackled in Part I pertains to quantified provability logic. In its full generality, the quantified provability logic of PA (QPL(PA)) is not recursively axiomatizable [203], which leads us to ask:

2. Is it possible to restrict the language of QPL(PA) and obtain a weaker but still useful and tractable version?

The answer is positive again. We introduce the Quantified Reflection Calculus with one modality (QRC_1) in Chapter 4, which is shown to axiomatize the strictly positive fragment of QPL(PA) in Chapter 5. In fact, our results are more general: QRC_1 is the strictly positive quantified provability logic of any sound computably enumerable (c.e.) extension of Elementary Arithmetic (EA) and the quantified reflection logic of any sound c.e. extension of EA plus Σ^0_1 collection. As pre-requisites, we see that QRC_1 is sound and complete with respect to Kripke sheaf semantics, and furthermore enjoys the finite model property. Thus, QRC_1 is decidable, a remarkable drop in complexity when compared with the Π^0_2-complete QPL(PA). Finally, QRC_1 is the strictly positive fragment of the standard extension of GL to a first-order setting (QGL).

The concepts introduced above are properly defined, contextualized, and explained during Chapter 2, which serves as a more complete introduction to Part I.

1.3 Temporal laws

Some regulations specify algorithms. These algorithms are implemented as software and used to make decisions on the legality of certain circumstances. However, there is often a mismatch between the regulated specification and its implementation. Sometimes this is unavoidable, because the specification failed to account for corner cases or would require unfeasible amounts of memory or computation.

We focus on two European regulations on the road transport of people and goods, namely Regulation (EU) 2016/799 [90] and Regulation (EC) 561/2006 [88] (see Section 6.2). These are of particular interest to Formal Vindications S.L. and Guretruck S.L., which sponsored and motivated part of this thesis. One particularity of these regulations is that they concern themselves with time-based events, some of which pegged to the UTC calendar. Furthermore, they include several informal descriptions of algorithms, which are implemented both in tachographs (devices installed in vehicles that record the driver's activity) and in specialized software used to produce legal decisions.

For example, drivers must not spend too much consecutive time driving without a rest, and there are a number of restrictions on the minimum amount of consecutive time to be spent resting each day and week (easily analogized to nights and weekends). Since the drivers' activities are recorded in the tachograph's logs, one can check these restrictions by analyzing those logs, which in practice is done automatically.

However, we find some discrepancies between our immediate intuitive interpretations of certain parts of these regulations and our updated interpretations after a second (or fifth!) reading. We show that a part of Regulation (EU) 2016/799 [90] describes an inductive algorithm lacking an appropriate base case, that the choice of calendar (for example, with or without leap seconds) radically changes outcomes in some cases, and that parts of Regulation (EC) 561/2006 [88] are non-local. These findings are described in Chapter 7.

Be that as it may, after fixing reasonable interpretations we are interested in expressing them within some mathematical formalism. This has two clear advantages: first, it is a much more robust way of describing restrictions and algorithms; second, for some formalisms we already have solvers or model checkers, avoiding the need for purpose-built tools. To that end, we focus on Monadic Second-order Logic (MSO) and its fragments, in particular Linear Temporal Logic (LTL), described in Chapter 6. Thus our question can be stated as:

3. Is it possible to express parts of Regulation (EC) 561/2006 [88] within fragments of MSO?

The answer depends on the parts in question and on the specific fragments of MSO. Regulations have many aspects and we do not attempt to implement them in their full generality. However, some parts have more or less clear embeddings in MSO and then the task reduces to finding smaller fragments on which they are expressible. In Chapter 8 we see how to describe two specific articles of Regulation (EC) 561/2006 [88] within LTL and the Σ_1^1 fragment of MSO, respectively. Then we show how the language of LTL without the "until" modality is not enough for either, and that even with "until" one of the articles is not expressible via low-complexity formulas.

1.4 Formalization

Formalization is the process of translating some informally presented statement (be it a definition, a theorem, a proof, or something else) into a formal system with clear and well-defined rules. For example, a proof of $\varphi \to \varphi$ might be omitted during an informal argument, whereas a formalization would need to settle the language to which φ belongs and the system in which $\varphi \to \varphi$ is to be proved, complete with an axiomatization or enough results with which to otherwise deduce the goal, and then describe how to do so in detail.

Formalizing proofs can be tedious and in many contexts it is pointless. One may find mistakes while trying to formalize wrong results, but it is not necessarily easier to check a formalized result by hand than it is to check an informal and well-written proof, mainly due to the increased size typical of formalizations.

This changes with access to proof assistants. Proof assistants, also known as interactive theorem provers, are software tools that aid the process of formalization. They provide an interface between the user and full formalizations, permitting a level of informal reasoning through automation and providing complex record-keeping beyond what people are usually capable of.

Formalization through proof assistants is then useful to check one's (or others') work, specially when there are reasons for doubt. It is also a worthwhile exercise even if there is no real question of correctness, because formalizing a result forces one to understand it at a level beyond what is usually needed to informally prove it, and often leads to simplified statements or proofs.

It is also possible to formalize software, which usually starts with either the software to be formalized or a specification thereof. In this context it is common for the correctness results to not be informally stated or proved anywhere before being formalized.

The formalizations described in Part III use the Coq proof assistant [66]. More details on computer-aided formalization in general and Coq in particular can be found in Chapter 9.

We now restate the question for this part:

4. What is a good way of formalizing RC^0, WC, QRC_1, and UTC in Coq?

The difficulty lies in choosing the right abstractions and proofs, which are not always the ones used in informal presentations. Even then, each formalization may be very time-consuming, specially if the matter at hand relies on a large body of previous mathematical knowledge or has an implicit formalization very different from the typical informal presentation.

Symbolic logic is an easy formalization target in general, as it typically starts with simple languages and axiomatizations. Thus, formal representations of RC^0, WC, and QRC_1 can readily be obtained and used to prove simple RC^0, WC, or QRC_1 theorems. More complex systems such as PA can also be formalized [179, 103], but proving facts within PA can quickly become tedious when compared with informal proofs that do not detail each step in a Hilbert-style proof.

We focus on formalizing the main modal logical results of Chapters 3 (RC^0 and WC) and 4 (QRC_1), and leave formalizations of the remaining results as future work. Since the formalization of the results pertaining to RC^0 and WC is rather straightforward, we do not describe it in this thesis and instead refer the reader to the source code and its documentation [3].

The formalization of the modal results on QRC_1 is described in Chapter 10. We focus on interesting design decisions and the difficulties that led to different and simplified proofs.

On a somewhat different register, we delve into software verification in Chapter 11. We formalize UTC calendars with leap seconds and without time zones. The main functions are translations between times in a year-month-day-hour-minute-second format and timestamps. We also include basic time arithmetic functions. Representing moments in time and knowing how to add and subtract times and durations is a necessary prerequisite for the formalization of the regulations discussed in Part II.

1.5 Contributions

Most of the results described here have been published in stand alone documents, many of which with co-authors. They are as follows, roughly in order of appearance in this thesis:

[12] de Almeida Borges, A., & Joosten, J. J. (2018). The Worm Calculus. In G. Bezhanishvili, G. D'Agostino, G. Metcalfe, & T. Studer (Eds.) *Advances in Modal Logic 12*, (pp. 13–27). College Publications.
URL `http://www.aiml.net/volumes/volume12/deAlmeidaBorges-Joosten.pdf`

[13] de Almeida Borges, A., & Joosten, J. J. (2020). Quantified Reflection Calculus with one modality. In N. Olivetti, R. Verbrugge, S. Negri, & G. Sandu (Eds.) *Advances in Modal Logic 13*, (pp. 13–32). College Publications.
URL `http://www.aiml.net/volumes/volume13/deAlmeidaBorges-Joosten.pdf`

[14] de Almeida Borges, A., & Joosten, J. J. (2022). An escape from Vardanyan's Theorem. *The Journal of Symbolic Logic*, (pp. 1–26).
URL `https://www.cambridge.org/core/journals/journal-of-symbolic-logic/article/an-escape-from-vardanyans-theorem/03E13D4C282563F918AF77CF6E2830DA`

[9] de Almeida Borges, A., Conejero Rodríguez, J. J., Fernández-Duque, D., González Bedmar, M., & Joosten, J. J. (2019). The second order traffic fine: Temporal reasoning in European transport regulations. In J. Gamper, S. Pinchinat, & G. Sciavicco (Eds.) *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*, vol. 147 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (pp. 6:1–6:16). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
URL `http://drops.dagstuhl.de/opus/volltexte/2019/11364`

[10] de Almeida Borges, A., Conejero Rodríguez, J. J., Fernández-Duque, D., González Bedmar, M., & Joosten, J. J. (2021). To drive or not to drive: A logical and computational analysis of European transport regulations. *Information and Computation*, *280*.
URL `https://www.sciencedirect.com/science/article/pii/S0890540120301243`

[4] de Almeida Borges, A. (2022). Towards a Coq formalization of a quantified modal logic. In C. Benzmüller, & J. Otten (Eds.) *Proceedings of the 4th International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2022)*, (pp. 13–27). Haifa, Israel: CEUR.
URL `https://ceur-ws.org/Vol-3326/ARQNL2022_paper1.pdf`

[11] de Almeida Borges, A., González Bedmar, M., Conejero Rodríguez, J., Hermo Reyes, E., Casals Buñuel, J., & Joosten, J. J. (2022). FV Time: A formally verified Coq library. arXiv:2209.14227 [cs.SE].
URL `https://arxiv.org/abs/2209.14227`

Furthermore, the code pertaining to the formalization efforts can be found in the following repositories:

[3] de Almeida Borges, A. (2018). Worms in Coq.
URL `https://gitlab.com/ana-borges/WormsCoq`

[5] de Almeida Borges, A. (2023). Coq formalization of QRC1. Version 1.0.0.
URL `https://gitlab.com/ana-borges/QRC1-Coq/-/releases/v1.0.0`

[7] de Almeida Borges, A., Casals Buñuel, J., Conejero Rodríguez, J., González Bedmar, M., & Hermo Reyes, E. (2023). The FormalV Library. Version 1.2.0.
URL `https://gitlab.com/formalv/formalv/-/releases/1.2.0`

Part I

Provability

2 Background

Contents

2.1 Introduction

In this chapter we present an overview of the state of the art regarding the study of formalized provability in theories of arithmetic through the modal logical lens. It was from the state of affairs described here that we obtained the motivation to study the several topics of Part I. We tried to make this thesis reasonable self-contained, and thus go into some detail on particularly relevant definitions. We also mention many related topics with less detail, which are nevertheless part of the picture.

We start with the arithmetical context and motivation in Sections 2.2 and 2.3, followed by four sections on different kinds of modal logic. In each of these sections, we start by describing the particular kind of logic at hand, highlighting relevant examples, and then dwell on both relational and arithmetical semantics. Section 2.4 describes propositional modal logic, which sets the stage for the remaining kinds of modal logic. Section 2.5 goes into quantified modal logic and Section 2.6, which is independent from the previous one, goes into (propositional) polymodal logic. Finally, Section 2.7 describes strictly positive logic, which is ubiquitous during the rest of Part I.

2.2 Arithmetic

The language of arithmetic $\mathcal{L}_{\text{arith}}$ is that of First-order Logic (FOL) together with the symbols $\{\overline{0}, \overline{1}, +, \times, \exp, <, =\}$ with their usual arities (see [122, 30] for details on this section and the next). The terms of arithmetic are either variables (of which we assume countably-many are available), $\overline{0}$, $\overline{1}$, or built up from other terms using the binary $+$, the binary $\times$, or the unary $\exp$. Terms of the form $\overline{1}+\overline{1}+\cdots+\overline{1}$ are called numerals, and typically abbreviated as $\overline{n}$; for example, $\overline{2}$ is the numeral $\overline{1}+\overline{1}$. When no confusion is possible, we omit the $\bar{\cdot}$. There is a so-called standard model for arithmetic, $\mathbb{N} = \{0, 1, 2, \ldots\}$, where $\overline{0}$ is interpreted as 0, $\overline{1}$ as 1, $+$ as addition, $\times$ as multiplication, $\exp$ as base-2 exponentiation, $<$ as less-than, and $=$ as equality. Theories of arithmetic for which the standard model is a model are said to be sound. There are also other models of arithmetic not isomorphic to $\mathbb{N}$, which are said to be non-standard.

We use standard notation for the language of FOL, with $\neg$ representing negation, $\wedge$ representing conjunction, $\vee$ representing disjunction, $\forall$ representing universal quantification, $\exists$ representing existential quantification, $\to$ representing implication, and $\leftrightarrow$ representing equivalence. Quantification typically binds stronger than implication, so $\forall\, x\, \varphi \to \psi$ is read as $(\forall\, x\, \varphi) \to \psi$.[1]

The notation $\forall\, x < t\, \varphi$ is shorthand for $\forall\, x\, (x < t \to \varphi)$ and $\exists\, x < t\, \varphi$ is shorthand for $\exists\, x\, (x < t \wedge \varphi)$, where x is a variable, t is a term, and φ is a formula in the language of arithmetic. When x does not appear in t we say that these quantifiers are bounded. Formulas that have only bounded quantifiers (and in particular quantifier-free formulas) are said to be elementary. Nested non-bounded quantifiers lead to a measure of formula complexity and to the arithmetical hierarchy.

Definition 2.2.1 (Arithmetical hierarchy, $\Delta^0_0, \Pi^0_n, \Sigma^0_n$)**.** A formula is said to be Δ^0_0 if it is elementary. Then the classes of formulas Σ^0_n and Π^0_n are defined as the smallest classes such that:

[1] In Part III this is sometimes not the case, because Coq uses the opposite convention. We use a comma after the quantifier to distinguish that case, so $\forall x, \varphi \to \psi$ is read as $\forall\, x\, (\varphi \to \psi)$.

- $\Sigma^0_0 := \Pi^0_0 := \Delta^0_0$;
- if $\varphi \in \Sigma^0_n$ then $\forall x\, \varphi \in \Pi^0_{n+1}$;
- if $\varphi \in \Pi^0_n$ then $\exists x\, \varphi \in \Sigma^0_{n+1}$.

We say that a formula φ is Π^0_n in T (respectively, Σ^0_n in T) if there is a formula $\psi \in \Pi^0_n$ (respectively, $\psi \in \Sigma^0_n$) such that $T \vdash \varphi \leftrightarrow \psi$, where T is a theory of arithmetic. A formula is Δ^0_n in T if it is both Σ^0_n in T and Π^0_n in T. Usually T is omitted, because in sufficiently strong classical theories such as the ones we consider here all formulas have a fixed minimum complexity. It is only when dealing with a weaker setting, such as intuitionistic logic or very weak classical theories of arithmetic, that this distinction is relevant. Via coding of pairs of variables, which is available in every theory of arithmetic considered here, we can see that $\forall x\, \forall y\, \varphi$ has the same complexity as $\forall y\, \varphi$, and similarly for consecutive existential quantifiers.

The arithmetical hierarchy is a useful measure of complexity. The decidable formulas are precisely the Δ^0_0 ones, while Σ^0_1 represents the semi-decidable[2] ones. On the other hand, genuine Π^0_1 formulas (for example, Π^0_1 formulas which are not simultaneously Δ^0_0) already represent undecidable sets. A set Γ is said to be arithmetical if there is some formula $\gamma(x)$ in the language of arithmetic such that $\mathbb{N} \vDash \gamma(n)$ if and only if $n \in \Gamma$.

The weakest theory of arithmetic we consider here is Elementary Arithmetic (EA). It includes axioms for First-order Logic, equality, and arithmetical symbols ($\overline{0}$, $\overline{1}$, $+$, $\times$, $\exp$ and $<$). It also includes an induction axiom schema restricted to elementary formulas, which justifies its name.

Definition 2.2.2 (Induction axiom schema, **IA**$_\Gamma$)**.** Let Γ be a set of formulas. The induction axiom schema restricted to Γ is:

$$\textbf{IA}_\Gamma := \varphi(0) \wedge \forall n\, (\varphi(n) \to \varphi(n+1)) \to \forall n\, \varphi(n),$$

where $\varphi(n) \in \Gamma$ is a formula with possibly other free variables besides n. The full induction schema **IA** is as above with unrestricted φ.

Stronger theories are easily defined by considering stronger induction schemas. For example, $\mathrm{I}\Sigma_n$ is EA $+$ **IA**$_{\Sigma^0_n}$. Including an unrestricted induction schema leads to Peano Arithmetic (PA).

We briefly mention the collection principle, as it is used in Chapter 5.

Definition 2.2.3 (Collection principle, **COLL**$_\Gamma$)**.** Let Γ be a set of formulas and $\varphi(y,z) \in \Gamma$ be a formula without x or w as a free variable but possibly other free variables not mentioned here. The collection principle restricted to Γ is:

$$\textbf{COLL}_\Gamma := \forall y < x\; \exists z\, \varphi(y,z) \to \exists w\; \forall y < x\; \exists z < w\, \varphi(y,z).$$

The full collection principle **COLL** is as above with unrestricted φ.

[2]Also known as computably enumerable (c.e.) or recursively enumerable (r.e.).

One way to interpret the collection principle is to think of $\varphi(y,z)$ as describing the graph of a (multi-valued) function. Then, $\forall\, y < x\; \exists\, z\, \varphi(y,z)$ can mean "for every input $y < x$ there is (at least) an output z such that $f(y) = z$, where f is the function described by φ". The consequent of the principle states that we can bound the outputs of this function (for inputs smaller than x) by some number w, obtaining a collection of all possible outputs. However intuitive this observation may be, the collection principle is not provable in every theory of arithmetic. In fact, we have the following relationship between induction and collection.

Lemma 2.2.4 ([122]). *In the following, the notation $T \vdash$ **X** where **X** is a collection of formulas is meant to be interpreted as "T proves every instance of **X**."*

- $\mathsf{EA} + \mathbf{IA}_{\Sigma^0_{n+1}} \vdash \mathbf{COLL}_{\Sigma^0_{n+1}}$;
- $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_{n+1}} \vdash \mathbf{IA}_{\Sigma^0_{n}}$.

2.3 Formalized provability

An interesting feature of arithmetic is that sufficiently strong theories such as EA can introspect, i.e., reason about themselves. One important tool for doing so is the coding of arithmetical formulas into numbers, known as Gödel numbering. We do not go into details here (see [92], and [185] for the EA version). Suffice it to say that there is a Gödel number $\ulcorner\varphi\urcorner$ representing each arithmetical formula φ so that for each syntactic operation such as $\psi \wedge \delta$ there is an easily computable arithmetical function such as $\mathsf{conj}(\cdot,\cdot)$ that translates the Gödel numbers of two formulas into the Gödel number of their conjunction (in symbols, $\mathsf{conj}(\ulcorner\psi\urcorner, \ulcorner\delta\urcorner) = \ulcorner\psi \wedge \delta\urcorner$). When $\varphi(x)$ has a free variable x, we write $\ulcorner\varphi(\dot{x})\urcorner$ to represent a pseudo-term[3] for a function that maps a number n to the number $\ulcorner\varphi(\overline{n})\urcorner$. We can code other things, such as sequences of numbers (and thus sequences of formulas). These codes allow us to express statements such as "n is the code of a sequence", "n is the code of an axiom", "n is the code of a Σ^0_m formula", and so on. We can also quantify over formulas by quantifying over numbers that represent formulas.

Thus, with Gödel numbers we can define formulas representing the axiomatizations of simple theories of arithmetic such as EA and its common extensions. When we have a formula $\tau(u)$ with a single free variable u that is true in the standard model if and only if u is the Gödel number of an axiom of T, we say that $\tau(u)$ is an axiomatization of T. Theories with an elementary axiomatization are said to be elementary presented.

We can now define Gödel's provability predicate $\Box_\tau \ulcorner\varphi\urcorner$ for a given axiomatization τ of a theory of arithmetic as $\exists\, p\, \mathsf{Proof}_\tau(p, \ulcorner\varphi\urcorner)$, where:

$$\begin{aligned}\mathsf{Proof}_\tau(p,n) := \mathsf{sequence}(p) \wedge p_{|p|-1} = n \wedge& \\ \forall\, k < |p|\, (\tau(p_k) \vee \exists\, i < k\; \exists\, j < k\, \mathsf{MP}(p_i, p_j, p_k) \vee \exists\, i < k\, \mathsf{Gen}(p_i, p_k)).&\end{aligned}$$

[3]A pseudo-term as defined by Boolos [49] is a formula $\psi(x,y)$ such that $\forall\, x\, \exists!\, y\, \psi(x,y)$ is provable. Pseudo-terms represent functions which can be reasoned about in arithmetic, even if they cannot be denoted by arithmetical terms.

Roughly, $\Box_\tau\ulcorner\varphi\urcorner$ states that there is a Hilbert-style proof of φ in the theory axiomatized by τ, that is, a finite sequence of formulas $\psi_0, \dots, \psi_m$ such that ψ_m is φ and each ψ_k is either an axiom in the sense of τ or follows from previous elements of the sequence through either *modus ponens* or generalization.

Note that, except for τ, all the functions used in the definition of Proof_τ can be naturally defined as elementary formulas in EA, and thus the complexity of Proof_τ is determined by the complexity of τ. The theories of arithmetic we care about all have computably enumerable (i.e., Σ^0_1) axiomatizations, and by Craig's trick [75, 92], these theories have an equivalent elementary axiomatization.

Theories can have many different axiomatizations, but in general we assume an elementary axiomatization Ax_T has been fixed for each theory T considered here. When we do not wish to highlight the chosen axiomatization, we write Proof_T and $\Box_T$ instead of $\mathsf{Proof}_{\mathsf{Ax}_T}$ and $\Box_{\mathsf{Ax}_T}$. Thus, Proof_T is assumed to be an elementary formula, which implies that $\Box_T\ulcorner\varphi\urcorner$ is a Σ^0_1 formula (regardless of the complexity of φ; remember, $\ulcorner\varphi\urcorner$ is just a number). In Chapter 5 we sometimes need to refer to a specific c.e. axiomatization τ of a theory, and in that case Proof_τ may become a Σ^0_1 formula instead of an elementary one. However, $\Box_\tau$ is still Σ^0_1 in that case.

In the remainder of this thesis, we mostly omit Gödel numbers. Thus, we write $\Box_T\varphi$ instead of $\Box_T\ulcorner\varphi\urcorner$, $\Box_T\varphi(\dot{x})$ instead of $\Box_T\ulcorner\varphi(\dot{x})\urcorner$ when we wish to highlight free variables, and we further write $\Diamond_T\varphi$ as shorthand for $\neg\Box_T\neg\varphi$, which represents the consistency of $T + \varphi$ (and similarly when specifying the axiomatization τ).

The provability predicate as defined above behaves like our intuitive notion of provability, namely we have that, if T is a c.e. extension of EA, then φ is provable in T if and only if $\Box_T\varphi$ is satisfied by the standard model:

$$T \vdash \varphi \iff \mathbb{N} \vDash \Box_T\varphi. \tag{2.1}$$

Furthermore, it is possible to formalize well-known results about provability, such as the deduction theorem.

Theorem 2.3.1 (Formalized deduction [92]). *Let T be a c.e.. extension of* EA *and φ be a formula of arithmetic with free variables $\boldsymbol{y}$. Then:*

$$\mathsf{EA} \vdash \forall\,\theta(\boldsymbol{x})\ \forall\,\boldsymbol{x},\boldsymbol{y}\ \left(\Box_{T+\varphi(\dot{\boldsymbol{y}})}\theta(\dot{\boldsymbol{x}}) \leftrightarrow \Box_T(\varphi(\dot{\boldsymbol{y}}) \to \theta(\dot{\boldsymbol{x}}))\right).$$

We now list some other properties of the provability predicate, which make a reappearance in the next section when we talk about provability logic. We start with the Hilbert-Bernays-Löb derivability conditions, whose name comes from the observation that any predicate $\Box_T$ that fulfills them is sufficient to prove Gödel's second incompleteness theorem [120] about a sufficiently strong and consistent theory of arithmetic T.

Lemma 2.3.2 (Hilbert-Bernays-Löb derivability conditions [161, 49, 30]). *Let T be a c.e. extension of* EA *and φ, ψ be arithmetical formulas. Then:*

1. *if $T \vdash \varphi$, then* EA $\vdash \Box_T\varphi$*;*
2. EA $\vdash \Box_T(\varphi \to \psi) \to (\Box_T\varphi \to \Box_T\psi)$*;*

3. EA $\vdash \Box_T\varphi \to \Box_T\Box_T\varphi$.

Above, the formula φ may have free variables, which under the box shall appear dotted (i.e., $\Box_T\varphi(\dot{x})$).

The third derivability condition is a corollary of provable Σ^0_1-completeness.

Lemma 2.3.3 (Provable Σ^0_1-completeness [193, 122])**.** *Let T be a c.e. extension of* EA *and $\sigma(\boldsymbol{x})$ be a Σ^0_1 formula with free variables $\boldsymbol{x} := x_0, \dots x_{n-1}$. Then,* EA $\vdash \sigma(\boldsymbol{x}) \to \Box_T\sigma(\dot{\boldsymbol{x}})$.

Another important result pertaining to the provability predicate is Löb's theorem.

Theorem 2.3.4 (Löb [161])**.** *Let T be a c.e. extension of* EA *and φ be a sentence of arithmetic. Then:*

$$T \vdash \Box_T\varphi \to \varphi \iff T \vdash \varphi.$$

We can easily obtain Gödel's second incompleteness theorem if we take φ to be $\bot$ (or $0 = 1$) in Löb's theorem: $T \vdash \neg\Box_T\bot$ if and only if $T \vdash \bot$, i.e., T proves its own consistency if and only if T is inconsistent.

As we've seen, there are several notable properties of the provability predicate, and so it makes sense to design a system that collects all such properties without needing to focus on the specific implementation of $\Box_T$. The language $\mathcal{L}_\Box$ of propositional modal logic is optimally suited for this purpose.

2.4 Modal logic

The language of propositional modal logic $\mathcal{L}_\Box$ is obtained from the language of propositional logic by extending it with a unary $\Box$ (box) symbol:

$$\mathcal{L}_\Box ::= \bot \mid p \mid \varphi \to \varphi \mid \Box\varphi,$$

where $p \in$ PROP is a propositional variable (PROP being the set of propositional variables), and φ is an $\mathcal{L}_\Box$ formula. We use the typical abbreviations for propositional logic, such as $\neg\varphi := \varphi \to \bot$, and $\Diamond$ (diamond) as the dual modality to $\Box$, i.e., $\Diamond\varphi := \neg\Box\neg\varphi$. The modal symbols bind stronger than the binary propositional symbols, so for example $\Box\varphi \to \psi$ is a different formula from $\Box(\varphi \to \psi)$.

Modal logic is used as a tool in numerous fields, each reading the modalities in different ways (see [129, 106] for a thorough study of modal logic in general, and detailed explanations of the topics mentioned in this section). Typically the $\Box$ has a flavor of necessity and the $\Diamond$ a flavor of possibility. In the provability case, the one used in this part of the thesis, $\Box\varphi$ reads as "φ is provable in T" and $\Diamond\varphi$ as "φ is consistent with T", where T is a theory of arithmetic such as the ones described in Section 2.2. In Linear Temporal Logic, which we use in Part II, $\Box\varphi$ means "φ holds forever" and $\Diamond\varphi$ means "φ holds at some future moment."

The following modal statements have a relevant interpretation under the provability lens:

- $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$ (**K** or distribution);
- $\Box\varphi \to \Box\Box\varphi$ (**4** or transitivity);

- $\Box(\Box\varphi \to \varphi) \to \Box\varphi$ (**Löb**);

When read through a provability lens, **K** states that if an implication is provable and its antecedent is provable, then so is the consequent. In other words, distribution is formalized *modus ponens*, one of the Hilbert-Bernays-Löb derivability conditions. In the same vein, **4** states that if a formula is provable then this fact is also provable, and it is another derivability condition. Finally, **Löb** is the formalized Löb's theorem.

We briefly mention some other modal statements that are relevant in the modal logic literature but not directly connected to provability:

- $\Box\varphi \to \varphi$ (**T** or reflection);
- $\Diamond\varphi \to \Box\Diamond\varphi$ (**5** or Euclidity);
- $\Box(\varphi \wedge \Box\varphi \to \psi) \vee \Box(\psi \wedge \Box\psi \to \varphi)$ (**.3** or connectedness).

Note that **T** clashes with the provability reading, since under that reading it states that everything that is provable is true. However, it holds under the truth interpretation, i.e. when $\Box\varphi$ is read as "φ is true in the standard model of arithmetic".

There are many modal logics built by taking some specific modal logic as a base and adding one or more axiom schemas to it. The typical base modal logic, called K, is also the minimal *normal* modal logic.

Definition 2.4.1 (Normal modal logic, K)**.** A logic L in the language $\mathcal{L}_\Box$ is called normal if:

- L proves all classical propositional tautologies in $\mathcal{L}_\Box$ (for example, $L \vdash \Box p \vee \neg\Box p$);
- L proves all instances of **K**;
- L is closed under *modus ponens* (if $L \vdash \varphi \to \psi$ and $L \vdash \varphi$ then $L \vdash \psi$);
- L is closed under necessitation (if $L \vdash \varphi$ then $L \vdash \Box\varphi$);
- L is closed under substitution (for example, if $L \vdash p \to \Box p$ then $L \vdash \varphi \to \Box\varphi$ for any formula φ).

K is defined as the minimal normal modal logic.

We can now define a number of well-known normal modal logics, via extending a given normal modal logic with one or more axiom schemas:

- K4 := K + **4**;
- K4.3 := K4 + **.3**;
- GL := K + **Löb**;
- GL.3 := GL + **.3**;
- T := K + **T**;
- S4 := T + **4**;
- S4.3 := S4 + **.3**;
- K5 := K + **5**;
- S5 := T + **5**.

Although we don't directly use or study any of the above logics in this thesis, the work here is very much inspired by the previous study of GL and related logics, so we say some more words about it (see also [49]). The name GL stands for Gödel–Löb, alluding to the status of GL as a provability logic, about which we go into more detail on Section 2.4.2. The logic GL is sometimes defined as $\mathsf{K4} + \textbf{Löb}$, but the definitions are equivalent, since we can derive all instances of **4** in $\mathsf{K} + \textbf{Löb}$.

There are also some important non-normal modal logics, particularly GLS, whose axioms are all the theorems of GL and the axiom schema **T**, with *modus ponens* as the single rule (so it is not closed under necessitation). The theorems of GLS are precisely the always-true modal sentences, as we'll see in Section 2.4.2.

All of the above modal logics (GLS included, see Boolos [49]) are decidable [106, 205]. We go into detail on their complexity in Table 2.1 on page 33, where we also list some of their strictly positive (and less complex) fragments.

2.4.1 Relational semantics

The typical semantics used for modal logics is relational or Kripke semantics. We briefly describe it in this section; refer to Hughes and Cresswell [129] for details.

Definition 2.4.2 (Kripke frame, Kripke model)**.** A Kripke frame is a tuple $\langle W, R\rangle$ where W is a non-empty set of so-called worlds and $R \subseteq W \times W$ is a binary relation on worlds, which we think of as accessibility (we write wRu instead of $\langle w, u\rangle \in R$). In other words, a Kripke frame is simply a directed graph.

A Kripke model is a Kripke frame together with a valuation function $V : \textsc{Prop} \to \wp(W)$, determining which propositional symbols hold at which worlds.

Definition 2.4.3 (Satisfiability)**.** Given a Kripke model $\mathcal{M} = \langle W, R, V\rangle$, the notion of satisfiability of a formula φ at a world w, written $\mathcal{M}, w \Vdash \varphi$, is defined recursively on the formula as follows:

- $\mathcal{M}, w \nVdash \bot$;
- $\mathcal{M}, w \Vdash p$ if and only if $w \in V(p)$, where $p \in \textsc{Prop}$;
- $\mathcal{M}, w \Vdash \varphi \to \psi$ if and only if either $\mathcal{M}, w \nVdash \varphi$ or $\mathcal{M}, w \Vdash \psi$;
- $\mathcal{M}, w \Vdash \Box\varphi$ if and only if for every u such that wRu we have $\mathcal{M}, u \Vdash \varphi$.

If $\mathcal{M}, w \Vdash \varphi$ for every world $w \in W$, we say that φ is valid in $\mathcal{M}$ and write $\mathcal{M} \vDash \varphi$. If $\langle \mathcal{F}, V\rangle \vDash \varphi$ for every valuation V, we say that φ is valid in $\mathcal{F}$ and write $\mathcal{F} \vDash \varphi$.

Since $\Diamond\varphi$ is just a notation for $\neg\Box\neg\varphi$, we can easily check that $\mathcal{M}, w \Vdash \Diamond\varphi$ if and only if there is a world $u \in W$ such that wRu and $\mathcal{M}, w \Vdash \varphi$. Thus, this notion of semantics reads $\Box\varphi$ as "φ holds everywhere accessible from here" and $\Diamond\varphi$ as "there is an accessible world where φ holds".

K is sound and complete with respect to Kripke semantics.

Lemma 2.4.4. *Let φ be a modal formula. Then, $\mathsf{K} \vdash \varphi$ if and only if φ is valid in every Kripke model.*

The simple notion of Kripke model above does not suffice for other modal logics such as K4 though, since we can easily build a model not satisfying instances of **4** such as $\Box p \to \Box\Box p$, as described in Figure 2.1.

$$\begin{tikzcd} a : \{\} \arrow[r] & b : \{p\} \arrow[r] & c : \{\} \end{tikzcd}$$

Figure 2.1: A graphical representation of a Kripke model where $\Box p \to \Box\Box p$ does not hold. Each node includes a label (a, b, or c) and the set of propositional variables that hold at that world. An arrow between two worlds means that the second is accessible from the first. We see that $a \Vdash \Box p$ because b is the only world accessible from a, but $a \nVdash \Box\Box p$ because c, where p does not hold, is accessible in two steps.

However, every model based on a transitive frame satisfies all instances of **4**, and every formula not provable in K4 has a transitive counter-model. For this reason, we say that transitivity is a frame condition for K4 (which justifies referring to **4** as the transitivity axiom schema) and that K4 is frame complete.

Many modal logics have well-known frame conditions, which we summarize as follows (see [106]):

- K4 is sound and complete for transitive (if wRu and uRv then wRv) frames;
- K4.3 is sound and complete for transitive and weakly connected (if wRu and wRv, then either uRv, vRu, or $u = v$) frames;
- GL is sound and complete for transitive and Noetherian (R^{-1} is well-founded, i.e. there are no infinite paths $w_0 R w_1 R w_2 R \cdots$) frames;
- GL.3 is sound and complete for transitive, Noetherian, and weakly connected frames;
- T is sound and complete for reflexive (wRw for all $w \in W$) frames;
- S4 is sound and complete for reflexive and transitive frames (i.e., when R is a preorder);
- S4.3 is sound and complete for reflexive, transitive, and weakly connected frames;
- K5 is sound and complete for Euclidean (if wRu and wRv then uRv) frames;
- S5 is sound and complete for reflexive, transitive, and symmetric (wRu implies uRw) frames (i.e., when R is an equivalence relation).

Note that we can still build non-transitive models satisfying all instances of **4**, and analogously for the other formulas and frame conditions. However, frame completeness is a useful property for a modal logic to have.

2.4.2 Arithmetical semantics

We've been informally talking about interpreting the modal symbols in a provability sense. Here we make that more precise, by introducing the notion of arithmetical realization. The idea is to translate formulas in the modal language into formulas in the arithmetical language. Each different realization may interpret propositional symbols as different arithmetical sentences, but Boolean connectives in the modal language are always mapped to their arithmetical counterparts, and $\Box$ is always mapped to the provability predicate of a chosen arithmetical theory.

Definition 2.4.5 (Arithmetical realization for $\mathcal{L}_\Box$, $\cdot^*$, $\cdot^{*_T}$)**.** An arithmetical realization $\cdot^*$ is a function from Prop to sentences in the language of arithmetic.

Given a c.e. theory T extending EA, and an arithmetical realization $\cdot^*$, we define $\cdot^{*_T}$ on modal formulas as follows:

- $\bot^{*_T} := (0 = 1)$;
- $p^{*_T} := p^*$, where $p \in$ Prop;
- $(\varphi \to \psi)^{*_T} := \varphi^{*_T} \to \psi^{*_T}$;
- $(\Box\varphi)^{*_T} := \Box_T \varphi^{*_T}$.

We can now discuss provable structural properties, i.e., the provable properties that do not depend on the chosen realization. For example, we have EA $\vdash (\Box(p \to q) \to (\Box p \to \Box q))^{*_{\mathsf{EA}}}$ for any realization $\cdot^*$, because EA $\vdash \Box_{\mathsf{EA}}(p^* \to q^*) \to (\Box_{\mathsf{EA}} p^* \to \Box_{\mathsf{EA}} q^*)$ for any arithmetical formulas p^* and q^* (Lemma 2.3.2). The collection of these properties for a given theory T is called the provability logic of T.

Definition 2.4.6 (Provability logic, PL)**.** If T is a c.e. extension of EA, the provability logic of T is defined as:

$$\mathsf{PL}(T) := \{\varphi \in \mathcal{L}_\Box \mid \text{for any } \cdot^*, \text{ we have } T \vdash \varphi^{*_T}\}.$$

Via (2.1) we know that ($T \vdash \varphi^{*_T}$ for any $\cdot^*$) if and only if ($\mathbb{N} \vDash \Box_T \varphi^{*_T}$ for any $\cdot^*$) for c.e. theories T extending EA. Clearly, φ^{*_T} only depends on the value of $\cdot^*$ for the finitely many propositional variables that occur in φ, and so the universal quantifier "for any $\cdot^*$" can be coded and made internal, making it possible to characterize PL(T) by $\{\varphi \in \mathcal{L}_\Box \mid \mathbb{N} \vDash \forall \cdot^* \, \Box_T \varphi^{*_T}\}$. Since $\Box_T \psi^{*_T}$ is a Σ^0_1 formula, this means that PL(T) has a Π^0_2 definition.

Interestingly, PL(PA) has a much simpler, decidable definition: the theorems of GL. This result is due to Solovay and justifies labeling GL as a provability logic. It was shown later by de Jongh *et al.* [79] that this also holds for theories as weak as EA.

Theorem 2.4.7 (Solovay [194])**.** *Let T be a c.e. Σ^0_1-sound*[4] *extension of* EA *and $\varphi \in \mathcal{L}_\Box$. Then:*

$$\mathsf{GL} \vdash \varphi \iff \textit{for any } \cdot^{*_T}, \textit{ we have } T \vdash \varphi^{*_T}.$$

In other words, GL $=$ PL(T).

We can define the set of true structural principles analogously to how we defined the set of structural provable principles.

[4]A theory is Γ-sound if all formulas in Γ provable in the theory are valid in the standard model $\mathbb{N}$.

Definition 2.4.8 (Truth logic, TPL)**.** If T is a c.e. extension of EA, the truth logic of T is defined as:

$$\mathsf{TPL}(T) := \{\varphi \in \mathcal{L}_\Box \mid \text{for any } \cdot^*, \text{ we have } \mathbb{N} \vDash \varphi^{*_T}\}.$$

A priori, $\mathsf{TPL}(\mathsf{PA})$ falls outside the arithmetical hierarchy due to Tarski's [200] result on the undefinability of arithmetical truth. However, Solovay also simplified this logic by showing that $\mathsf{TPL}(\mathsf{PA})$ is described by the theorems of GLS, and consequently decidable.

Theorem 2.4.9 (Solovay [194])**.** *Let T be a c.e. Σ^0_1-sound extension of EA and $\varphi \in \mathcal{L}_\Box$. Then:*

$$\mathsf{GLS} \vdash \varphi \iff \textit{for any } \cdot^{*_T}\textit{, we have } \mathbb{N} \vDash \varphi^{*_T}.$$

In other words, $\mathsf{GLS} = \mathsf{TPL}(T)$*.*

2.5 Quantified modal logic

The language of modal logic can be extended to a first-order setting in the natural way. We describe a simple version with no equality nor function symbols, $\mathcal{L}_{\Box\forall}$. Fix a signature Σ, which in this case is simply a set of relation symbols Rel_Σ and their respective arity (we sometimes omit specific mention of Σ). Then:

$$\mathcal{L}_{\Box\forall} ::= \bot \mid S(\boldsymbol{x}) \mid \varphi \to \varphi \mid \forall\, x\, \varphi \mid \Box\varphi,$$

where φ represents an $\mathcal{L}_{\Box\forall}$-formula, $S \in \mathsf{Rel}_\Sigma$ is a relation symbol with arity n, $\boldsymbol{x}$ is an n-tuple of variables, and x is a variable. We use the typical notations, including $\exists\, x\, \varphi := \neg \forall\, x\, \neg\varphi$. The quantifiers bind stronger than the binary propositional symbols, so for example $\forall\, x\, \varphi \to \psi$ is a different formula from $\forall\, x\, (\varphi \to \psi)$.

The free variables of a formula φ are defined as usual, and denoted by $\mathsf{fv}(\varphi)$. The expression $\varphi[x{\leftarrow}y]$ denotes the formula φ with all free occurrences of the variable x simultaneously replaced by the variable y. We say that y is free for x in φ if no occurrence of y becomes bound in $\varphi[x{\leftarrow}y]$.

Given a propositional modal logic L, we define its quantified version $\mathsf{Q}L$ as described by Hughes and Cresswell [129]:

- $\mathsf{Q}L$ proves all substitution instances of theorems of L in the language $\mathcal{L}_{\Box\forall}$;
- $\mathsf{Q}L$ proves all instances of $\forall\, x\, \varphi \to \varphi[x{\leftarrow}y]$, with y free for x in φ ($\forall$**E**);
- if $\mathsf{Q}L \vdash \varphi \to \psi$, then $\mathsf{Q}L \vdash \varphi \to \forall\, x\, \psi$, with $x \notin \mathsf{fv}(\varphi)$ ($\forall$**I**);
- $\mathsf{Q}L$ is closed under *modus ponens*;
- $\mathsf{Q}L$ is closed under necessitation.

This leads to the straightforward definition of logics such as $\mathsf{QK4}$ and QGL.

2.5.1 Relational semantics

There have been several proposals for relational semantics for quantified modal logics, from Kripke [145] to many others. Overviews can be found in [129, 107, 110]. The idea of the Kripke models for propositional modal logic described in Section 2.4.1 is to glue together several propositional models (worlds) through an accessibility relation R. Similarly, the idea with quantified modal logic is to glue together several first-order models. However, the precise way in which to do it can vary, due to different ways to interpret the relationship between quantifiers and modal symbols, as well as with the first-order domains. A simple choice is to fix a domain for the full model, leading to so-called constant domain models. An alternative is to allow each world to have a specific domain, and possibly set some restrictions on the relationships between these domains. Different quantified modal logics are sound and complete for different choices of domain restrictions.

Consider the formula $\forall x\, \Diamond P(x)$. When interpreted at a world w with naive Kripke semantics, it reads "for every domain element x, there is an accessible world u where $P(x)$ holds." However, assuming this is not a constant domain model, the immediate question is: which domain does the quantification "for every domain element" range over?

In the possibilist reading of quantifiers, it is stipulated that a quantification always ranges over the full domain of the model (i.e., the union of the domains of the individual worlds). We do not use this interpretation here; see [129] for details. The alternative, called actualist interpretation, is to say that a quantifier only ranges over the domain of the world in which it is being interpreted. But then a problem still needs to be solved: what should the interpretation of $P(x)$ at world u be when x is taken to represent a domain element that is not part of the domain of u? One can assign truth values to the interpretation of $P(x)$ when x represents a non-existing domain element, forbid such cases, or even say that there is no truth-value at all, i.e., that the statement is undefined. An exploration of the various different ways around this issue can be found in [129]. Regardless, these kinds of models for QK and its extensions require inclusive (also known as increasing or expanding) domains, i.e., domains where wRu implies that the domain of w is a subset of the domain of u.

In this thesis, we use a different, although related, notion of relational semantics, called Kripke sheaf [107].[5] The idea is that between any two worlds w and u such that wRu, there is a compatibility function $\eta_{w,u}$ translating the domain elements of w into domain elements of u. This ensures that we always know what a domain element in one world corresponds to in an accessible world. As we see below, this side-steps the inclusiveness requirement, although the essence of the idea of inclusiveness remains.

Definition 2.5.1 (Kripke sheaf)**.** A Kripke sheaf $\mathcal{S}$ is a tuple $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}\rangle$ where:

- W is a non-empty set (the set of worlds, where individual worlds are referred to as w, u, v, etc);
- R is a binary relation on W (the accessibility relation);

[5]Even though a sheaf is a different mathematical concept (which inspired the notion of Kripke sheaf), we sometimes omit the "Kripke" prefix in this thesis, where non-Kripke sheaves are never considered.

- each M_w is a non-empty set (the domain of the world w, whose elements are referred to as d, d_0, d_1, etc);
- if wRu, then $\eta_{w,u}$ is a function from M_w to M_u.

Furthermore, we require two conditions on the compatibility functions:

- if wRu, uRv, and wRv, then for every $d \in M_w$ we have $\eta_{u,v}(\eta_{w,u}(d)) = \eta_{w,v}(d)$;
- if wRw then $\eta_{w,w}$ is the identity function on M_w.

We say that a sheaf is finite if both W and the domains M_w for every $w \in W$ are finite, and that it is constant domain if all the M_w coincide and all the $\eta_{w,u}$ are the identity function.

The definition of sheaf is not that different from the definition of frame in a first-order context, and in particular a sheaf is trivially reduced to a frame when constant domains are considered.

Definition 2.5.2 (First-order Kripke model). Given a signature Σ, a first-order Kripke model $\mathcal{M}$ is a tuple $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}, \{J_w\}_{w\in W}\rangle$ where $\mathcal{S} := \langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}\rangle$ is a Kripke sheaf and for each $w \in W$ the interpretation J_w assigns a set of tuples $S^{J_w} \subseteq \wp((M_w)^n)$ to each n-ary relation symbol $S \in \textsc{Rel}_\Sigma$. In this case, we say that the model $\mathcal{M}$ is based on the sheaf $\mathcal{S}$.

In order to define the notion of satisfiability, we need one more ingredient, namely a way to interpret free variables. For this, we use assignments. Given a world w, a w-assignment g is a function from the set of variables to the domain M_w. Any w-assignment can be seen as a u-assignment as long as wRu, by composing it with $\eta_{w,u}$ on the left. We write g^u to shorten $\eta_{w,u} \circ g$ when w is clear from context.

Two w-assignments g and h are Γ-alternative, written $g \sim_\Gamma h$, if they coincide on all variables other than the ones in Γ. When $\Gamma = \{x\}$, we write x-alternative and $g \sim_x h$.

We are finally ready to define the notion of satisfiability.

Definition 2.5.3 (Satisfiability). Let $\mathcal{M} = \langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}, \{J_w\}_{w\in W}\rangle$ be a Kripke model in some signature Σ, and let $w \in W$ be a world, g be a w-assignment, $S \in \textsc{Rel}_\Sigma$ be an n-ary relation symbol, and φ, ψ be $\mathcal{L}_{\Box\forall}$ formulas in the language of Σ.

We define $\mathcal{M}, w \Vdash^g \varphi$ (φ is true at w under g) recursively on φ as follows:

- $\mathcal{M}, w \nVdash^g \bot$;
- $\mathcal{M}, w \Vdash^g S(x_0, \dots, x_{n-1})$ if and only if $\langle g(x_0), \dots, g(x_{n-1})\rangle \in S^{J_w}$;
- $\mathcal{M}, w \Vdash^g \varphi \to \psi$ if and only if either $\mathcal{M}, w \nVdash^g \varphi$ or $\mathcal{M}, w \Vdash^g \psi$;
- $\mathcal{M}, w \Vdash^g \forall x\, \varphi$ if and only if for all w-assignments h such that $h \sim_x g$ we have $\mathcal{M}, w \Vdash^h \varphi$;
- $\mathcal{M}, w \Vdash^g \Box\varphi$ if and only if for every $u \in W$ such that wRu we have $\mathcal{M}, u \Vdash^{g^u} \varphi$.

If $\mathcal{M}, w \Vdash^g \varphi$ for every w-assignment g we say that φ is satisfied at w and write $\mathcal{M}, w \Vdash \varphi$. If φ is satisfied at every world w, we say that φ is valid in $\mathcal{M}$ and write $\mathcal{M} \vDash \varphi$. Finally, if φ is valid in every model $\mathcal{M}$ based on a given sheaf $\mathcal{S}$, we say that φ is valid in $\mathcal{S}$ and write $\mathcal{S} \vDash \varphi$.

Note that when interpreting $\forall x\, \varphi$ in the world w we quantify over w-assignments only, which is to say, we interpret the quantification as ranging over the domain of w and nothing else. Note also that $\mathcal{M}, w \Vdash^g \varphi$ is only defined when g is a w-assignment.

We now sketch the proof of a simple soundness result, which will be useful in Section 4.5. We go into much more detail on the soundness proof described in Section 4.3, which shares many of the same ideas, even if it is about a logic in a different language.

Theorem 2.5.4 (Soundness for $\mathrm{Q}L$). *Let $\mathcal{F}$ be a Kripke frame for $\mathcal{L}_\Box$ as described in Definition 2.4.2. If $\mathcal{F}$ satisfies every theorem of a propositional modal logic L, then any sheaf based on $\mathcal{F}$ (i.e., with the same set of worlds and accessibility relation) satisfies every theorem of $\mathrm{Q}L$.*

Proof. We proceed by induction on a $\mathrm{Q}L$ proof of a $\mathcal{L}_{\Box\forall}$ formula.

Given a substitution instance of a theorem of L, i.e., a formula $\varphi[p_0{\leftarrow}\delta_0]\cdots[p_{n-1}{\leftarrow}\delta_{n-1}]$ (which we abbreviate as $\varphi[\boldsymbol{p}{\leftarrow}\boldsymbol{\delta}]$) where $\delta_i \in \mathcal{L}_{\Box\forall}$ for each $i < n$ and $L \vdash \varphi$ (note that $\varphi \in \mathcal{L}_\Box$ and $\varphi[\boldsymbol{p}{\leftarrow}\boldsymbol{\delta}] \in \mathcal{L}_{\Box\forall}$), we reason by contradiction. Let $\mathcal{M}_1$ be a first-order Kripke model based on any sheaf based on $\mathcal{F}$, w be a world in that model, and g be a w-assignment such that $\mathcal{M}_1, w \nVdash^g \varphi[\boldsymbol{p}{\leftarrow}\boldsymbol{\delta}]$.

We now define a propositional Kripke model $\mathcal{M}_0$ based on $\mathcal{F}$ such that, if u is either w or accessible from w, we have $\mathcal{M}_0, u \Vdash p_i$ if and only if $\mathcal{M}_1, u \Vdash^{\eta_{w,u}\circ g} \delta_i$ (with $\eta_{w,w}$ as the identity function for the purposes of easing this definition), for each $i < n$. We do not care about the valuation of the symbols p_i at other worlds nor about the valuation of any other propositional symbols, so we can assume they are not validated anywhere else. It is easy to check by induction on φ that $\mathcal{M}_1, w \Vdash^g \varphi[\boldsymbol{p}{\leftarrow}\boldsymbol{\delta}]$ if and only if $\mathcal{M}_0, w \Vdash \varphi$ (see Lemma 4.3.9 for a proof of a different result following the same idea). This is a contradiction, because φ is valid in $\mathcal{M}_0$ by assumption (it is a model based on $\mathcal{F}$).

We turn to $\forall$**E**, where we need to show that if $\mathcal{M}, w \Vdash^g \forall x\, \varphi$, then $\mathcal{M}, w \Vdash^g \varphi[x{\leftarrow}y]$, with y free for x in φ and arbitrary $\mathcal{M}$, w, and g. Let $h \sim_x g$ be a w-assignment such that $h(x) = g(y)$. By assumption, $\mathcal{M}, w \Vdash^h \varphi$. This is equivalent to $\mathcal{M}, w \Vdash^g \varphi[x{\leftarrow}y]$ via a standard First-order Logic result (see Lemma 4.3.4 for a proof of this result in a slightly different setting).

For $\forall$**I**, assume that $\mathrm{Q}L \vdash \varphi \to \psi$ and that x is not free in φ. We wish to show that if $\mathcal{M}, w \Vdash^g \varphi$ then $\mathcal{M}, w \Vdash^g \forall x\, \psi$ for arbitrary $\mathcal{M}$, w, and g. Let $h \sim_x g$ be an arbitrary w-assignment. By the induction hypothesis with h, we have $\mathcal{M}, w \Vdash^h \varphi \to \psi$. Since x is not free in φ and $\mathcal{M}, w \Vdash^g \varphi$ by assumption, we also know that $\mathcal{M}, w \Vdash^h \varphi$ by another well-known First-order Logic result (see Lemma 4.3.3). Then we may conclude $\mathcal{M}, w \Vdash^h \psi$ from the stated induction hypothesis, as desired.

It is clear from the definition of satisfiability that *modus ponens* is sound.

Finally, for necessitation, we assume $\mathrm{Q}L \vdash \varphi$ with $\varphi \in \mathcal{L}_{\Box\forall}$ and wish to show that for any first-order Kripke model $\mathcal{M}$ based on any sheaf based on $\mathcal{F}$, any world w, and any w-assignment g we have $\mathcal{M}, w \Vdash^g \Box\varphi$. Let u be any world such that wRu; we wish to show $\mathcal{M}, u \Vdash^{g^u} \varphi$. This is straightforward from the induction hypothesis applied to $\mathrm{Q}L \vdash \varphi$ with the world u and assignment g^u. $\Box$

From this soundness result we conclude for example that any first-order Kripke model based on a sheaf is a QK model, any such transitive model is a QK4 model, and any such transitive and

Noetherian model is a QGL model. Many more similar corollaries can be obtained from the list of frame conditions described in Section 2.4.1.

So far we have not mentioned any formulas with both quantifiers and modalities, nor any logics axiomatized by such formulas. In fact, these formulas do not play a large role in this thesis. However, we cannot end this section without mentioning the well-known Barcan formula (or, more accurately, Barcan schema):

$$\textbf{BF} := \forall x\, \Box\varphi \to \Box \forall x\, \varphi.$$

This schema was first isolated by Barcan [26] when she described one of the first systems of quantified modal logic.

The Barcan formula is not derivable in QK, nor in any of the other logics defined above. In fact, it is easy to find counter-models for **BF** if one picks non-surjective compatibility functions. Conversely, Kripke sheaves with surjective compatibility functions satisfy all instances of **BF**. When using Kripke frame instead of Kripke sheaf semantics, the Barcan formula is associated with constant domains [129].

BF is not valid in a provability setting: when we interpret the $\Box$ as, say, "provable in PA", it states that external universal quantifiers can be made internal, which is not the case. On the other hand, the converse of the Barcan formula is already provable in QK, and easily seen to be arithmetically sound.

2.5.2 Arithmetical semantics

We extend the notion of arithmetical realization to $\mathcal{L}_{\Box\forall}$ in the natural way. To simplify matters, in the sections dealing with both the quantified modal language and the language of arithmetic, we always use variables named x_i (for some i) in the modal setting and variables named y_i in the arithmetical setting.

Definition 2.5.5 (Arithmetical realization for $\mathcal{L}_{\Box\forall}$, $\cdot^*$, $\cdot^{*_T}$)**.** An arithmetical realization $\cdot^*$ is a function from relation symbols applied to a tuple of n variables (corresponding to the symbol's arity) to formulas in the language of arithmetic with n free variables. We require that a variable x_i in the modal language be translated to y_i in the arithmetical language.

Given a c.e. theory T extending EA, and an arithmetical realization $\cdot^*$, we define $\cdot^{*_T}$ on quantified modal formulas as follows:

- $\bot^{*_T} := (0 = 1);$
- $S(\boldsymbol{x})^{*_T} := S(\boldsymbol{x})^*;$
- $(\varphi \to \psi)^{*_T} := \varphi^{*_T} \to \psi^{*_T};$
- $(\forall x_k\, \varphi)^{*_T} := \forall y_k\, \varphi^{*_T};$
- $(\Box\varphi)^{*_T} := \Box_T \varphi^{*_T}.$

After Solovay's completeness theorems, it was natural to ask whether one could find nice characterizations for the counterparts of PL(PA) and TPL(PA) in the quantified modal language, defined as follows (see Boolos [49]).

Definition 2.5.6 (Quantified provability logic, QPL and quantified truth logic, QTPL)**.** If T is a c.e. extension of EA, the quantified provability logic of T is defined as:[6]

$$\mathsf{QPL}(T) := \{\varphi \in \mathcal{L}_{\Box\forall} \mid \text{for any } \cdot^*, \text{ we have } T \vdash \forall \boldsymbol{y}\, \varphi^{*_T}\}.$$

In general, $\mathsf{QPL}(T)$ can be different depending on the chosen axiomatization τ for T (see [21, 146, 147]), but here we fix an axiomatization for each relevant theory T for simplicity.

The quantified truth logic of T is defined as:

$$\mathsf{QTPL}(T) := \{\varphi \in \mathcal{L}_{\Box\forall} \mid \text{for any } \cdot^*, \text{ we have } \mathbb{N} \vDash \forall \boldsymbol{y}\, \varphi^{*_T}\}.$$

The natural candidate for a quantified provability logic for PA was QGL, but although QGL $\subseteq$ QPL(PA), Montagna [175] showed that this inclusion is strict, i.e., that QGL is not complete for arithmetical semantics. Soon after, Artemov [20] proved that QTPL(PA) is not even arithmetically definable. Last hopes were scattered when Vardanyan [203] and McGee [170] independently showed that QPL(PA) is as complex as it can possibly be: Π^0_2-complete. This is a very high complexity, and it implies that QPL(PA) does not have a recursive axiomatization. We refer to this result as Vardanyan's theorem. Moreover, Boolos and McGee [50] showed that QTPL(PA) is also as complex as possible: Π^0_1-complete in the set of true sentences of arithmetic.

These negative results are quite robust in various ways. For example, Vardanyan [204] showed that restricting the language to the fragment with a single unary predicate and without any modality nesting doesn't break the Π^0_2-completeness. Restricting the complexity of the realizations to Σ^0_1 formulas doesn't help either: the corresponding set is still Π^0_2-complete, as shown by Berarducci [44].

One may ask whether this is merely a problem for PA or if it holds for other theories of arithmetic too. It is easy to see that the degenerate case of $\mathsf{QPL}(\mathsf{PA} + \Box_{\mathsf{PA}}\bot)$ is axiomatized by FOL together with $\Box\bot$.[7] However, Visser and de Jonge [207] proved that for most natural theories T, $\mathsf{QPL}(T)$ is indeed Π^0_2-complete. The title of their paper is *No escape from Vardanyan's theorem*.

Be that as it may, there have already been some "escapes" from Vardanyan's theorem. For example, the one-variable fragment of QPL(PA) is decidable, as shown by Artemov and Japaridze [23]. Furthermore, an arithmetically complete quantified modal logic was proposed by Yavorsky [212] (see also [124]). This logic, called QGL^b, assumes that in $\Box A$ every free variable of A is bound under the box.

Chapters 4 and 5 were inspired by this state of affairs, and describe an alternative escape from Vardanyan's theorem, obtained by restricting the language to its strictly positive fragment (see Section 2.7 for an introduction to strictly positive logic).

2.6 Polymodal logic

We have already spoken of the standard provability predicate, but there are other relevant notions of provability, such as provability under assumptions. The interplay between several different

[6]Recall that we assume all modal variables are of the form x_i and that these are translated to the arithmetical variables y_i. Thus, with $\forall \boldsymbol{y}\, \varphi^{*_T}$ we mean to quantify over all the free variables of φ^{*_T}.

[7]Given a quantified modal formula φ, let $\widetilde{\varphi}$ be φ with every subformula of the form $\Box\psi$ replaced by $\top$. Let $\cdot^*$ be any realization. Then $\mathsf{PA} + \Box_{\mathsf{PA}}\bot \vdash \varphi^*$ if and only if $\mathsf{PA} \vdash \widetilde{\varphi}^*$ (by induction on φ). Similarly, $\mathsf{FOL} + \Box\bot \vdash \varphi$ if and only if $\mathsf{FOL} \vdash \widetilde{\varphi}$.

provability notions can be represented by polymodal logics, i.e., logics with several different, non inter-definable, modalities. Polymodal logic appears in several contexts, but here we focus on the provability one. We talk about Linear Temporal Logic in Part II, which is a different kind of polymodal logic.

Japaridze [139] generalized the logic GL to a polymodal version GLP with countably many modalities, and Beklemishev [31] generalized this further to a transfinite setting, yielding GLP$_\Lambda$, where for each ordinal $\alpha < \Lambda$ there is a provability modality $[\alpha]$ (and the corresponding $\langle\alpha\rangle$ defined as $\langle\alpha\rangle\varphi := \neg[\alpha]\neg\varphi$), and larger ordinals refer to stronger provability notions. Let $\mathcal{L}_{[\alpha],\alpha<\Lambda}$ be the language of GLP$_\Lambda$.

Definition 2.6.1 (GLP$_\Lambda$ [139, 31])**.** Let φ and ψ be formulas in the language of GLP$_\Lambda$, and $\alpha, \beta < \Lambda$ be ordinals. The axioms and rules of GLP$_\Lambda$ are:

(i) all propositional tautologies in the language $\mathcal{L}_{[\alpha],\alpha<\Lambda}$;

(ii) $[\alpha](\varphi \to \psi) \to ([\alpha]\varphi \to [\alpha]\psi)$ (distribution);

(iii) $[\alpha]([\alpha]\varphi \to \varphi) \to [\alpha]\varphi$ (Löb);

(iv) $\langle\beta\rangle\varphi \to [\alpha]\langle\beta\rangle\varphi$, for $\alpha > \beta$ (negative introspection);

(v) $[\beta]\varphi \to [\alpha]\varphi$, for $\alpha > \beta$ (monotonicity);

(vi) if GLP$_\Lambda \vdash \varphi \to \psi$ and GLP$_\Lambda \vdash \varphi$, then GLP$_\Lambda \vdash \psi$ (*modus ponens*);

(vii) if GLP$_\Lambda \vdash \varphi$ then GLP$_\Lambda \vdash [\alpha]\varphi$ (necessitation).

As we can see, each $[\alpha]$ behaves like the $\Box$ in GL, and there are two axiom schemas describing how $[\alpha]$ and $[\beta]$ interact depending on the order of the ordinals α and β, one of which is reminiscent of **5**.

GLP$_\Theta$ is a conservative extension of GLP$_\Gamma$ whenever $\Theta > \Gamma$ [40]. In other words, introducing higher modalities does not lead to new GLP theorems that do not mention them. Furthermore, in a purely modal context the study of GLP$_\Lambda$ does not differ significantly from the study of GLP$_2$. For these reasons, we omit the subscript Λ whenever it is not relevant to the matter at hand.

2.6.1 Relational semantics

We can extend the notion of Kripke frame to a polymodal logic such as GLP by introducing one relation R_α for each modality $[\alpha]$.

Definition 2.6.2 (Kripke frame and model for $\mathcal{L}_{[\alpha],\alpha<\Lambda}$)**.** A Kripke frame for $\mathcal{L}_{[\alpha],\alpha<\Lambda}$ is a tuple $\langle W, \{R_\alpha\}_{\alpha<\Lambda}\rangle$ where W is a set of worlds and for each $\alpha < \Lambda$, $R_\alpha \subseteq W \times W$ is a binary relation on worlds, which we think of as accessibility via α.

A Kripke model is a Kripke frame together with a valuation function $V : \textsc{Prop} \to \wp(W)$, determining which propositional symbols hold at which worlds.

The notion of satisfiability in the polymodal case is just as in the unimodal one (Definition 2.4.3), with $[\alpha]$ being interpreted as follows:

- $\mathcal{M}, w \Vdash [\alpha]\varphi$ if and only if for every u such that $wR_\alpha u$ we have $\mathcal{M}, u \Vdash \varphi$.

We can easily see that in order for a frame to validate all the theorems of GLP we should have simultaneously that:

- each R_α is transitive and Noetherian, due to Löb's axiom for each α;
- if $wR_\alpha u$ and $wR_\beta v$ with $\beta < \alpha$, then $uR_\beta v$ (poly-Euclidity), due to the negative introspection axiom;
- if $\beta \leq \alpha$, then $R_\alpha \subseteq R_\beta$ (monotonicity), due to the monotonicity axiom.

Unfortunately, these conditions are in conflict (unless there are no relations R_α with $\alpha > 0$). To see this, suppose $\beta < \alpha$ and R_α is non-empty. If $wR_\alpha u$ for some worlds w and u, it follows by monotonicity that $wR_\beta u$, but then by poly-Euclidity it must be that $uR_\beta u$, which implies that R_β is not Noetherian.

For this reason, relational semantics is not much used in the context of GLP, although there are Kripke models for which GLP is sound and complete [33]. Instead, GLP has good (albeit complex) topological semantics [41], which we do not explore here, and several different choices of arithmetical semantics, which we mention in Section 2.6.2.

In contrast, some fragments of GLP do have well-behaved relational semantics, particularly the closed fragment GLP^0, which we discuss presently, and the strictly positive fragment RC, which we discuss in Section 2.7. There is another fragment with good Kripke semantics named J [33]. This is the fragment of GLP without monotonicity but with poly-transitivity, i.e., $[\beta]\varphi \to [\alpha][\beta]\varphi \wedge [\beta][\alpha]\varphi$ with $\beta \leq \alpha$, which is derivable in GLP. It was via J that complete Kripke models for GLP were found, but we do not further comment on J in this thesis.

GLP^0 has a universal Kripke model[8] called Ignatiev's model [135, 96], or $\mathcal{I}$. By universal, we mean that every theorem of GLP^0 is validated in $\mathcal{I}$ and that if a closed formula φ is not a theorem of GLP^0 then there is a world of $\mathcal{I}$ where it is refuted. This model is essentially infinite, having fractal features.

In order to describe $\mathcal{I}$, we start with some auxiliary definitions, and we assume basic knowledge about ordinals (see, for example, [183]). We use On to represent the class of all ordinals.

Definition 2.6.3 (End-logarithm, ℓ)**.** The end-logarithm ℓ is a function from On to On defined as follows:

- $\ell(0) := 0$;
- $\ell(\alpha + \omega^\beta) := \beta$.

We can iterate end-logarithms, obtaining hyper-logarithms. For this definition, we use the notion of initial function. This is a function that maps initial segments of ordinals $[0, \alpha]$ (i.e., intervals including 0) onto initial segments $[0, \beta]$. The end-logarithm is an initial function: finite initial segments $[0, n]$ are mapped to the point $[0]$, while $[0, \omega]$ is the first segment to be mapped to $[0, 1]$, $[0, \omega^2]$ the first to be mapped to $[0, 2]$, and so on.

[8]It is actually a frame, but since we only care about closed formulas, every frame is a model as well.

Definition 2.6.4 (Hyper-logarithms, ℓ^{ξ})**.** The hyper-logarithms are iterates of ℓ. We write ℓ^{ξ} to denote the ξ-th iterate of ℓ, and define:

- $\ell^0 :=$ id (the identity function);
- $\ell^1 := \ell$;
- $\ell^{\alpha+\beta} := \ell^{\beta} \circ \ell^{\alpha}$.

These three properties do not tell us anything about ℓ^{ξ} for an additively indecomposable ξ, so we further require that each ℓ^{ξ} be point-wise maximal among all families of initial ordinal functions $\{f^{\xi}\}_{\xi \in \mathsf{On}}$ that satisfy the three properties. In this way, each ℓ^{ξ} defines an initial function. For the purposes of this thesis, many of the exact details of the ℓ^{ξ} functions are irrelevant and we refer the interested reader to [95, 96] for further details.

We now compute the first few of values of ℓ^{ω} in order to obtain some intuition. In what follows, ε_{ζ} denotes the ζ-th fixpoint of $x \mapsto \omega^x$. Since the initial segment $[0]$ should be mapped onto an initial segment, it must be that $\ell^{\omega}(0) = 0$. If $\alpha < \varepsilon_0$ then it is easy to see that, after some finite number n of iterations, we have $\ell^n(\alpha) = 0$ (for example, $\ell^3(\omega^{\omega^2}) = \ell^2(\omega^2) = \ell(2) = 0$). Thus, if we take such an n for α, we can see that $\ell^{n+\omega}(\alpha) = \ell^{\omega} \circ \ell^n(\alpha) = \ell^{\omega}(0) = 0$, which means that $\ell^{\omega}(\alpha) = 0$ as well (because $\omega = n + \omega$). Consequently, each initial segment $[0, \alpha]$ with $\alpha < \varepsilon_0$ is mapped by ℓ^{ω} to $[0]$.

What about $\ell^{\omega}(\varepsilon_0)$? If we disregard the requirement on initiality, it is not hard to see that $\ell^{\omega}(\varepsilon_0)$ could be any value. However, since ℓ^{ω} should map the initial segment $[0, \varepsilon_0]$ to an initial segment and do so in a maximal way, it must be that $\ell^{\omega}(\varepsilon_0) = 1$.

We observe that $\ell^{\omega}(\xi + \zeta) = \ell^{1+\omega}(\xi + \zeta) = \ell^{\omega} \circ \ell(\xi + \zeta) = \ell^{\omega} \circ \ell(\zeta) = \ell^{1+\omega}(\zeta) = \ell^{\omega}(\zeta)$ so that $\ell^{\omega}(\varepsilon_0 + \alpha) = 0$ for any $\alpha < \varepsilon_0$.

Following these kinds of arguments, we can see that the next value where ℓ^{ω} increases is at ε_1 and $\ell^{\omega}(\varepsilon_1) = 2$. Fortunately, we do not need to reason from scratch any time we want to know some value of $\ell^{\alpha}(\beta)$, as there is a recursive algorithm for the hyper-logarithms.

Lemma 2.6.5 (Fernández-Duque and Joosten [95, 96])**.** *For ordinals ξ, ζ, the following recursion is well-defined*[9] *and determines all the $\ell^{\xi}(\zeta)$ values:*

- $\ell^0(\alpha) = \alpha$*;*
- $\ell^{\xi}(0) = 0$*;*
- $\ell^1(\alpha + \omega^{\beta}) = \beta$*;*
- $\ell^{\omega^{\rho}+\xi} = \ell^{\xi} \circ \ell^{\omega^{\rho}}$ *provided that* $\xi < \omega^{\rho} + \xi$*;*
- $\ell^{\omega^{\rho}}(\zeta) = \ell^{\omega^{\rho}}(\ell^{\eta}(\zeta))$ *if* $\rho > 0$ *and* $\eta < \omega^{\rho}$ *is such that* $\ell^{\eta}(\zeta) < \zeta$ *(if such* η *exists);*
- $\ell^{\omega^{\rho}}(\zeta) = \sup\limits_{\delta \in [0,\zeta)} (\ell^{\omega^{\rho}}(\delta) + 1)$ *if* $\rho > 0$ *and* $\zeta = \ell^{\eta}(\zeta)$ *for all* $\eta < \omega^{\rho}$*.*

[9]This is not obvious, since $\ell^{\omega^{\rho}}(\zeta)$ with $\rho > 0$ is defined as depending on a chosen η in some cases. However, every choice of η that fulfills the restriction leads to the same value of $\ell^{\omega^{\rho}}(\zeta)$.

Now that the hyper-logarithms have been defined, we can specify the points of Ignatiev's model $\mathcal{I}$ which are the so-called ℓ-*sequences*.

Definition 2.6.6 (ℓ-sequence)**.** An ℓ-*sequence* is a function $f : \mathsf{On} \to \mathsf{On}$ such that for each ordinal ζ we have $f(\zeta) \leq \ell^{-\xi+\zeta}(f(\xi))$ for $\xi < \zeta$ large enough.[10]

At times we shall write f_ξ instead of $f(\xi)$. We note that for each ℓ-sequence f the inequality $f(\alpha+1) \leq \ell(f(\alpha))$ holds. Furthermore, the requirement of $\xi < \zeta$ being large enough is important, as it means that $f = \langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots 1, 0, \ldots \rangle$ is an ℓ-sequence, where $f(0) = \omega^{\varepsilon_0+1}$, $f(n) = \varepsilon_0$ for $0 < n < \omega$, $f(\omega) = 1$ and $f(\alpha) = 0$ for $\alpha > \omega$. It is easy to see that $\ell^\omega(\omega^{\varepsilon_0+1}) = 0$ and $\ell^\omega(\varepsilon_0) = 1$. Then, $f(\omega) \leq \ell^\omega(f(1))$ but it is not the case that $f(\omega) \leq \ell^\omega(f(0))$.

We can now define the class-size version of Ignatiev's model as the collection of all ℓ-sequences with suitable relations R_ξ to model each of the $[\xi]$ modalities. For all practical purposes we can take sufficiently large set-size truncations of the class-size model.

Definition 2.6.7 (Ignatiev's model, $\mathcal{I}$ [135, 96])**.** Ignatiev's model $\mathcal{I}$ is a Kripke model $\langle I, \{R_\xi\}_{\xi \in \mathsf{On}} \rangle$, where I is the collection of all ℓ-sequences and $f R_\xi g$ if and only if both $f(\alpha) = g(\alpha)$ for $\alpha < \xi$ and $f(\xi) > g(\xi)$.

For example:

$$\begin{array}{lll}
\langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 0, 0, \ldots \rangle & R_0 & \langle \varepsilon_0, \varepsilon_0, \varepsilon_0, \ldots, 0, 0, \ldots \rangle, \\
\langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 1, 0, \ldots \rangle & R_0 & \langle \varepsilon_0, \varepsilon_0, \varepsilon_0, \ldots, 1, 0, \ldots \rangle, \\
\langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 1, 0, \ldots \rangle & R_\omega & \langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 0, 0, \ldots \rangle, \\
\langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 1, 0, \ldots \rangle & \cancel{R_\omega} & \langle \varepsilon_0, \varepsilon_0, \varepsilon_0, \ldots, 0, 0, \ldots \rangle.
\end{array}$$

We can finally state the universality theorem for $\mathcal{I}$.

Theorem 2.6.8 ([96])**.** GLP^0 *is sound and complete with respect to Ignatiev's model, that is:*

$$\mathsf{GLP}^0 \vdash \varphi \iff \mathcal{I} \vDash \varphi.$$

2.6.2 Arithmetical semantics

There are several different interpretations for the modalities of GLP. The first one, proposed by Japaridze [139] for GLP_ω, interprets $[n]\varphi$ as "φ is provable over T with at most n nested applications of the ω-rule", where T is a fixed theory of arithmetic and the ω-rule is an infinitary rule stating that if $P(\overline{0}), P(\overline{1}), P(\overline{2}), \ldots$ are all provable for a given predicate P, then $\forall x\, P(x)$ is too. The logic GLP_ω is sound and complete for this interpretation ([139], see also [136, 34] for simplified proofs). Fernández-Duque and Joosten [98] later extended it to the transfinite setting, showing soundness and completeness for GLP_Λ as long as Λ is a computable ordinal.

A different arithmetical interpretation was proposed by Beklemishev and Pakhomov [42] in first order arithmetic enriched with a collection of truth predicates indexed by the ordinals.

Another possible interpretation is to read $[n]\varphi$ as "φ is provable over T in the presence of all true Π^0_n-sentences." This interpretation has become popular and was used by Beklemishev to perform

[10] In particular, one can take ξ as the predecessor of ζ if ζ has a predecessor.

proof-theoretical analysis of PA and related logics [37, 29, 30]. A generalization of this interpretation to transfinite modalities has been proposed by Joosten [141].

The logic GLP_ω has been successfully used to perform a modular ordinal analysis of PA and related systems (see [29], and more recently [42]). A key feature in the ordinal analysis is that consistency operators $\langle n\rangle$ can be interpreted as reflection principles, which are finitely axiomatizable. Of note, GLP^0 actually suffices for various purposes. In fact, even the strictly positive fragment of GLP^0 suffices. We describe this fragment in the next section.

2.7 Strictly positive logic

Strictly positive logics are logics with the language reduced to the fragment containing only the connectives $\top$, $\wedge$, $\forall$ and $\Diamond$ (or $\langle\alpha\rangle$ in the polymodal setting). Thus we define the strictly positive fragments of $\mathcal{L}_\Box$, $\mathcal{L}_{\Box\forall}$, and $\mathcal{L}_{[\alpha],\alpha<\Lambda}$, respectively as:

$$\mathcal{L}_\Box^+ ::= \top \mid p \mid \varphi\wedge\varphi \mid \Diamond\varphi;$$
$$\mathcal{L}_{\Box\forall}^+ ::= \top \mid S(\boldsymbol{x}) \mid \varphi\wedge\varphi \mid \forall x\,\varphi \mid \Diamond\varphi;$$
$$\mathcal{L}_{[\alpha],\alpha<\Lambda}^+ ::= \top \mid p \mid \varphi\wedge\varphi \mid \langle\alpha\rangle\varphi, \text{with } \alpha<\Lambda.$$

Note that the above are the only available connectives. The absence of implication and negation does not allow us to define any of the other ones.[11] The statements of a strictly positive logic are implications between two strictly positive formulas, typically written $\varphi \vdash \psi$ (where φ, ψ are strictly positive) instead of $\varphi \to \psi$ to remind ourselves that implication cannot be nested.

It might seem strange that $\Diamond$ is said to be a positive connective, when its Kripke semantics is existential ("there is an accessible world where the formula holds"). However, we are going to be interested in strictly positive logics mainly from the provability point of view, and arithmetically it is the $\Box$ which has an existential flavor ("there is a proof of the formula").

One might think that $\mathcal{L}_\Box^+$ is too weak to say anything of interest, but even this language can lead to interesting studies [144]. The interplay between $\forall$ and $\Diamond$ adds an extra layer of complexity, and we study a non-trivial logic in this language in Chapters 4 and 5. In the case of $\mathcal{L}_{[\alpha],\alpha<\Lambda}^+$, the interplay between different $\langle\alpha\rangle$ modalities is a well of information, as we see during the rest of this section and in Chapter 3.

The study of strictly positive logics has been developed in several contexts, such as description logics [149, 143] and linear logics [144], but here we focus on so-called reflection calculi [78, 36, 94]. We present the original reflection calculus in this section and a new one in Chapter 4.

When considering a strictly positive logic P, one may ask whether there is some modal logic L whose strictly positive fragment coincides with P (see [38]). In that case we would have:

$$\varphi \vdash_P \psi \iff L \vdash \varphi \to \psi, \tag{2.2}$$

where φ and ψ are strictly positive formulas.

[11] There is a significant body of work done on (non strictly) positive logics [85, 58], where only negations and nested implications are disallowed, but we do not study them here.

The Reflection Calculus (RC_ω) is a propositional strictly positive logic (with language $\mathcal{L}^+_{[\alpha],\alpha<\omega}$) first introduced by Beklemishev [35] as an axiomatization for the strictly positive fragment of GLP_ω in the sense of (2.2). This fragment had already been previously studied and described by Dashkov [78] with a slightly different axiomatization. The transfinite version RC_Λ for an arbitrary computable ordinal Λ was introduced later [97], although modaly speaking working with RC_2, RC_ω, or RC_Λ for larger ordinals Λ is not much different, and so we often omit the subscript and write simply RC. We explicitly write RC_1 when considering the unimodal fragment.

Definition 2.7.1 (Reflection Calculus, RC_Λ, [78, 35, 97])**.** Let φ, ψ and χ be formulas in the language $\mathcal{L}^+_{[\alpha],\alpha<\Lambda}$, and $\alpha, \beta < \Lambda$ be ordinals.

The axioms and rules of RC_Λ are:

(i) $\varphi \vdash_{\mathsf{RC}} \varphi$ and $\varphi \vdash_{\mathsf{RC}} \top$;

(ii) $\varphi \wedge \psi \vdash_{\mathsf{RC}} \varphi$ and $\varphi \wedge \psi \vdash_{\mathsf{RC}} \psi$ (conjunction elimination);

(iii) if $\varphi \vdash_{\mathsf{RC}} \psi$ and $\varphi \vdash_{\mathsf{RC}} \chi$, then $\varphi \vdash_{\mathsf{RC}} \psi \wedge \chi$ (conjunction introduction);

(iv) if $\varphi \vdash_{\mathsf{RC}} \psi$ and $\psi \vdash_{\mathsf{RC}} \chi$, then $\varphi \vdash_{\mathsf{RC}} \chi$ (cut);

(v) if $\varphi \vdash_{\mathsf{RC}} \psi$, then $\langle\alpha\rangle\varphi \vdash_{\mathsf{RC}} \langle\alpha\rangle\psi$ (necessitation);

(vi) $\langle\alpha\rangle\langle\alpha\rangle\varphi \vdash_{\mathsf{RC}} \langle\alpha\rangle\varphi$ (transitivity);

(vii) $\langle\alpha\rangle\varphi \vdash_{\mathsf{RC}} \langle\beta\rangle\varphi$, for $\alpha > \beta$ (monotonicity);

(viii) $\langle\alpha\rangle\varphi \wedge \langle\beta\rangle\psi \vdash_{\mathsf{RC}} \langle\alpha\rangle(\varphi \wedge \langle\beta\rangle\psi)$ for $\alpha > \beta$ (poly-Euclidity).

If $\varphi \vdash_{\mathsf{RC}} \psi$, we say that ψ follows from φ in RC. If both $\varphi \vdash_{\mathsf{RC}} \psi$ and $\psi \vdash_{\mathsf{RC}} \varphi$, we say that φ and ψ are equivalent in RC and write $\varphi \equiv_{\mathsf{RC}} \psi$.

We briefly comment on the axiomatization of RC. The first four items are typical in any propositional logic presented in sequent-style. RC-necessitation is easily obtained in GLP through a mix of GLP-necessitation and distribution. Both transitivity and monotonicity in this setting are equivalent to their GLP counterparts. Finally, poly-Euclidity is a consequence of negative introspection (noting that $\mathsf{K} \vdash \Box\varphi \wedge \Diamond\psi \to \Diamond(\varphi \wedge \psi)$, which can be easily translated to the polymodal setting).

Like GLP, the logic RC_Θ is conservative over RC_Λ whenever $\Theta \geq \Gamma$.

Lemma 2.7.2. *Let* $\varphi, \psi \in \mathcal{L}^+_{[\alpha],\alpha<\Lambda}$ *and* $\Theta \geq \Lambda$. *Then:*

$$\varphi \vdash_{\mathsf{RC}_\Theta} \psi \implies \varphi \vdash_{\mathsf{RC}_\Lambda} \psi.$$

Proof. By the analogous result for GLP [40] and the conservativity of GLP_Γ over RC_Γ [78]. $\Box$

Even though RC has a strictly positive language, it is remarkably expressive, giving rise to an ordinal notation system [93] and being an appropriate tool for Π^0_1 ordinal analysis [42]. Furthermore, the arithmetical semantics becomes more expressive, as we see in Section 2.7.2.

Perhaps surprisingly, many applications of RC can in fact be performed using only its closed fragment RC^0. This is due to some special inhabitants of RC^0: the worms, called so due to their relation to the Worm Battle [32]. These are the formulas of RC^0 that have no conjunction symbols. It is provable in RC^0 that each of its formulas is equivalent to a single worm, a result which motivates Chapter 3.

Lemma 2.7.3 ([136, 93]). *For each formula φ of RC^0 there is a worm A such that $\varphi \equiv_{\mathrm{RC}} A$ with all the modalities appearing in A also appearing in φ.*

Worms can be read in various ways. One can conceive of them as consistency statements or reflection principles. Furthermore, natural fragments of arithmetic are denoted by worms [153]. The worms modulo provable equivalence can be ordered, and one can also conceived of them as ordinals [97]. Apart from their interpretation as consistency statements, reflection principles, fragments of arithmetic, or ordinals, worms also stand in an intimate relation with Turing progressions [140]. All of these mathematical entities can be manipulated and reasoned about within the rather simple modal logic RC^0.

We end this section with a striking observation. Strictly positive fragments have a tendency to have significantly lower complexity than their full language counterparts (assuming the polynomial hierarchy doesn't collapse somewhere, eg. that PTIME is different from PSPACE), as summarized in Table 2.1. Note that there are strictly positive logics that are not the strictly positive fragment of anything, such as $\mathrm{RC}\omega$ (not to be confused with RC_ω, where the ω is a subscript), a logic closely related to $\mathrm{RC}_{\omega+1}$ that includes the persistence axioms $\langle\omega\rangle\varphi \vdash \varphi$ (see [36] for more details).

Table 2.1: Complexity of some modal logics and their strictly positive counterparts. See the associated references for the definitions of the logics not previously mentioned in this thesis.

Full logic		Strictly positive fragment	
K	PSPACE-complete [150]	K^+	PTIME [38, 39]
K4 and GL	PSPACE-complete [106]	RC_1	PTIME [78, 35][a]
K4.3 and GL.3	coNP-complete [160]	$\mathrm{K4.3}^+$	PTIME [199]
GLP^0_Λ $(\Lambda \geq \omega)$[b]	PSPACE-complete [180]	RC^0_Λ	PTIME [78, 35][c]
GLP	PSPACE-complete [191]	RC	PTIME [78, 35]
T	PSPACE-complete [150]	$\{e_{refl}\}$	PTIME [143]
S4	PSPACE-complete [150]	$\mathcal{E}_{qo}$	PTIME [143]
S4.3	coNP-complete [106]	$\mathcal{E}_{lin}$	PTIME [143]
S5	coNP-complete [91]	$\mathrm{S5}^+$	PTIME [199]
$\mathrm{S5}_m$ $(m > 1)$	PSPACE-complete [91]	$\mathrm{S5}^+_m$	PTIME [199]
QPL(PA)	Π^0_2-complete [203]	QRC_1	decidable [14]

[a]Observing that RC is conservative over RC_1 by Lemma 2.7.2.
[b]$\mathrm{GLP}^0{}_n$ with $n < \omega$ is decidable in PTIME [180].
[c]Observing that RC is conservative over RC^0 (replace any propositional symbol in an RC-proof with $\top$ to obtain an RC^0-proof).

This reduction in complexity inspired us to study the strictly positive fragment of QPL(PA) as a hope that it would lead to a complexity lower than Π^0_2-complete. The resulting logic, which we called QRC_1, is indeed a simplification. For more details, see Chapters 4 and 5.

2.7.1 Relational semantics

There is nothing special about relational semantics for strictly positive logics, since we can restrict the notions of Kripke frame (or sheaf) and model to the relevant language fragment and keep everything else the same. However, it is worthwhile to briefly talk about relational semantics for RC.

With each $\mathcal{L}^{+}_{[\alpha],\alpha<\Lambda}$ formula φ we associate a model $\mathcal{T}[\varphi]$ based on the syntax tree of φ, with leaves representing diamond-free formulas, forks representing conjunctions of diamond formulas, and labeled arrows representing diamonds (see [78] for details). The model $\mathcal{T}[\varphi]$ is then the closure of this syntax tree of φ under transitivity for each R_α, monotonicity, and poly-Euclidity. Note that these are precisely the frame conditions associated with the axioms of RC, as previously mentioned. See Figure 2.2 for an example.

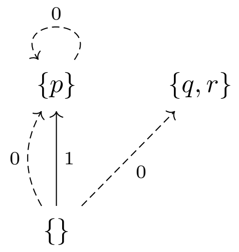

Figure 2.2: A graphical representation of $\mathcal{T}[\langle 1\rangle p \wedge \langle 0\rangle(q \wedge r)]$. Each node includes the set of propositional variables that hold at that world. An arrow between two worlds with a label n means that the second is reachable from the first through R_n. We use dashed arrows for R_0 and full arrows for R_1.

It is possible to characterize the sequents provable in RC using the $\mathcal{T}[\cdot]$ models, as the following theorem shows.

Theorem 2.7.4 (Dashkov [78])**.** *Let $\varphi, \psi \in \mathcal{L}^{+}_{[\alpha],\alpha<\Lambda}$. Then:*

$$\varphi \vdash_{\mathsf{RC}} \psi \iff \mathcal{T}[\varphi], r \Vdash \psi,$$

where r is the root of $\mathcal{T}[\varphi]$.

With this result, checking whether $\varphi \vdash_{\mathsf{RC}} \psi$ for specific formulas φ and ψ becomes entertainingly easy. In fact, it was with an algorithm based on it that Dashkov [78] showed how to decide RC in polynomial time.

2.7.2 Arithmetical semantics

As with relational semantics, we can reuse the arithmetical semantics available for the full languages. Thus, the arithmetical realizations for propositional modal logic (Definition 2.4.5) are available for $\mathcal{L}^{+}_{\Box}$, the ones for quantified modal logic (Definition 2.5.5) are available for $\mathcal{L}^{+}_{\Box\forall}$, and the several different arithmetical realizations mentioned for GLP (Section 2.6.2) are available for $\mathcal{L}^{+}_{[\alpha],\alpha<\Lambda}$ and in particular RC. However, the restricted language provides another avenue for arithmetical interpretations.

Typical arithmetical realizations interpret modal formulas as arithmetical formulas, which amounts to interpreting modal formulas as finite extensions of arithmetical theories. However, non finitely axiomatizable principles are not available as translations of modal formulas. In particular, limit modalities such as $\langle\omega\rangle$ have no obvious translation, since they would normally translate to non finitely axiomatizable reflection schemas [36]. Interpretations for strictly positive languages can overcome this by interpreting a modal formula as an axiomatization of a theory of arithmetic instead of as an arithmetical formula. This is not available in the full language, since the non-positive connectives have no clear counterparts. One can thus distinguish arithmetical semantics for strictly positive languages over whether they are finitary (like all the ones mentioned so far) or infinitary.

Recall from Section 2.3 that an axiomatization of a theory of arithmetic T is a formula $\tau(u)$ with one free variable u such that $\tau(u)$ is true in the standard model if and only if u is the Gödel number of an axiom of T.

Definition 2.7.5 (Infinitary arithmetical realization for $\mathcal{L}^+_\Box$, $\cdot^\circ$, $\cdot^{\circ_\tau}$)**.** Let $\cdot^\circ$ be a function from propositional symbols to axiomatizations of theories of arithmetic. Given a base axiomatization τ with one free variable u, we extend $\cdot^\circ$ to formulas of $\mathcal{L}^+_\Box$ as follows:

- $\top^{\circ_\tau} := \tau(u)$;
- $p^{\circ_\tau} := \tau(u) \vee p^\circ$;
- $(\varphi \wedge \psi)^{\circ_\tau} := \varphi^{\circ_\tau} \vee \psi^{\circ_\tau}$;
- $(\Diamond\varphi)^{\circ_\tau} := \tau(u) \vee (u = \ulcorner \Diamond_{\varphi^{\circ_\tau}} \top \urcorner)$.

Intuitively, we wish to interpret modal formulas as (axiomatizations of) theories, and in particular conjunctions of formulas as unions of theories. It makes sense, then, that $(\varphi \wedge \psi)^{\circ_\tau}$ is translated to a disjunction, since a formula is an axiom of a union of two theories if and only if it is an axiom of one or of the other. On the other hand, $\Diamond\varphi$ is interpreted as the base theory augmented with a single axiom stating the consistency of the theory axiomatized by φ°_τ}.

The same idea can be extended to both $\mathcal{L}^+_{\Box\forall}$ and $\mathcal{L}^+_{[\alpha],\alpha<\Lambda}$. We go into detail on the former in Chapter 5 and refer the reader to [36] for the latter.

3 Worm Calculus

Contents

3.1 Introduction

As mentioned in Section 2.6, the polymodal provability logic GLP is both PSPACE-complete and frame-incomplete. However, several fragments of GLP are still useful as provability logics. The simplest such previously available fragment was RC^0, the closed fragment of the Reflection Calculus. In this chapter, whose results were published in [12], we introduce a further simplification: the Worm Calculus (WC).

Recall from Lemma 2.7.3 that every formula in the language of RC^0 is equivalent to a worm. One may wonder whether there is some decent axiomatization of the worm fragment of RC^0. The current chapter settles this question in the positive, presenting a calculus WC that only manipulates worms, so that RC^0, and thus also GLP^0, are conservative extensions of WC.

In the last section of this chapter we dwell on semantics for WC. In particular we see that although WC has the finite model property, any (moderately nice) universal model for WC inherits much of the intrinsic complexity of Ignatiev's universal model for GLP^0.

The conservativity of RC^0 over WC was formalized in Coq [3], which required describing the language and axiomatization of both RC^0 and WC, as well as the formalization of most of the proofs presented in this chapter. Since these are elementary and self-contained results, the process was rather straightforward and we do not describe it in this thesis.

3.2 Axiomatization

Recall that a worm is a conjunction-free strictly positive formula in a polymodal language, or in other words, a sequence of diamonds ending in $\top$. We use capital letters to represent worms, and $\mathbb{W}(\Lambda)$ for the class of all worms in a given language $\mathcal{L}_{[\alpha],\alpha<\Lambda}$. When the Λ is clear from context we write simply $\mathbb{W}$. We further use $\mathbb{W}_\alpha$ to represent the class of worms with modalities at least α.

Since the language of worms is so simple, we omit the $\langle\cdot\rangle$, obtaining formulas which are strings of ordinals ending in $\top$. To further simplify, we write A and B for the worms $A\top$ and $B\top$. When we write AB this is understood as $AB\top$.

We propose a Worm Calculus, denoted by WC, which derives sequents of worms.

Definition 3.2.1 (Worm Calculus, WC_Λ)**.** Let A, B and C be worms in $\mathbb{W}(\Lambda)$, and $\alpha, \beta < \Lambda$ be ordinals. The axioms and rules of WC_Λ are the following:

(i) $A \vdash_{\mathsf{WC}} \top$;

(ii) $\alpha\alpha A \vdash_{\mathsf{WC}} \alpha A$ (transitivity);

(iii) $\alpha A \vdash_{\mathsf{WC}} \beta A$, for $\alpha > \beta$ (monotonicity);

(iv) if $A \vdash_{\mathsf{WC}} B$ and $B \vdash_{\mathsf{WC}} C$, then $A \vdash_{\mathsf{WC}} C$ (cut);

(v) if $A \vdash_{\mathsf{WC}} B$, then $\alpha A \vdash_{\mathsf{WC}} \alpha B$ (necessitation);

(vi) if $A \vdash_{\mathsf{WC}} B$ and $A \vdash_{\mathsf{WC}} \alpha C$, then $A \vdash_{\mathsf{WC}} B\alpha C$, for $B \in \mathbb{W}_{\alpha+1}$.

If $A \vdash_{\mathsf{WC}} B$, we say that B follows from A in WC. If both $A \vdash_{\mathsf{WC}} B$ and $B \vdash_{\mathsf{WC}} A$, we say that A and B are equivalent in WC_Λ, and write $A \equiv_{\mathsf{WC}} B$. We typically omit the Λ subscript in WC_Λ and write simply WC.

Some comments on the definition of WC. As mentioned before, it was inspired by RC^0, and as such we kept all axioms and rules that could be expressed in the reduced language of worms, with the exception of $A \vdash_{\mathsf{WC}} A$, which can be derived from the others (see Corollary 3.2.4 below). The ones that deal exclusively with conjunction are not important here, but Axiom 2.7.1.(viii) is. That axiom, also known as poly-Euclidity, states:

$$\langle\alpha\rangle\varphi \wedge \langle\beta\rangle\psi \vdash_{\mathsf{RC}} \langle\alpha\rangle(\varphi \wedge \langle\beta\rangle\psi).$$

Rule 3.2.1.(vi) was inspired by the above RC^0 axiom, using a particular kind of worm concatenation to represent conjunction, a trick we explore below.

The number of diamonds of a worm A is called the length of A. This serves as a notion of simplicity that allows us to recursively define operations on worms.

Definition 3.2.2 (Length)**.** The length of a worm A, denoted by $|A|$, is defined recursively as such: $|\top| := 0$ and $|\alpha A| := |A| + 1$.

We can immediately prove some facts about worms using the worm calculus.

Lemma 3.2.3. *For any worms A and B, we have that $AB \vdash_{\mathsf{WC}} A$.*

Proof. The proof goes by induction on the length of A. Starting from $B \vdash_{\mathsf{WC}} \top$ (base case), repeatedly apply necessitation (Rule 3.2.1.(v)) to build A up front. □

From this lemma we obtain a simple but useful corollary.

Corollary 3.2.4. *For any worm A, we have that $A \vdash_{\mathsf{WC}} A$.*

It is in general not true that $AB \vdash_{\mathsf{WC}} B$, but there is a special case.

Lemma 3.2.5. *For any ordinal α and worms A and B such that $A \in \mathbb{W}_{\alpha+1}$, we have that $A\alpha B \vdash_{\mathsf{WC}} \alpha B$.*

Proof. By induction on the length of A, with the help of transitivity and monotonicity (Axioms 3.2.1.(ii) and 3.2.1.(iii), respectively). □

Note how we have shown that the concatenation of worms can be interpreted as a conjunction of sorts, under some circumstances. As long as $A \in \mathbb{W}_{\alpha+1}$, we have the following:

- $A\alpha B \vdash_{\mathsf{WC}} A$ (Lemma 3.2.3);
- $A\alpha B \vdash_{\mathsf{WC}} \alpha B$ (Lemma 3.2.5);
- if $C \vdash_{\mathsf{WC}} A$ and $C \vdash_{\mathsf{WC}} \alpha B$ then $C \vdash_{\mathsf{WC}} A\alpha B$ (Rule 3.2.1.(vi)).

Finally, we show how worms of length one can be obtained, which will be useful later.

Lemma 3.2.6. *For any non-trivial worm $A \in \mathbb{W}_\alpha$, we have that $A \vdash_{\mathsf{WC}} \alpha$.*

Proof. By induction on the length of A. If $|A| = 1$, then $A = \beta$ for some $\beta \geq \alpha$. The result follows by monotonicity (Axiom 3.2.1.(iii)) and Corollary 3.2.4. For the induction step, consider $A = \beta A'$, where $\beta \geq \alpha$ and we already know $A' \vdash_{\mathsf{WC}} \alpha$. Then by necessitation (Rule 3.2.1.(v)) and transitivity (Axiom 3.2.1.(ii)), $\alpha A' \vdash_{\mathsf{WC}} \alpha\alpha \vdash_{\mathsf{WC}} \alpha$. Since $\beta A' \vdash_{\mathsf{WC}} \alpha A'$, we are done. □

It is easy to see that RC extends WC. As we shall later see, RC is conservative over WC, which means that this extension is, in a sense, not proper. The first of these two claims is articulated in the following theorem.

Theorem 3.2.7. *For any two worms A and B we have that $A \vdash_{\mathsf{WC}} B$ implies $A \vdash_{\mathsf{RC}} B$.*

Proof. By an easy induction on the length of a WC proof. To see that Rule 3.2.1.(vi) is admissible in RC, we use induction on the length of B and RC's poly-Euclidity axiom (Axiom 2.7.1.(viii)). □

The proof of the converse is a bit more involved. We shall use the fact that an implication between worms can be recursively broken down into implications between simpler worms.

3.3 Worm decomposition

The notions of α-head and α-remainder are useful to break down worms into smaller ones.

Definition 3.3.1 (α-head, α-remainder)**.** Let A be a worm and α be an ordinal.

The α-head of A, denoted by $h_\alpha(A)$, is defined inductively as:

- $h_\alpha(\top) := \top$;
- $h_\alpha(\beta A) := \beta h_\alpha(A)$ if $\beta \geq \alpha$;
- $h_\alpha(\beta A) := \top$ if $\beta < \alpha$.

Likewise, the α-remainder of A, denoted by $r_\alpha(A)$, is defined inductively as:

- $r_\alpha(\top) := \top$;
- $r_\alpha(\beta A) := r_\alpha(A)$ if $\beta \geq \alpha$;
- $r_\alpha(\beta A) := \beta A$ if $\beta < \alpha$.

Intuitively, the α-head of A is the greatest initial segment of A which is in $\mathbb{W}_\alpha$, and the α-remainder is what remains after cutting off the α-head. It then follows that $A = h_\alpha(A) r_\alpha(A)$, for every worm A and ordinal α. An immediate consequence is that the lengths of the α-head and of the α-remainder of a worm are always at most the length of the worm itself.

It is possible to prove that $A \equiv_{\mathsf{RC}} h_\alpha(A) \wedge r_\alpha(A)$ for every worm A and ordinal α. In WC we cannot state such a result due to the lack of the conjunction connective in the language. We can, however, obtain the same consequences.

Lemma 3.3.2. *Let A and B be worms and α be an ordinal. Then:*

1. $A \vdash_{\mathsf{WC}} h_\alpha(A)$*;*

2. $A \vdash_{\mathsf{WC}} r_\alpha(A)$*;*

3. If $B \vdash_{\mathsf{WC}} h_\alpha(A)$ *and* $B \vdash_{\mathsf{WC}} r_\alpha(A)$*, then* $B \vdash_{\mathsf{WC}} A$*.*

Proof. Note that $A = h_\alpha(A)r_\alpha(A)$, this is to say, they are syntactically the same. Thus, Item 1 follows from Lemma 3.2.3. Item 2 is a consequence of Lemma 3.2.5, taking into consideration that $h_\alpha(A) \in \mathbb{W}_\alpha$ and that $r_\alpha(A)$ always starts with either $\top$, rendering the result trivial, or with an ordinal less than α. Item 3 follows from Rule 3.2.1.(vi) unless $r_\alpha(A) = \top$, in which case it is trivial. □

There is another relevant part of a worm, the α-body. It is obtained from the $(\alpha+1)$-remainder by dropping its leftmost modality (as long as said remainder is not trivial).

Definition 3.3.3 (α-body)**.** Let A be a worm and α an ordinal. The α-body of A, denoted by $b_\alpha(A)$, is defined from $r_{\alpha+1}(A)$ as follows:

- if $r_{\alpha+1}(A) = \top$, then $b_\alpha(A) := \top$;
- if $r_{\alpha+1}(A) = \beta B$, then $b_\alpha(A) := B$.

The α-body of a non-trivial worm A is particularly useful because its length is always strictly smaller than the length of A. We can also prove a counterpart of Lemma 3.3.2 about the α-body.

Lemma 3.3.4. *Let* α *be an ordinal,* A *be a non-trivial worm in* $\mathbb{W}_\alpha$ *and* B *be any worm. Then:*

1. $A \vdash_{\mathsf{WC}} \alpha b_\alpha(A)$*;*

2. if $B \vdash_{\mathsf{WC}} h_{\alpha+1}(A)$ *and* $B \vdash_{\mathsf{WC}} \alpha b_\alpha(A)$*, then* $B \vdash_{\mathsf{WC}} h_{\alpha+1}(A)\alpha b_\alpha(A)$*;*

3. $h_{\alpha+1}(A)\alpha b_\alpha(A) \vdash_{\mathsf{WC}} A$*;*

4. $A \equiv_{\mathsf{WC}} h_{\alpha+1}(A)\alpha b_\alpha(A)$*.*

Proof. We make a case distinction on $r_{\alpha+1}(A)$ in order to prove Items 1 to 3 separately in each case. Note that Item 4 is a corollary of the others.

Suppose that $r_{\alpha+1}(A) = \beta b_\alpha(A)$ for some ordinal β. Since $A \in \mathbb{W}_\alpha$, then $\beta \geq \alpha$. But since it is in the $(\alpha+1)$-remainder, $\beta < \alpha + 1$. We conclude that $\beta = \alpha$, and hence that $r_{\alpha+1}(A) = \alpha b_\alpha(A)$. Then Items 1 to 3 are just a corollary of Lemma 3.3.2.

However it can be the case that $r_{\alpha+1}(A) = \top$ and hence $b_\alpha(A) = \top$ as well. Then Item 1 becomes an instance of Lemma 3.2.6, Item 2 follows from Rule 3.2.1.(vi) and Item 3 is a consequence of Lemma 3.2.3. □

The following result describes part of a recursive decision procedure for provability in RC between worms.

Lemma 3.3.5 ([97, Lemma 3.15])**.** *For any two worms* A *and* B *and for any ordinal* α *we have that* $A \vdash_{\mathsf{RC}} \alpha B$ *if and only if both* $h_\alpha(A) \vdash_{\mathsf{RC}} \alpha h_\alpha(B)$ *and* $A \vdash_{\mathsf{RC}} r_\alpha(B)$*.*

Let us see that we can prove one of the implications in WC, which we will later use in the proof of our main theorem (Theorem 3.5.1). There is no *a priori* reason why the other implication can't also hold (in fact, we will see later that it does, since the calculi are equivalent for worms), but it won't be needed here.

Lemma 3.3.6. *For any two worms A and B, and for any ordinal α, if we have that $h_\alpha(A) \vdash_{\mathsf{WC}} \alpha h_\alpha(B)$ and $A \vdash_{\mathsf{WC}} r_\alpha(B)$, then we have $A \vdash_{\mathsf{WC}} \alpha B$.*

Proof. Taking into consideration that $A = h_\alpha(A) r_\alpha(A)$ and similarly for B, consider two cases. In the first case, $r_\alpha(B) = \top$, and the result is a consequence of Lemma 3.3.2. In the second case, $r_\alpha(B) = \beta C$ for some $\beta < \alpha$ and worm C. Then the result follows from Rule 3.2.1.(vi). □

We now want to prove that RC is conservative over WC using the following inductive strategy. If $A \vdash_{\mathsf{RC}} B$, we use Lemma 3.3.5 to recast this into a collection of provability statements in RC between worms with smaller lengths. We then translate them to WC using the induction hypotheses, and finally go back with the help of Lemma 3.3.6. However, depending on the worms A and B, it could be the case that these two theorems are not enough, since they don't always reduce the length of the provability statements. In what follows, we introduce some more useful notions and results, which will help us deal with that problem.

3.4 Well-founded orders on worms

It is possible to define an order relation between worms as is standard in the literature.

Definition 3.4.1 (Ordering worms)**.** We say that $A <_\alpha^{\mathsf{WC}} B$ if $B \vdash_{\mathsf{WC}} \alpha A$. Furthermore, we say that $A \leq_\alpha^{\mathsf{WC}} B$ if either $A <_\alpha^{\mathsf{WC}} B$ or $A \equiv_{\mathsf{WC}} B$. The provability can be taken in RC to obtain $<_\alpha^{\mathsf{RC}}$ and $\leq_\alpha^{\mathsf{RC}}$, respectively. During the rest of this chapter we write $<_\alpha$ and $\leq_\alpha$ instead of $<_\alpha^{\mathsf{WC}}$ and $\leq_\alpha^{\mathsf{WC}}$.

It is well-known that $<_\alpha^{\mathsf{RC}}$ is irreflexive [40]. Since WC is embedded in RC, we also know that $<_\alpha$ is irreflexive. It is easy to see that both relations are transitive.

Our goal now is to show that $<_\alpha$ is a total relation over worms in $\mathbb{W}_\alpha$. This has been shown for $<_\alpha^{\mathsf{RC}}$ using worm normal forms [31, 40], but here we follow a different strategy, proposed by Fernández-Duque [93]. We start by presenting a number of useful sufficient conditions to deduce $A <_\alpha B$, and one to deduce $A \equiv_{\mathsf{WC}} B$.

Lemma 3.4.2. *Let $A, B \in \mathbb{W}_\alpha$ such that $A, B \neq \top$. Then in* WC *(and hence in* RC*) we have the following:*

1. *If $b_\alpha(B) \vdash A$, then $A <_\alpha B$;*
2. *If $A <_\alpha b_\alpha(B)$, then $A <_\alpha B$;*
3. *If $b_\alpha(A) <_\alpha B$ and $h_{\alpha+1}(A) <_{\alpha+1} h_{\alpha+1}(B)$, then $A <_{\alpha+1} B$ (and consequently $A <_\alpha B$);*
4. *If $b_\alpha(A) <_\alpha B$ and $b_\alpha(B) <_\alpha A$ and $h_{\alpha+1}(A) \equiv h_{\alpha+1}(B)$, then $A \equiv B$.*

Proof. For the first item, from $b_\alpha(B) \vdash A$ we get by necessitation that $\alpha b_\alpha(B) \vdash \alpha A$. Since by Lemma 3.3.4 we know that $B \vdash \alpha b_\alpha(B)$, we can conclude that $B \vdash \alpha A$. The second item follows by the transitivity of $<_\alpha$, taking into account that $b_\alpha(B) <_\alpha B$ (Lemma 3.3.4). The third item follows from Lemma 3.3.6. Finally, for the fourth item we use $B \vdash \alpha b_\alpha(A)$ and $h_{\alpha+1}(B) \vdash h_{\alpha+1}(A)$ to get $B \vdash h_{\alpha+1}(A)\alpha b_\alpha(A)$, and hence $B \vdash A$. Then we obtain $A \vdash B$ in the same way. $\square$

Now we are ready to prove the totality of $<_\alpha$ for worms in $\mathbb{W}_\alpha$, following the same strategy as Fernández-Duque [93].

Lemma 3.4.3 (Trichotomy)**.** *Given worms $A, B \in \mathbb{W}_\alpha$, we have that either $A <_\alpha B$, or $A \equiv_{\mathsf{WC}} B$, or $B <_\alpha A$.*

Proof. We proceed by induction on the length of AB. If the length is zero, *i.e.*, if $A = B = \top$, then clearly $A \equiv_{\mathsf{WC}} B$.

Note that by Lemma 3.2.6, $\top <_\alpha C$, as long as $C \neq \top$ and $C \in \mathbb{W}_\alpha$. Then if exactly one of A or B is $\top$ we have also solved our problem.

Now for the induction step, take both A and B with positive length. Our induction hypothesis is:

For any ordinal β and worms $C, D \in \mathbb{W}_\beta$ such that $|CD| < |AB|$, we have that either $C <_\beta D$, or $C \equiv_{\mathsf{WC}} D$, or $D <_\beta C$.

Let ξ be the minimum ordinal in AB, which means that $\alpha \leq \xi$. According to Lemma 3.4.2, if $A \leq_\xi b_\xi(B)$ or $B \leq_\xi b_\xi(A)$, we can conclude $A <_\xi B$ or $B <_\xi A$, respectively. Assume then that $A \not\leq_\xi b_\xi(B)$ and $B \not\leq_\xi b_\xi(A)$. Since we assume $A \neq \top$, it is clear that $|b_\xi(A)| < |A|$, and analogously for B. Then by the induction hypothesis we have $b_\xi(B) <_\xi A$ and $b_\xi(A) <_\xi B$.

Since we are assuming ξ is in AB, we also know that:

$$|h_{\xi+1}(A)h_{\xi+1}(B)| < |AB|,$$

and thus by the induction hypothesis we have:

$$\begin{gathered} h_{\xi+1}(A) <_{\xi+1} h_{\xi+1}(B), \text{ or} \\ h_{\xi+1}(B) <_{\xi+1} h_{\xi+1}(A), \text{ or} \\ h_{\xi+1}(A) \equiv_{\mathsf{WC}} h_{\xi+1}(B). \end{gathered}$$

Then by Lemma 3.4.2, we can conclude $A <_\xi B$, or $B <_\xi A$, or $A \equiv_{\mathsf{WC}} B$, respectively.

As a final remark, we observe that since $\alpha \leq \xi$, we have that $C <_\xi D$ implies $C <_\alpha D$ for any worms $C, D \in \mathbb{W}_\xi$. $\square$

The following technical result can now be readily proved. It will be useful in Section 3.6.

Corollary 3.4.4. *If $A, B \in \mathbb{W}_\alpha$, then either $\alpha A \vdash_{\mathsf{WC}} \alpha B$ or $\alpha B \vdash_{\mathsf{WC}} \alpha A$.*

Proof. By Lemma 3.4.3, either $A \vdash_{\mathsf{WC}} \alpha B$, $A \equiv_{\mathsf{WC}} B$, or $B \vdash_{\mathsf{WC}} \alpha A$. If $A \vdash_{\mathsf{WC}} B$, then immediately by necessitation $\alpha A \vdash_{\mathsf{WC}} \alpha B$. On the other hand, if $A \vdash_{\mathsf{WC}} \alpha B$ we can easily obtain $\alpha A \vdash_{\mathsf{WC}} \alpha B$ by necessitation and transitivity, and similarly for the $B \vdash_{\mathsf{WC}} \alpha A$ case. $\square$

With the totality of $<_\alpha$ on $\mathbb{W}_\alpha$ at hand, we can already show that proofs in RC and WC are equivalent for certain worms.

Theorem 3.4.5. *If $A, B \in \mathbb{W}_\alpha$, then:*

$$A <_\alpha B \iff A <_\alpha^{\mathsf{RC}} B;$$
$$A \equiv_{\mathsf{WC}} B \iff A \equiv_{\mathsf{RC}} B.$$

Proof. Both left-to-right implications are a consequence of RC extending WC (Theorem 3.2.7).

For the right-to-left implication of the first statement, we reason as follows: suppose that $A <_\alpha^{\mathsf{RC}} B$ but it is not the case that $A <_\alpha B$. Then by the totality of $<_\alpha$ for worms in $\mathbb{W}_\alpha$ (Lemma 3.4.3), either $B <_\alpha A$, or $A \equiv_{\mathsf{WC}} B$. By the inclusion of WC in RC, we then conclude that either $B <_\alpha^{\mathsf{RC}} A$, or $A \equiv_{\mathsf{RC}} B$. But neither of these two cases is possible, as they contradict the irreflexivity of $<_\alpha^{\mathsf{RC}}$. Then it must be the case that $A <_\alpha B$. The proof of the second statement is analogous. □

3.5 Conservativity of RC over WC

We are now ready to prove the main result of this chapter.

Theorem 3.5.1. *For any worms A and B', if $A \vdash_{\mathsf{RC}} B'$, then $A \vdash_{\mathsf{WC}} B'$.*

Proof. If $B' = \top$, the result is immediate. Assume then that $B' = \alpha B$ for some ordinal α and worm B.

The proof proceeds by induction on the length of $A\alpha B$. The minimum length of $A\alpha B$ is 1, and it occurs only when $A = B = \top$. The premise $\top \vdash_{\mathsf{RC}} \alpha$ is absurd since it would contradict the irreflexivity of $<_\alpha^{\mathsf{RC}}$, and hence there is nothing left to prove in the base case.

For the induction step, our induction hypothesis is the following:

For any worms C, D such that $|CD| < |A\alpha B|$ and $C \vdash_{\mathsf{RC}} D$, we have that $C \vdash_{\mathsf{WC}} D$.

Assume that $A \vdash_{\mathsf{RC}} \alpha B$. From Lemma 3.3.5 we get $h_\alpha(A) \vdash_{\mathsf{RC}} \alpha h_\alpha(B)$ and $A \vdash_{\mathsf{RC}} r_\alpha(B)$.

If both $r_\alpha(A) = \top$ and $r_\alpha(B) = \top$, then $A, B \in \mathbb{W}_\alpha$, and hence $A \vdash_{\mathsf{WC}} \alpha B$ by Theorem 3.4.5. Suppose then that either $r_\alpha(A) \neq \top$ or $r_\alpha(B) \neq \top$. Since $|A| = |h_\alpha(A)| + |r_\alpha(A)|$ (and similarly for B), we know that $|h_\alpha(A)| < |A|$ or $|h_\alpha(B)| < |B|$. Then $|h_\alpha(A)\alpha h_\alpha(B)| < |A\alpha B|$. Furthermore, $|Ar_\alpha(B)| < |A\alpha B|$. By using the induction hypothesis twice we get $h_\alpha(A) \vdash_{\mathsf{WC}} \alpha h_\alpha(B)$ and $A \vdash_{\mathsf{WC}} r_\alpha(B)$, which is enough to show $A \vdash_{\mathsf{WC}} \alpha B$ by Lemma 3.3.6. □

The proof of the preceding result (together with the proofs of the results it uses) gives us a constructive algorithm to decide whether $A \vdash_{\mathsf{WC}} B$. Furthermore, if indeed $A \vdash_{\mathsf{WC}} B$, this algorithm provides a list of syntactical steps which form a formal proof. Since at each iteration of the recursion we may need to decide several different statements with only slightly smaller lengths, the algorithm is *a priori* exponential. It is known that there is a polynomial procedure to decide RC [78] (and hence, as we've seen, WC), but it uses semantics. Finding a polynomial syntactical algorithm remains an open problem.

Combining Theorems 3.2.7 and 3.5.1, we obtain the promised result: RC is a conservative extension of WC.

Theorem 3.5.2. *For any worms A and B we have that $A \vdash_{\mathsf{RC}} B$ if and only if $A \vdash_{\mathsf{WC}} B$.*

Combining this theorem with Lemma 2.7.3 we obtain the following corollary.

Corollary 3.5.3. *Let φ and ψ be closed* RC *formulas and A and B be worms such that $\varphi \equiv_{\mathsf{RC}} A$ and $\psi \equiv_{\mathsf{RC}} B$. Then we have $\varphi \vdash_{\mathsf{RC}} \psi$ if and only if $A \vdash_{\mathsf{WC}} B$.*

3.6 Universal models

Inspired by the finite model property of RC (see Section 2.7.1) and whence of WC, in this section we question whether WC may have a universal model $\mathcal{U}$ that is significantly simpler than Ignatiev's model $\mathcal{I}$ described in Definition 2.6.7. We settle the answer to this question in the positive and in the negative.

Positively, $\mathcal{U}$ can be simpler in that we can bound the length of the strict chains of successors in $\mathcal{U}$ by ω. For this, it suffices to take the disjoint union of all finite RC^0 counter-models for all statements for which $A \nvdash_{\mathsf{RC}} B$. This clearly defines a universal model for RC^0 with only finite strict R_α-chains, whereas Ignatiev's model $\mathcal{I}$ has arbitrarily long strict R_α-chains.

Negatively, we shall see that a large class of universal models $\mathcal{U}$ inherit much of the intrinsic complexity of $\mathcal{I}$ in that for infinitely many essentially different points $f \in \mathcal{I}$ we can find corresponding points $w \in \mathcal{U}$ such that f and w have the same modal theory.

We begin with some preliminary definitions and results about $\mathcal{I}$, firstly discussing an important subset of its worlds with rather nice properties: the main axis.

Definition 3.6.1 (Main axis, MA)**.** By MA we denote the main axis of Ignatiev's model $\mathcal{I}$ and define it as such: $f \in \mathsf{MA}$ if and only if $\forall \zeta \ \forall \xi < \zeta \ f(\zeta) = \ell^{-\xi+\zeta}(f(\xi))$.

For example, $\langle \omega^{\varepsilon_0+1}, \varepsilon_0, \varepsilon_0, \ldots, 1, 0, \ldots \rangle$ is not on the main axis, whereas $\langle \omega^{\varepsilon_0+1}, \varepsilon_0+1, 0, \ldots \rangle$ is. One of the nice properties of the main axis is that each point on it is modally definable.

Lemma 3.6.2 (Fernández-Duque and Joosten [96])**.** *For each $f \in$ MA there is a worm A such that for any given $g \in \mathcal{I}$, we have $\mathcal{I}, g \Vdash A \wedge [0]\neg A$ if and only if $f = g$. Moreover, each worm A defines a point on the main axis via $A \wedge [0]\neg A$.*

We now present a way of merging two worlds of $\mathcal{I}$ that preserves the satisfiability of worms, which will be useful to prove a characterization lemma below. Given two sequences f and g and an ordinal α, we can create a new sequence by considering the $(< \alpha)$ part of the first one and the $(\geq \alpha)$ part of the second one. In that case, we write $\alpha(f,g)$ for the resulting sequence. In symbols, $\alpha(f,g)_\zeta := f_\zeta$ for $\zeta < \alpha$ and $\alpha(f,g)_\zeta := g_\zeta$ for $\zeta \geq \alpha$. Clearly, whenever both f and g are ℓ-sequences and $g_\alpha \leq f_\alpha$, we have that $\alpha(f,g)$ is again an ℓ-sequence. Furthermore, this new ℓ-sequence satisfies any worms that both f and g do, as the following lemma shows.

Lemma 3.6.3. *For any worm A and any ℓ-sequences f and g with $g_\alpha \leq f_\alpha$ we have that if both $\mathcal{I}, f \Vdash A$ and $\mathcal{I}, g \Vdash A$, then $\mathcal{I}, \alpha(f,g) \Vdash A$.*

Proof. By induction on A with the base case being trivial. Thus we consider the inductive case where $A = \zeta B$ for some ζ, assuming $\mathcal{I}, f \Vdash \zeta B$ and $\mathcal{I}, g \Vdash \zeta B$.

In the case where $\zeta < \alpha$, we see that for any h such that $fR_\zeta h$, we also have $\alpha(f, g)R_\zeta h$, which tells us that $\mathcal{I}, \alpha(f, g) \Vdash \zeta B$.

In the case where $\zeta \geq \alpha$, we find h and h' such that $fR_\zeta h$ and $\mathcal{I}, h \Vdash B$ on the one hand, and $gR_\zeta h'$ and $\mathcal{I}, h' \Vdash B$ on the other. If $\zeta > \alpha$, note that $h_\alpha = f_\alpha \geq g_\alpha = h'_\alpha$ by the definition of R_ζ, so $\alpha(h, h')$ is also an ℓ-sequence. By the induction hypothesis, we know that $\mathcal{I}, \alpha(h, h') \Vdash B$. Since $\alpha(f, g)R_\zeta\alpha(h, h')$, we see that $\mathcal{I}, \alpha(f, g) \Vdash \zeta B$, as was to be shown.

If, however, $\zeta = \alpha$, it might be that $h_\alpha < h'_\alpha$ (if not, the reasoning of the $\zeta > \alpha$ case can be repeated). In that case, $\alpha(h, h')$ is not necessarily an ℓ-sequence and we cannot use the induction hypothesis.[1] What we can do is observe that if $h_\alpha < h'_\alpha$ then necessarily $h_\alpha < g_\alpha$ (because $gR_\alpha h'_\alpha$). Then $\alpha(f, g)R_\alpha h$ and consequently $\mathcal{I}, \alpha(f, g) \Vdash \alpha B$, as we wanted to show. □

We now define a relation $g \succeq f$ on $\mathcal{I}$ as g being point-wise at least f, that is, $g \succeq f$ if and only if for all ξ we have $g_\xi \geq f_\xi$. The following lemma tells us that this relation and the point f where a worm A is true for the first time characterize all the points of $\mathcal{I}$ where A holds.

Lemma 3.6.4. *Let A be a worm and $f, g \in \mathcal{I}$. We have that:*

$$\mathcal{I}, f \Vdash A \implies (g \succeq f \implies \mathcal{I}, g \Vdash A);$$
$$\mathcal{I}, f \Vdash A \wedge [0]\neg A \implies (g \succeq f \iff \mathcal{I}, g \Vdash A).$$

Proof. The first item is proven by an easy induction on A. It was already observed by Icard [131, Lemma 2.4.3]. Note also that the $\Longrightarrow$ direction of the second item follows from the first one.

For the $\Longleftarrow$ direction of the second item, we fix f such that $\mathcal{I}, f \Vdash A \wedge [0]\neg A$, consider g such that $\mathcal{I}, g \Vdash A$, and assume for a contradiction that $g \not\succeq f$. Let ξ be the smallest ordinal such that $g_\xi < f_\xi$. Then it is easy to see that $fR_\xi\xi(f, g)$. Since $\mathcal{I}, \xi(f, g) \Vdash A$ by Lemma 3.6.3, it follows that $\mathcal{I}, f \Vdash \xi A$. But this implies $\mathcal{I}, f \Vdash 0A$ by the soundness of $\mathcal{I}$ (Theorem 2.6.8), which contradicts the assumption that $\mathcal{I}, f \Vdash [0]\neg A$. □

With this characterization lemma, we can easily prove the following admissible rule for GLP^0.

Lemma 3.6.5. *Let A and B be worms. Then:*

$$\mathsf{GLP}^0 \vdash A \wedge [0]\neg A \to B \implies \mathsf{GLP}^0 \vdash A \to B.$$

Proof. Given a worm A, let f be the unique ℓ-sequence where $A \wedge [0]\neg A$ holds (see Lemma 3.6.2). Clearly, since GLP^0 $\vdash A \wedge [0]\neg A \to B$, we also have that $\mathcal{I}, f \Vdash B$. We prove that for any $g \in \mathcal{I}$, if $\mathcal{I}, g \Vdash A$, then $\mathcal{I}, g \Vdash B$, from which the result follows by completeness (Theorem 2.6.8). By Lemma 3.6.4, if $\mathcal{I}, g \Vdash A$, then $g \succeq f$. But then, using the other part of the same lemma, we may conclude that $\mathcal{I}, g \Vdash B$. □

[1] This possibility was not considered in the original proof published in [12], of which this is a correction.

We now prove two simple results about poly-Euclidean models. Recall that a model is called poly-Euclidean whenever $wR_\alpha u$ and $wR_\beta v$ imply $uR_\beta v$ for $\beta < \alpha$.

Lemma 3.6.6. *Let $\mathcal{M}$ be any poly-Euclidean model and $w \in \mathcal{M}$. Then, for worms A and B with $A \in \mathbb{W}_{\alpha+1}$, if $\mathcal{M}, w \Vdash A$ and $\mathcal{M}, w \Vdash \alpha B$, we also have that $\mathcal{M}, w \Vdash A\alpha B$.*

Proof. By induction on the length of A with the base case being trivial. For the inductive case, suppose that $\mathcal{M}, w \Vdash \gamma A$ for some $\gamma > \alpha$, and that $\mathcal{M}, w \Vdash \alpha B$. Then there is $u \in \mathcal{M}$ such that $wR_\gamma u$ and $\mathcal{M}, u \Vdash A$. Likewise, there is $v \in \mathcal{M}$ such that $wR_\alpha v$ and $\mathcal{M}, v \Vdash B$. Since $\mathcal{M}$ is poly-Euclidean, we know that $uR_\alpha v$, which allows us to prove $\mathcal{M}, u \Vdash \alpha B$ and thus obtain $\mathcal{M}, u \Vdash A\alpha B$ by the induction hypothesis. Then clearly $\mathcal{M}, w \Vdash \gamma A\alpha B$. ☐

With this lemma at hand we can show that although a poly-Euclidean model for WC cannot speak directly about conjunctions, it can indirectly do so. We say that a model $\mathcal{M}$ is a model for WC when $\varphi \vdash_{\mathsf{WC}} \psi$ implies that for every world $w \in \mathcal{M}$ we have $\mathcal{M}, w \Vdash \varphi \implies \mathcal{M}, w \Vdash \psi$.

Lemma 3.6.7. *Let $\mathcal{M}$ be a poly-Euclidean model for* WC *and $w \in \mathcal{M}$. Let A, B and C be worms such that $A \wedge B \equiv_{\mathsf{RC}} C$. Then, if both $\mathcal{M}, w \Vdash A$ and $\mathcal{M}, w \Vdash B$, we also have $\mathcal{M}, w \Vdash C$.*

Proof. By induction on the number of different symbols of AB following the standard proof that each conjunction of worms is equivalent to a worm in the same language (Lemma 2.7.3 here, but see also [31, Lemma 9]). The base case is trivial. For the induction step, let α be the minimal modality of AB and $C \equiv_{\mathsf{RC}} A \wedge B$. From Lemmas 3.3.2 and 3.3.4 together with the assumptions that $\mathcal{M}, w \Vdash A$ and $\mathcal{M}, w \Vdash B$ we collect the following observations:

- $\mathcal{M}, w \Vdash h_{\alpha+1}(A)$;
- $\mathcal{M}, w \Vdash h_{\alpha+1}(B)$;
- $\mathcal{M}, w \Vdash \alpha b_\alpha(A)$;
- $\mathcal{M}, w \Vdash \alpha b_\alpha(B)$.

Let D be provably equivalent to $h_{\alpha+1}(A) \wedge h_{\alpha+1}(B)$ and such that $D \in \mathbb{W}_{\alpha+1}$ (just like $h_{\alpha+1}(A) \in \mathbb{W}_{\alpha+1}$ and $h_{\alpha+1}(B) \in \mathbb{W}_{\alpha+1}$). Then, since there is no α in D, we know that $\mathcal{M}, w \Vdash D$ by the induction hypothesis. By Corollary 3.4.4, we know that either $b_\alpha(A) <_\alpha \alpha b_\alpha(B)$ or $b_\alpha(B) <_\alpha \alpha b_\alpha(A)$. Let αE be the maximum. We obtain $\mathcal{M}, w \Vdash D\alpha E$ by Lemma 3.6.6.

We now see that $D\alpha E \equiv_{\mathsf{RC}} A \wedge B$: the right-to-left direction is a simple application of a generalization of Rule 3.2.1.(vi) (which is admissible in RC as seen in Theorem 3.2.7). For the left-to-right direction, we prove $D\alpha E \vdash_{\mathsf{WC}} A$, with $D\alpha E \vdash_{\mathsf{WC}} B$ being analogous. By Lemma 3.3.4 and Rule 3.2.1.(vi) it suffices to show $D\alpha E \vdash_{\mathsf{WC}} h_{\alpha+1}(A)$ and $D\alpha E \vdash_{\mathsf{WC}} \alpha b_\alpha(A)$. The former is a consequence of Lemma 3.2.3 and the latter is a consequence of Lemma 3.2.5, noting that $D \in \mathbb{W}_{\alpha+1}$.

It remains to show that $\mathcal{M}, w \Vdash C$. Since $C \equiv_{\mathsf{RC}} A \wedge B$, we know by transitivity of $\equiv_{\mathsf{RC}}$ that $C \equiv_{\mathsf{RC}} D\alpha E$, and consequently that $C \equiv_{\mathsf{WC}} D\alpha E$. Then the result follows by the assumption that $\mathcal{M}$ is a WC model. ☐

We now finally consider universal WC models and prove the main result of this section. A model $\mathcal{U}$ is a universal WC model when:

$$A \vdash_{\mathsf{WC}} B \iff (\text{for all } w \in \mathcal{U}, \text{we have } \mathcal{U}, w \Vdash A \implies \mathcal{U}, w \Vdash B).$$

Given a model $\mathcal{M}$, let $\mathsf{Th}_{\mathcal{M}}(w) := \{A \mid \mathcal{M}, w \Vdash A\}$ be the collection of worms satisfied at the world w of $\mathcal{M}$. We are finally ready to prove the main result of this section.

Theorem 3.6.8. *Let $\mathcal{U}$ be a poly-Euclidean universal model for* WC*. Then, for each point $f \in \mathcal{I}$ with $f \in$* MA*, there is some world $w \in \mathcal{U}$ such that $\mathsf{Th}_{\mathcal{I}}(f) = \mathsf{Th}_{\mathcal{U}}(w)$.*

Proof. Let $f \in$ MA be arbitrary and let A be the worm given by Lemma 3.6.2 such that $A \wedge [0]\neg A$ is true at f and nowhere else. Since $A \nvdash_{\mathsf{WC}} 0A$ by the irreflexivity of $<_0$, we can find $w \in \mathcal{U}$ such that $\mathcal{U}, w \Vdash A$ and $\mathcal{U}, w \nVdash 0A$. We shall show that for this particular choice of w we have $\mathsf{Th}_{\mathcal{I}}(f) = \mathsf{Th}_{\mathcal{U}}(w)$.

First, suppose that $\mathcal{I}, f \Vdash B$ for some worm B. By the definability of f and the completeness of $\mathcal{I}$ (Theorem 2.6.8), we know that $\mathsf{GLP}^0 \vdash A \wedge [0]\neg A \rightarrow B$. By Lemma 3.6.5 and the conservativity of GLP^0 over WC, we may conclude that $A \vdash_{\mathsf{WC}} B$, and hence that $\mathcal{U}, w \Vdash B$.

For the opposite inclusion, suppose that $\mathcal{I}, f \nVdash B$ for some worm B. Then $\mathsf{GLP}^0 \nvdash A \rightarrow B$, which implies that $A \nvdash_{\mathsf{WC}} B$. Let C be a worm equivalent to $A \wedge B$, which exists by Lemma 2.7.3. Clearly, since $A \nvdash_{\mathsf{WC}} B$ and $A \nvdash_{\mathsf{RC}} \langle 0 \rangle (A \wedge B)$, we have that $A <_0^{\mathsf{RC}} C$ by the trichotomy of $<_0^{\mathsf{RC}}$, whence $C \vdash_{\mathsf{WC}} 0A$. We assume towards a contradiction that $\mathcal{U}, w \Vdash B$. In that case, since also $\mathcal{U}, w \Vdash A$, we may conclude by Lemma 3.6.7 that $\mathcal{U}, w \Vdash C$. But since $C \vdash_{\mathsf{WC}} 0A$, this would mean that $\mathcal{U}, w \Vdash 0A$, which is a contradiction by our choice of w. □

We conclude by observing that the results proven about (universal) poly-Euclidean models for WC also hold for (universal) poly-Euclidean models for RC^0. It remains to see whether the same results hold for non-poly-Euclidean models.

4

Quantified Reflection Calculus with one modality as a modal logic

Contents

4.1 Introduction

As described in Section 2.5.2, there are no nice provability logics in the full quantified modal language; more precisely, $\mathsf{QPL}(T)$ is Π^0_2-complete for a large class of theories of arithmetic T [203, 207].

In order to isolate a quantified provability logic, we took inspiration from the field of strictly positive logics and their tendency to have lower complexity than their full-language counterparts (see Section 2.7 and in particular Table 2.1). We chose the Reflection Calculus with one modality as a starting point due to its status as a strictly positive provability logic, and extended it to the quantified setting, obtaining the Quantified Reflection Calculus with one modality, denoted by QRC_1.

In this chapter, we describe QRC_1 from the modal point of view, exhibiting relational soundness and completeness proofs, and also a proof of decidability and an upper-bound for its complexity. We further show that QRC_1 is the strictly positive fragment of the logics between $\mathsf{QK4}$ and QGL (both inclusive), just as RC_1 is the strictly positive fragment of the logics between $\mathsf{K4}$ and GL (also both inclusive) [78]. These results are published in [13] and [14].

We tackle the arithmetical semantics in Chapter 5, where we prove that QRC_1 is the strictly positive fragment of $\mathsf{QPL}(T)$ when T is a computably enumerable (c.e.) and sound extension of EA. Thus, QRC_1 is indeed a quantified provability logic, albeit a strictly positive one, and we do obtain a substantial drop in complexity: from Π^0_2-complete to decidable.

Many of the results of this chapter have been formalized in Coq [5], and some are presented here in exceeding detail because detailed proofs are easier to formalize. We describe the formalization process in Chapter 10, where the main difficulties are described, as well as some new insights into both the relational soundness and completeness proofs.

4.2 Axiomatization

The Quantified Reflection Calculus with one modality, denoted by QRC_1, is a calculus for strictly positive quantified modal formulas.

Towards describing the language of QRC_1, denoted by $\mathcal{L}^+_{\Box\forall\mathsf{c}}$, we fix a countable set of variables $x_0, x_1, \ldots$ (also referred to as x, y, z, etc.) and define a signature Σ as a set of constants denoted by $\textsc{Const}_\Sigma$ and a set of relation symbols with corresponding arity denoted by $\textsc{Rel}_\Sigma$. We have no function symbols nor equality. We use the letters $c, c_i, \ldots$ to refer to constants and the letters $P, Q, S, S_i, \ldots$ to refer to relation symbols. The signature is often not mentioned explicitly. Note that this deviates slightly from the language $\mathcal{L}^+_{\Box\forall}$ described in Section 2.5 for quantified modal logics, since here constants are included. This decision, inspired by Goldblatt [110], is helpful for proving the relational completeness result, but the constant-free fragment of QRC_1 is equally expressive (Lemma 4.2.6; see also Section 4.5).

Given a signature, a term t is either a variable or a constant of that signature. Both $\top$ and any n-ary relation symbol applied to n terms are atomic formulas. The set of formulas is the closure of the atomic formulas under the binary connective $\wedge$, the unary modal operator $\Diamond$, and the quantifier $\forall x$, where x is a variable. Formulas are represented by Greek letters such as φ, ψ, χ, etc.

The axioms and rules of QRC_1 are listed in the following definition.

Definition 4.2.1 (QRC_1)**.** Let Σ be a signature and φ, ψ, and χ be any formulas in $\mathcal{L}^{+}_{\Box\forall\mathrm{c}}$ with the signature Σ. The axioms and rules of QRC_1 are the following:

(i) $\varphi \vdash \top$ and $\varphi \vdash \varphi$;

(ii) $\varphi \wedge \psi \vdash \varphi$ and $\varphi \wedge \psi \vdash \psi$ (conjunction elimination);

(iii) if $\varphi \vdash \psi$ and $\varphi \vdash \chi$, then $\varphi \vdash \psi \wedge \chi$ (conjunction introduction);

(iv) if $\varphi \vdash \psi$ and $\psi \vdash \chi$, then $\varphi \vdash \chi$ (cut);

(v) if $\varphi \vdash \psi$, then $\Diamond\varphi \vdash \Diamond\psi$ (necessitation);

(vi) $\Diamond\Diamond\varphi \vdash \Diamond\varphi$ (transitivity);

(vii) if $\varphi \vdash \psi$, then $\varphi \vdash \forall x\, \psi$, for $x \notin \mathsf{fv}(\varphi)$ (quantifier introduction on the right);

(viii) if $\varphi[x{\leftarrow}t] \vdash \psi$, then $\forall x\, \varphi \vdash \psi$, for t free for x in φ (quantifier introduction on the left);

(ix) if $\varphi \vdash \psi$, then $\varphi[x{\leftarrow}t] \vdash \psi[x{\leftarrow}t]$, for t free for x in φ and ψ (term instantiation);

(x) if $\varphi[x{\leftarrow}c] \vdash \psi[x{\leftarrow}c]$, then $\varphi \vdash \psi$, for c not in φ nor ψ (constant elimination).

If $\varphi \vdash \psi$, we say that ψ follows from φ in QRC_1. When the signature is not clear from the context, we write $\varphi \vdash_\Sigma \psi$ instead.

We briefly comment on the above axioms and rules. The first six statements correspond to axioms and rules of RC_1, while the two quantifier rules are standard in first-order logics. The final two rules, term instantiation and constant elimination, fulfill an essential role in the completeness of QRC_1. The best way to think of them is as quantifier rules in disguise. Since our semantics (described in Section 4.3) interprets the free variables on both sides of $\vdash$ in the same way, we can also think of such free variables as being generalized outside this implication. In other words, $P(x) \vdash Q(x)$ can be thought of as $\forall x\, (P(x) \vdash Q(x))$. We never explicitly write the latter, since it falls outside the scope of the strictly positive language. However, we do wish to arrive at some conclusions of such a formula, namely, we wish to be able to simultaneously instantiate x on both sides of $\vdash$ by any term (accomplished by Rule 4.2.1.(ix)) and to simultaneously "generalize" a given term as well (accomplished by Rule 4.2.1.(x)).

The following easy lemma collects several results provable in QRC_1, which should provide flavor for the kinds of things one can prove in this system.

Lemma 4.2.2. *The following are theorems (or derivable rules) of* QRC_1*:*

1. $\forall x\, \forall y\, \varphi \vdash \forall y\, \forall x\, \varphi$ *(quantifier commutativity);*

2. $\forall x\, \varphi \vdash \varphi[x{\leftarrow}t]$ *for* t *free for* x *in* φ *(instantiation);*

3. *$\forall x\, \varphi \vdash \forall y\, \varphi[x{\leftarrow}y]$ for y free for x in φ and $y \notin \mathsf{fv}(\varphi)$ (variable renaming);*

4. *if $\varphi \vdash \psi$, then $\varphi \vdash \psi[x{\leftarrow}t]$ for x not free in φ and t free for x in ψ;*

5. *if $\varphi \vdash \psi[x{\leftarrow}c]$, then $\varphi \vdash \forall x\, \psi$ for x not free in φ and c not in φ nor ψ;*

6. *$\Diamond \forall x\, \varphi \vdash \forall x\, \Diamond\varphi$.*

Proof.

1. Starting from $\varphi \vdash \varphi$, apply quantifier introduction on the left (Rule 4.2.1.(viii)) twice (noting that a variable is always free for itself), concluding $\forall x\, \forall y\, \varphi \vdash \varphi$. Now neither x or y is free in the left-hand-side, so use quantifier introduction on the right (Rule 4.2.1.(vii)) twice to obtain $\forall x\, \forall y\, \varphi \vdash \forall y\, \forall x\, \varphi$, as desired.

2. By quantifier introduction on the left applied to $\varphi[x{\leftarrow}t] \vdash \varphi[x{\leftarrow}t]$.

3. By quantifier introduction on the right applied to instantiation (the previous item).

4. Observe that, since x is not free in φ, we have $\varphi \vdash \forall x\, \psi$ by quantifier introduction on the right. We also have $\forall x\, \psi \vdash \psi[x{\leftarrow}t]$ by instantiation, which is enough through cut (Rule 4.2.1.(iv)).

5. This is a consequence of constant elimination (Rule 4.2.1.(x)) when x is not free in φ and quantifier introduction on the right.

6. Starting from $\forall x\, \varphi \vdash \varphi$ (instantiation), use necessitation (Rule 4.2.1.(v)) to obtain $\Diamond \forall x\, \varphi \vdash \Diamond\varphi$ and end with quantifier introduction on the right.

$\square$

In order to analyze various aspects of our calculus we define the following complexity measures on formulas.

Definition 4.2.3 ($\mathsf{d}, \mathsf{d}_\Diamond, \mathsf{d}_\forall$). Given a formula φ, its depth $\mathsf{d}(\varphi)$ is defined recursively as follows:

- $\mathsf{d}(\top) := \mathsf{d}(S(x_0, \dots, x_{n-1})) := 0;$
- $\mathsf{d}(\psi \wedge \chi) := \max\{\mathsf{d}(\psi), \mathsf{d}(\chi)\} + 1;$
- $\mathsf{d}(\forall x\, \psi) := \mathsf{d}(\psi) + 1;$
- $\mathsf{d}(\Diamond\psi) := \mathsf{d}(\psi) + 1.$

The modal depth $\mathsf{d}_\Diamond(\varphi)$ is defined similarly, except neither conjunctions nor quantifiers increase it:

- $\mathsf{d}_\Diamond(\top) := \mathsf{d}_\Diamond(S(x_0, \dots, x_{n-1})) := 0;$
- $\mathsf{d}_\Diamond(\psi \wedge \chi) := \max\{\mathsf{d}_\Diamond(\psi), \mathsf{d}_\Diamond(\chi)\};$
- $\mathsf{d}_\Diamond(\forall x\, \psi) := \mathsf{d}_\Diamond(\psi);$
- $\mathsf{d}_\Diamond(\Diamond\psi) := \mathsf{d}_\Diamond(\psi) + 1.$

The quantifier depth $\mathsf{d}_\forall(\varphi)$ is analogous except for:

- $\mathsf{d}_\forall(\forall x\,\psi) := \mathsf{d}_\forall(\psi) + 1$;
- $\mathsf{d}_\forall(\Diamond\psi) := \mathsf{d}_\forall(\psi)$.

Let $d \in \{\mathsf{d}, \mathsf{d}_\Diamond, \mathsf{d}_\forall\}$. Given a finite set of formulas Γ, we define $d(\Gamma) := \max_{\varphi\in\Gamma}\{d(\varphi)\}$.

The modal depth provides a necessary condition for derivability, proven by a straightforward induction on $\varphi \vdash \psi$.

Lemma 4.2.4. *If $\varphi \vdash \psi$, then $\mathsf{d}_\Diamond(\varphi) \geq \mathsf{d}_\Diamond(\psi)$.*

In particular, we get irreflexivity for free as stated in the next result. For other calculi this usually requires hard work via either modal or arithmetical semantics, as can be seen in [40, 96, 98].

Corollary 4.2.5. *For any formula φ, we have $\varphi \nvdash \Diamond\varphi$.*

The following lemma tells us that adding new constants to our signature (which *a priori* could have an empty set of constants) yields a conservative extension of the calculus.

Lemma 4.2.6. *Let Σ be a signature and let C be a collection of new constants not yet occurring in Σ. By Σ_C we denote the signature obtained by including these new constants C in Σ. Let φ, ψ be formulas in the language of Σ. Then, if $\varphi \vdash_{\Sigma_C} \psi$, so does $\varphi \vdash_\Sigma \psi$.*

Proof. This is a standard result and a proof for a calculus similar to ours can be found in Goldblatt [110, Section 1.8]. The idea is to replace every constant from C appearing in the proof of $\varphi \vdash_{\Sigma_C} \psi$ by a fresh variable. It can easily be seen that axioms are mapped to axioms under this replacement, and that the rules are also mapped correctly. The most interesting case is that of constant elimination (Rule 4.2.1.(x)), because replacing new constants by variables in the premise $\varphi[x{\leftarrow}c] \vdash_{\Sigma_C} \psi[x{\leftarrow}c]$ may leave us unable to apply the same rule. Fortunately term instantiation (Rule 4.2.1.(ix)) suffices to complete the proof. $\square$

We note that QRC_1 does not allow one to have spurious relation symbols on the right side of a sequent.

Lemma 4.2.7. *Let φ and ψ be QRC_1 formulas. If $\varphi \vdash \psi$, then all the relation symbols appearing in ψ also appear in φ.*

Proof. By an easy induction on $\varphi \vdash \psi$. $\square$

This gives us interpolation for free.

Corollary 4.2.8 (Interpolation)**.** *Let φ and ψ be QRC_1 formulas. If $\varphi \vdash \psi$, then there is a formula χ such that all the relation symbols appearing in χ also appear in both φ and ψ, and furthermore we have both $\varphi \vdash \chi$ and $\chi \vdash \psi$.*

Proof. By Lemma 4.2.7, $\chi := \psi$ is an adequate interpolant. $\square$

We can also deduce that new relation symbols can be conservatively added to a signature.

Corollary 4.2.9. *Let Σ be a signature and let R be a collection of new relation symbols not yet occurring in Σ. By Σ_R we denote the signature obtained by including these new relation symbols R in Σ. Let φ, ψ be formulas in the language of Σ. Then, if $\varphi \vdash_{\Sigma_R} \psi$, so does $\varphi \vdash_\Sigma \psi$.*

Proof. By induction on $\varphi \vdash_{\Sigma_R} \psi$, assuming $\varphi, \psi \in \Sigma$. The only non-trivial case is that of the cut rule. If $\varphi \vdash_{\Sigma_R} \psi$ via a cut, let $\chi \in \Sigma_R$ be the cut formula such that $\varphi \vdash_{\Sigma_R} \chi$ and $\chi \vdash_{\Sigma_R} \psi$. By Lemma 4.2.7, the relation symbols of χ must be present in φ. Thus, $\chi \in \Sigma$, so we can use the induction hypotheses to conclude. □

4.3 Relational soundness

In order to obtain relational semantics for QRC_1, we adapt the Kripke sheaf semantics for quantified modal logic described in Section 2.5.1. The main difference is that, since we include constants in the language of QRC_1, the models for QRC_1 should know how to interpret constants.

Definition 4.3.1 ((Adequate) relational model)**.** A relational model for the language of QRC_1 in a signature Σ is a tuple $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}, \{I_w\}_{w\in W}, \{J_w\}_{w\in W}\rangle$ where $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}\rangle$ is a Kripke sheaf and:

- for each $w \in W$, the interpretation I_w assigns an element of the domain M_w to each constant $c \in \Sigma$, written c^{I_w};
- for each $w \in W$, the interpretation J_w assigns a set of tuples $S^{J_w} \subseteq \wp((M_w)^n)$ to each n-ary relation symbol $S \in \textsc{Rel}_\Sigma$.

We say that a model is adequate if:

- R is transitive: if wRu and uRv, then wRv;
- I is concordant: if wRu, then $c^{I_u} = \eta_{w,u}(c^{I_w})$ for every constant $c \in \Sigma$.

We say that the model is finite (respectively, constant domain) if the sheaf is finite (respectively, constant domain).

Recall that a w-assignment g is a function assigning a member of the domain M_w to each variable in the language, and that we write g^u as shorthand for $\eta_{w,u} \circ g$. A w-assignment g is extended to terms by defining $g(c) := c^{I_w}$ for any constant c. Note that this meshes nicely with the concordance restriction of an adequate model: for any term t, if wRu then $g^u(t) = \eta_{w,u}(g(t))$.

We are now ready to define satisfiability at a world. This is essentially the same definition we have for the full quantified modal language (Definition 2.5.3), except now focused on the strictly positive connectives and taking into account that terms might be constants as well as variables.

Definition 4.3.2 (Satisfiability)**.** Let $\mathcal{M} = \langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{wRu}, \{I_w\}_{w\in W}, \{J_w\}_{w\in W}\rangle$ be an adequate model in some signature Σ, and let $w \in W$ be a world, g be a w-assignment, S be an n-ary relation symbol of Σ, and φ, ψ be formulas in the language of Σ.

We define $\mathcal{M}, w \Vdash^g \varphi$ (φ is true at w under g) recursively on φ as follows:

- $\mathcal{M}, w \Vdash^g \top$;
- $\mathcal{M}, w \Vdash^g S(t_0, \ldots, t_{n-1})$ if and only if $\langle g(t_0), \ldots, g(t_{n-1})\rangle \in S^{J_w}$;
- $\mathcal{M}, w \Vdash^g \varphi \wedge \psi$ if and only if both $\mathcal{M}, w \Vdash^g \varphi$ and $\mathcal{M}, w \Vdash^g \psi$;
- $\mathcal{M}, w \Vdash^g \Diamond\varphi$ if and only if there is a $u \in W$ such that wRu and $\mathcal{M}, u \Vdash^{g^u} \varphi$;
- $\mathcal{M}, w \Vdash^g \forall x\, \varphi$ if and only if for all w-assignments h such that $h \sim_x g$, we have $\mathcal{M}, w \Vdash^h \varphi$.

We now present a number of simple results needed to prove the relational soundness of QRC_1. These are standard observations about either first-order models or Kripke models that we adapted to our case.

Lemma 4.3.3 (Absent free variables)**.** *Let $\mathcal{M}$ be an adequate model, w be a world, g, h be Γ-alternative w-assignments, and φ be a formula with no free variables in Γ. Then:*

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}, w \Vdash^h \varphi.$$

Proof. We take the different assumption that g and h are $(\mathsf{Var} \setminus \mathsf{fv}(\varphi))$-alternative (i.e., they agree on the valuations of every free variable of φ) in order to obtain a better induction principle and proceed by induction on φ. Note that this assumption implies the above one and that there is no need to inspect both implications, since the lemma statement is symmetrical.

If φ is $\top$ or a conjunction, the result is straightforward.

If φ is a predicate symbol S applied to a tuple of terms $t_0, \ldots, t_{n-1}$, then $\mathcal{M}, w \Vdash^g S(t_0, \ldots, t_{n-1})$ if and only if $\langle g(t_0), \ldots, g(t_{n-1})\rangle \in S^{J_w}$. Since g and h agree on the value of all the t_i, this is also equivalent to $\langle h(t_0), \ldots, h(t_{n-1})\rangle \in S^{J_w}$, which happens if and only if $\mathcal{M}, w \Vdash^h S(t_0, \ldots, t_{n-1})$.

Let φ be $\Diamond\psi$ and assume $\mathcal{M}, w \Vdash^g \Diamond\psi$. Then there is a world u such that wRu and $\mathcal{M}, u \Vdash^{g^u} \psi$. In order to see that $\mathcal{M}, w \Vdash^h \Diamond\psi$, we can use u as a witnessing world as long as we can show $\mathcal{M}, u \Vdash^{h^u} \psi$. But this is clear by the induction hypothesis, taking into account that for any set Δ we have that $g \sim_\Delta h$ implies $g^u \sim_\Delta h^u$.

Suppose φ is $\forall x\, \psi$. Assume $\mathcal{M}, w \Vdash^g \forall x\, \psi$, and let f be an arbitrary w-assignment such that $f \sim_x h$. We want to check $\mathcal{M}, w \Vdash^f \psi$. Define a new assignment f' that coincides with g everywhere except at x, where $f'(x) := f(x)$. By our hypothesis that $\mathcal{M}, w \Vdash^g \forall x\, \psi$, we know that $\mathcal{M}, w \Vdash^{f'} \psi$. Then by the induction hypothesis it suffices to show that $f \sim_{\mathsf{Var} \setminus \mathsf{fv}(\psi)} f'$, i.e., that for every free variable y of ψ we have $f(y) = f'(y)$. If y is x, this is straightforward by the definition of f'. If not, note that y must be a free variable of $\forall x\, \psi$ as well. Since $f \sim_x h$, we know that $f(y) = h(y)$. Since $h \sim_{\mathsf{Var} \setminus \mathsf{fv}(\forall x\, \psi)} g$, we conclude $f(y) = g(y)$, as desired. □

Lemma 4.3.4 (Substitution in formula)**.** *Let $\mathcal{M}$ be an adequate model, w be a world, and $g, \tilde{g}$ be x-alternative w-assignments such that $\tilde{g}(x) = g(t)$. Then for every formula φ with t free for x:*

$$\mathcal{M}, w \Vdash^{\tilde{g}} \varphi \iff \mathcal{M}, w \Vdash^g \varphi[x \leftarrow t].$$

Proof. By induction on φ. We only present the cases of the diamond and of the universal quantifier; the remaining cases are straightforward. We assume that x is a free variable of φ, since otherwise we could use Lemma 4.3.3.

Suppose that φ is $\Diamond\psi$ and assume that $\mathcal{M}, w \Vdash^{\tilde{g}} \Diamond\psi$. Then there is a world u such that wRu and $\mathcal{M}, u \Vdash^{\tilde{g}^u} \psi$. Note that $g^u \sim_x \tilde{g}^u$ and $\tilde{g}^u(x) = g^u(t)$ (either t is a variable and this is a consequence of $\tilde{g}(x) = g(t)$, or t is a constant and this follows from $t^{I_u} = \eta_{w,u}(t^{I_w})$), so by the induction hypothesis $\mathcal{M}, u \Vdash^{g^u} \psi[x{\leftarrow}t]$. This gives us $\mathcal{M}, w \Vdash^{g} \Diamond\psi[x{\leftarrow}t]$, as desired. The other direction is analogous.

Suppose now that φ is $\forall z\, \psi$ and assume that $\mathcal{M}, w \Vdash^{\tilde{g}} \forall z\, \psi$. Note that x and z are different variables, for otherwise x would not be free in φ. Let h be any w-assignment such that $h \sim_z g$. We wish to show $\mathcal{M}, w \Vdash^{h} \psi[x{\leftarrow}t]$. Define $\tilde{h}$ such that $\tilde{h} \sim_x h$ and $\tilde{h}(x) := h(t)$. Then by the induction hypothesis we can reduce our goal to $\mathcal{M}, w \Vdash^{\tilde{h}} \psi$. By our assumption, it is enough to check that $\tilde{h} \sim_z \tilde{g}$.

In order to see this, note first that $\tilde{h} \sim_{\{x,z\}} h$ (because $\tilde{h} \sim_x h$). Similarly, $h \sim_{\{x,z\}} g$ and $g \sim_{\{x,z\}} \tilde{g}$. Then $\tilde{h} \sim_{\{x,z\}} \tilde{g}$ by the transitivity of $\sim_{\{x,z\}}$. But $\tilde{g}(x) = g(t)$ by assumption; $g(t) = h(t)$ because $g \sim_z h$ (z and t are not the same variable because otherwise t would not be free for x in φ); and $h(t) = \tilde{h}(x)$ by construction of $\tilde{h}$. Thus $\tilde{g}(x) = \tilde{h}(x)$, and $\tilde{h} \sim_z \tilde{g}$.

Towards the other direction, assume that $\mathcal{M}, w \Vdash^{g} (\forall z\, \psi)[x{\leftarrow}t]$ and that x and z are not the same variable. Let $\tilde{h} \sim_z \tilde{g}$ be a w-assignment. We wish to show $\mathcal{M}, w \Vdash^{\tilde{h}} \psi$. Define $h \sim_x \tilde{h}$ such that $h(x) := g(x)$. Note that $h \sim_z g$ by the transitivity of $\sim_{\{x,z\}}$ (using a similar argument to the one above). Thus we know that $\mathcal{M}, w \Vdash^{h} \psi[x{\leftarrow}t]$ by assumption. It only remains to show that $\tilde{h}(x) = h(t)$, as we can then finally use the induction hypothesis to finish. If t is x there is nothing to show, and t cannot be z, because z is not free for x in $\forall z\, \psi$ (recalling our assumption that x is a free variable of φ). Thus, $h(t) = g(t) = \tilde{g}(x) = \tilde{h}(x)$. $\square$

We now wish to provide counterparts to Lemmas 4.3.3 and 4.3.4 for when the change happens in the interpretation of a constant instead of in the interpretation of a variable. It is straightforward to check that the interpretation of constants not appearing in a formula is not relevant for the truth of that formula:

Lemma 4.3.5 (Absent constants). *Let $\mathcal{M}$ and $\mathcal{M}'$ be adequate models differing only in their constant interpretations $\{I_w\}_{w\in W}$ and $\{I'_w\}_{w\in W}$. Let w be any world, g be any w-assignment, and φ be a formula whose constants are interpreted in the same way by both $\mathcal{M}$ and $\mathcal{M}'$. Then:*

$$\mathcal{M}, w \Vdash^{g} \varphi \iff \mathcal{M}', w \Vdash^{g} \varphi.$$

Proof. The statement is symmetric, so we prove only one direction. The $\top$ and conjunction cases are straightforward.

In the case of relation symbols, we assume that φ is $S(x, c)$ for some variable x and constant c, with other arities and term arrangements being analogous. If $\mathcal{M}, w \Vdash^{g} S(x, c)$, then $\langle g(x), c^{I_w}\rangle \in S^{J_w}$. Since $c^{I_w} = c^{I'_w}$, we also have that $\mathcal{M}', w \Vdash S(x, c)$.

If φ is $\Diamond\psi$ and $\mathcal{M}, w \Vdash^{g} \Diamond\psi$, then there is u such that wRu and $\mathcal{M}, u \Vdash^{g^u} \psi$. Since the constants in ψ are the same as in $\Diamond\psi$, we can use the induction hypothesis to see that $\mathcal{M}', u \Vdash^{g^u} \psi$, and hence we have $\mathcal{M}', w \Vdash^g \Diamond\psi$.

Assume now that φ is $\forall z\, \psi$ and that $\mathcal{M}, w \Vdash^g \forall z\, \psi$. Let $h \sim_z g$ be a w-assignment, and set out to prove $\mathcal{M}', w \Vdash^h \psi$. By the induction hypothesis this is equivalent to $\mathcal{M}, w \Vdash^h \psi$, which follows from our assumption. □

However, we need a bit of work to be able to state a counterpart of Lemma 4.3.4 for constants. We want to be able to replace the interpretation of a constant by an element of the domain of some world w, but this element may not exist in the domains of the worlds below w. Thus we need to first get rid of that part of the model and keep only the sub-model rooted at w.

Definition 4.3.6 (Sheaf and model restricted to a world)**.** Given a Kripke sheaf $\mathcal{S} = \langle W, R, \{M_w\}_{w \in W}, \{\eta_{w,u}\}_{wRu}\rangle$ and a world $r \in W$, the sheaf restricted at r, written $\mathcal{S}|_r = \langle W|_r, R|_r, \{M_w\}_{w \in W|_r} \{\eta_{w,u}\}_{wR|_r u}\rangle$, is defined as the restriction of $\mathcal{S}$ to the world r and all the worlds accessible from r by R. Thus, $W|_r := \{r\} \cup \{w \in W \mid rRw\}$, and the relation $R|_r$ is R restricted to $W|_r$.

If $\mathcal{M} = \langle \mathcal{S}, \{I_w\}_{w \in W}, \{J_w\}_{w \in W}\rangle$ is a model, then $\mathcal{M}|_r$ is defined as $\langle \mathcal{S}|_r, \{I_w\}_{w \in W|_r}, \{J_w\}_{w \in W|_r}\rangle$.

Note that if $\mathcal{M}$ is an adequate model, then so is $\mathcal{M}|_r$, for any world r of $\mathcal{M}$. The following is a well-known observation, noting that the satisfiability of a formula at a world w depends only on w and on the sub-model accessible from w.

Remark 4.3.7. Given an adequate model $\mathcal{M}$ and a world $r \in \mathcal{M}$, we have that for any formula φ, any world $w \in \mathcal{M}|_r$ and any w-assignment g:

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}|_r, w \Vdash^g \varphi.$$

We are now ready to define what it means for the interpretation of a constant to be changed to a particular domain element.

Definition 4.3.8 (Model with c replaced by d)**.** Let $\mathcal{M} = \langle \mathcal{S}, \{I_w\}_{w \in W}, \{J_w\}_{w \in W}\rangle$ be an adequate model, $r \in W$ be a world, $c \in \textsc{Const}_\Sigma$ be a constant, and d be an element of the domain of r. We define:

$$\mathcal{M}|_r[c{\leftarrow}d] := \langle \mathcal{S}|_r, \{I'_w\}_{w \in W|_r}, \{J_w\}_{w \in W|_r}\rangle$$

such that its sheaf is $\mathcal{S}$ truncated at r, the relation symbols interpretation and the interpretation of all constants except for c coincides with that of $\mathcal{M}|_r$, the interpretation of c at the world r is d (i.e., $c^{I'_r} := d$) and the interpretation of c at any other world $w \in W|_r$ is $\eta_{r,w}(d)$.

Note that, since we assume $\mathcal{M}$ is adequate, then so is $\mathcal{M}|_r[c{\leftarrow}d]$.

Lemma 4.3.9. *Given a constant c, a formula φ where c does not appear, an adequate model $\mathcal{M}$, a world w, and a w-assignment g, we have:*

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}|_w[c{\leftarrow}g(x)], w \Vdash^g \varphi[x{\leftarrow}c].$$

Proof. We proceed by induction on the formula φ. The cases of $\top$, relation symbols, and conjunction are trivial. We assume that x is free in φ, for otherwise we could use Lemma 4.3.5 and Remark 4.3.7.

Consider the diamond case where φ is $\Diamond\psi$. If $\mathcal{M}, w \Vdash^g \Diamond\psi$, then there is a world u such that wRu and $\mathcal{M}, u \Vdash^{g^u} \psi$. By the induction hypothesis we obtain $\mathcal{M}|_u[c{\leftarrow}g^u(x)], u \Vdash^{g^u} \psi[x{\leftarrow}c]$. Observe that $\mathcal{M}|_u[c{\leftarrow}g^u(x)]$ is the same model as $(\mathcal{M}|_w[c{\leftarrow}g(x)])|_u$, since they share the same Kripke sheaf, the same constant interpretation (because interpreting c as $g(x)$ at world w implies interpreting c as $\eta_{w,u}(g(x))$ at world u, and this is precisely $g^u(x)$; recall that the η functions respect transitivity, so future worlds are not affected) and the same relation symbol interpretation. Then by Remark 4.3.7 we get $\mathcal{M}|_w[c{\leftarrow}g(x)], u \Vdash^{g^u} \psi[x{\leftarrow}c]$ and consequently $\mathcal{M}|_w[c{\leftarrow}g(x)], w \Vdash^g \Diamond\psi[x{\leftarrow}c]$, as desired. The other implication is analogous.

Finally, let φ be $\forall z\, \psi$ and assume that $\mathcal{M}, w \Vdash^g \forall z\, \psi$. Let $h \sim_z g$ be a w-assignment, and set out to prove $\mathcal{M}|_w[c{\leftarrow}g(x)], w \Vdash^h \psi[x{\leftarrow}c]$ (note that z and x are not the same variable for otherwise x would not be free in φ). Since $h \sim_z g$, we know that $g(x) = h(x)$, so by the induction hypothesis it is enough to show $\mathcal{M}, w \Vdash^h \psi$, which follows from our assumption. The other implication is analogous. □

We are finally ready to prove that QRC_1 is sound with respect to the relational semantics presented above.

Theorem 4.3.10 (Relational soundness). *If $\varphi \vdash \psi$, then for any adequate model $\mathcal{M}$, for any world $w \in W$, and for any w-assignment g:*

$$\mathcal{M}, w \Vdash^g \varphi \implies \mathcal{M}, w \Vdash^g \psi.$$

Proof. By induction on the proof of $\varphi \vdash \psi$.

In the case of the axioms $\varphi \vdash \top$ and $\varphi \vdash \varphi$, the result is clear, as it is for the conjunction elimination axioms. The conjunction introduction and cut rules follow easily from the definitions.

For the necessitation rule assume the result for $\varphi \vdash \psi$ and further assume that $\mathcal{M}, w \Vdash^g \Diamond\varphi$. Then there is a world u such that wRu and $\mathcal{M}, u \Vdash^{g^u} \varphi$. We wish to see $\mathcal{M}, w \Vdash^g \Diamond\psi$. Taking u as a suitable witness, our goal changes to $\mathcal{M}, u \Vdash^{g^u} \psi$. Thus the induction hypothesis for u and g^u suffices.

The transitivity axiom is $\Diamond\Diamond\varphi \vdash \Diamond\varphi$, so assume that $\mathcal{M}, w \Vdash^g \Diamond\Diamond\varphi$. Then there is a world u such that wRu and $\mathcal{M}, u \Vdash^{\eta_{w,u}\circ g} \Diamond\varphi$, and also a subsequent world v such that uRv and $\mathcal{M}, v \Vdash^{\eta_{u,v}\circ(\eta_{w,u}\circ g)} \varphi$. Since R is transitive, we know that wRv and thus that $\eta_{u,v} \circ (\eta_{w,u} \circ g)$ extensionally coincides with $\eta_{w,v} \circ g$. Then $\mathcal{M}, v \Vdash^{\eta_{w,v}\circ g} \psi$ by Lemma 4.3.3 (since the two assignments are extensionally equal, we have $\eta_{u,v} \circ (\eta_{w,u} \circ g) \sim_{\{\}} \eta_{w,v} \circ g$), and consequently $\mathcal{M}, w \Vdash^g \Diamond\varphi$, as desired.

For the quantifier introduction on the right rule, assume the result for $\varphi \vdash \psi$ towards showing the soundness of $\varphi \vdash \forall x\, \psi$ with $x \notin \mathsf{fv}(\varphi)$. Assume further that $\mathcal{M}, w \Vdash^g \varphi$. Let h be a w-assignment such that $h \sim_x g$. We wish to see that $\mathcal{M}, w \Vdash^h \psi$. Since x is not a free variable in φ, we know that $\mathcal{M}, w \Vdash^h \varphi$ by Lemma 4.3.3. The result follows from the induction hypothesis with the w-assignment h.

Consider now the quantifier introduction on the left rule. Assume the result for $\varphi[x{\leftarrow}t] \vdash \psi$ with t free for x in φ and assume further that $\mathcal{M}, w \Vdash^g \forall x\, \varphi$. Then for every w-assignment h such that $h \sim_x g$ we have $\mathcal{M}, w \Vdash^h \varphi$. Define $h \sim_x g$ such that $h(x) := g(t)$. We obtain $\mathcal{M}, w \Vdash^g \psi$ by the induction hypothesis and Lemma 4.3.4.

Finally, the term instantiation rule is sound by Lemma 4.3.4, and the constant elimination rule is sound by Lemma 4.3.9. □

We end this section with the observation that, even though the language of QRC_1 is quite restricted, even its diamond-free fragment requires counter-models with arbitrarily large domains. For example, the sequent:

$$\forall x, y\, S(x, x, y) \land \forall x, y\, S(x, y, x) \land \forall x, y\, S(y, x, x) \vdash \forall x, y, z\, S(x, y, z)$$

is unprovable in QRC_1, but satisfied by every world with at most two domain elements. This reasoning can be extended to any n: if S is an n-ary predicate symbol, let φ_n be the conjunction of the $n(n-1)/2$ formulas of the form $\forall x_0, \ldots, x_{n-2}\, S(\ldots, x_0, \ldots, x_0, \ldots)$, with x_0 appearing in every possible pair of positions and every other position filled by a unique variable. Then φ_n does not entail $\psi_n := \forall x_0, \ldots, x_{n-1}\, S(x_0, \ldots, x_{n-1})$ but any world with at most $n-1$ domain elements that satisfies φ_n must also satisfy ψ_n.

4.4 Relational completeness

We now wish to prove the relational completeness of QRC_1. For every underivable sequent we provide an irreflexive, rooted, finite, and constant domain adequate model that doesn't satisfy it.

Before starting the formal proof, we briefly describe the main idea. The term models we build are such that each world is a pair of sets of closed formulas $p = \langle p^+, p^- \rangle$. The first set, p^+, is the set of formulas that will be satisfied at that world, or the positive part. The second set, p^-, is the set of formulas that will not be satisfied at that world, or the negative part. All worlds must be well-formed with respect to some finite set of closed formulas Φ, which means that:

- p is closed: every formula in p^+ and every formula in p^- is closed;
- p is Φ-maximal: every formula of Φ is in either p^+ or p^- and there are no formulas in p but not in Φ;
- p is consistent: if $\delta \in p^-$ then $\bigwedge p^+ \nvdash \delta$; and
- p is fully-witnessed: if $\forall x\, \varphi \in p^-$ then there is a constant c such that $\varphi[x{\leftarrow}c] \in p^-$.

In that case we say that p is Φ-maximal consistent and fully witnessed, or Φ-MCW for short (the closeness condition is included in the concept, although in practice almost every formula in this section is closed).[1] If p and q are pairs, we write $p \subseteq q$ when $p^+ \subseteq q^+$ and $p^- \subseteq q^-$. Furthermore, if Γ is a set of formulas we write $p \subseteq \Gamma$ instead of $p^+ \cup p^- \subseteq \Gamma$.

We want to have Φ-MCW pairs where Φ is closed under subformulas. However, the naive subformulas of $\forall x\, \varphi$ might be open. In order to avoid that, we use the notion of closure with respect to a set of constants C, where $\varphi[x{\leftarrow}c]$ is a valid subformula of $\forall x\, \varphi$ as long as $c \in C$.

Definition 4.4.1 ($\mathcal{C}\ell_C$)**.** Given a set of constants C, the closure of a formula φ under C, written $\mathcal{C}\ell_C(\varphi)$, is defined recursively on the formula as such:

[1] A Φ-maximal consistent pair p is fully determined by p^+ and Φ, so we could have worked with the positive parts and their Φ-complements instead of with the positive and negative parts. We chose this presentation because our original completeness proof used different sets Φ for different pairs (see [13]).

- $\mathcal{C}\ell_C(\top) := \{\top\}$;
- $\mathcal{C}\ell_C(S(t_0, \ldots, t_{n-1})) := \{S(t_0, \ldots, t_{n-1})), \top\}$;
- $\mathcal{C}\ell_C(\varphi \wedge \psi) := \{\varphi \wedge \psi\} \cup \mathcal{C}\ell_C(\varphi) \cup \mathcal{C}\ell_C(\psi)$;
- $\mathcal{C}\ell_C(\Diamond\varphi) := \{\Diamond\varphi\} \cup \mathcal{C}\ell_C(\varphi)$;
- $\mathcal{C}\ell_C(\forall x\, \varphi) := \{\forall x\, \varphi\} \cup \bigcup_{c \in C} \mathcal{C}\ell_C(\varphi[x{\leftarrow}c])$.

The closure under C of a set of formulas Γ is the union of the closures under C of each of the formulas in Γ:

$$\mathcal{C}\ell_C(\Gamma) := \bigcup_{\gamma \in \Gamma} \mathcal{C}\ell_C(\gamma).$$

The closure of a pair p is defined as the closure of $p^+ \cup p^-$.

Note that the closure of a set of closed formulas is itself a set of closed formulas.

Going back to the overview of the completeness proof, suppose that $\varphi \not\vdash \psi$, (assuming for now that φ and ψ are closed). Defining $p := \langle\{\varphi\}, \{\psi\}\rangle$, the counter-model will be rooted on a $\mathcal{C}\ell_C(p)$-MCW extension of p, where C is a set of constants to determine later. The set of constants C will be used as the domain of the root. Note that p is already closed and consistent, so taking the step to maximality is as simple as deciding whether to add each $\chi \in \mathcal{C}\ell_C(p)$ to the positive or to the negative part of the root without ruining its consistency. The hard part is doing so in a way that guarantees that the resulting pair is fully witnessed.

In the usual Henkin construction this is traditionally accomplished in two steps: first extend the signature to include a constant for each existential statement and add every closed formula of the form $\exists x\, \varphi \to \varphi[x{\leftarrow}c_\varphi]$ to your set, proving that this didn't break consistency. Then prove a Lindenbaum lemma to the effect that consistent sets can be extended to maximal consistent sets. The resulting sets will be maximal, consistent, and fully witnessed. However we cannot do this because we cannot express implications. Thus if we were to add a witness for every existential formula in our original pair p (read: universal formula in p^-) and then use a Lindenbaum lemma to make it maximal, there could be new existential formulas without witnesses. We might have to iterate the process over and over again, or at least a proof of termination would be non-trivial.

We now observe that if $\bigwedge p^+ \not\vdash \forall x\, \chi$, then also $\bigwedge p^+ \not\vdash \chi[x{\leftarrow}c]$, as long as c does not appear in either p^+ or χ (Lemma 4.2.2.5). This suggests a way of sorting the formulas of $\mathcal{C}\ell_C(p)$ into positive and negative parts: mark a formula as positive if and only if it is a consequence of p^+. This guarantees that there are witnesses for the negative universal formulas as long as there are enough constants to go around. In the course of studying QRC_1, we obtained two different ways of making sure there are enough constants: one that leads to increasing domains (published first in [13]) and one that leads to a constant domain (published later in [14]). Here, we describe only the latter, since it is a stronger result. However, the two different proofs share many ideas.

We introduce a new measure of formula complexity.

Definition 4.4.2 ($\mathsf{d}_{\mathsf{const}}$)**.** The number of different constants in a given formula φ is represented by $\mathsf{d}_{\mathsf{const}}(\varphi)$. The maximum number of different constants per formula in a set of formulas Γ is defined as $\mathsf{d}_{\mathsf{const}}(\Gamma) := \max_{\varphi \in \Gamma}\{\mathsf{d}_{\mathsf{const}}(\varphi)\}$.

Note that $\mathsf{d}_{\mathsf{const}}(\Gamma)$ is *not* the number of different constants appearing in Γ, but the maximum number of different constants in any single formula of Γ. For example, $\mathsf{d}_{\mathsf{const}}(\{S(a,b), P(c)\}) = 2$.

We observe that the maximum number of distinct constants per formula in the closure under C of a set of formulas can be bounded by a number that does not depend on C.

Remark 4.4.3. For any formula φ, set of formulas Φ, and set of constants C:

- $\mathsf{d}_{\mathsf{const}}(\varphi) \le \mathsf{d}_{\mathsf{const}}(\varphi[x{\leftarrow}c]) \le \mathsf{d}_{\mathsf{const}}(\varphi) + 1$;
- $\mathsf{d}_{\mathsf{const}}(\varphi) \le \mathsf{d}_{\mathsf{const}}(\mathcal{C\ell}_C(\varphi)) \le \mathsf{d}_{\mathsf{const}}(\varphi) + \mathsf{d}_\forall(\varphi)$;
- $\mathsf{d}_{\mathsf{const}}(\Phi) \le \mathsf{d}_{\mathsf{const}}(\mathcal{C\ell}_C(\Phi)) \le \mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_\forall(\Phi)$.

We are now ready to prove a Lindenbaum-like lemma.

Lemma 4.4.4 (Lindenbaum)**.** *Given a finite signature Σ with constants C and a finite set of closed formulas Φ in the language of Σ such that $|C| > 2(\mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_\forall(\Phi))$, if $p \subseteq \mathcal{C\ell}_C(\Phi)$ is a closed consistent pair and p^+ is a singleton, then there is a pair $q \supseteq p$ in the language of Σ such that q is $\mathcal{C\ell}_C(\Phi)$-MCW, and $\mathsf{d}_\Diamond(q^+) = \mathsf{d}_\Diamond(p^+)$.*

Proof. We start by defining a pair $q \supseteq p$ such that for each $\chi \in \mathcal{C\ell}_C(\Phi)$, $\chi \in q^+$ if $p^+ \vdash \chi$ and $\chi \in q^-$ otherwise. It is easy to see that this pair q is $\mathcal{C\ell}_C(\Phi)$-maximal, and we have $\mathsf{d}_\Diamond(q^+) = \mathsf{d}_\Diamond(p^+)$ by Lemma 4.2.4.

Now assume by way of contradiction that q is not consistent, and let $\psi \in q^-$ be such that $\bigwedge q^+ \vdash \psi$. Note that for every $\chi \in q^+$ we know that $p^+ \vdash \chi$, because this was the required condition to add χ to q^+ in the first place. Thus, it must be that $p^+ \vdash \psi$. But then we would have placed ψ in q^+ instead of q^- and we reach a contradiction. We conclude that q is consistent.

It remains to show that q is fully witnessed. Let $\forall x\, \psi$ be a formula in q^-. We claim that there is $d \in C$ such that d does not appear either in p^+ or in $\forall x\, \psi$. For this it is enough to see that $|C| > \mathsf{d}_{\mathsf{const}}(\bigwedge p^+) + \mathsf{d}_{\mathsf{const}}(\forall x\, \psi)$, which is the same as $|C| > \mathsf{d}_{\mathsf{const}}(p^+) + \mathsf{d}_{\mathsf{const}}(\forall x\, \psi)$ (note the lack of a big conjunction behind p^+) because p^+ is a singleton by assumption. Since $p^+ \subseteq \mathcal{C\ell}_C(\Phi)$, we know that $\mathsf{d}_{\mathsf{const}}(p^+) \le \mathsf{d}_{\mathsf{const}}(\mathcal{C\ell}_C(\Phi))$, and similarly for $\forall x\, \psi$. Then by Remark 4.4.3 we may conclude that $\mathsf{d}_{\mathsf{const}}(p^+) + \mathsf{d}_{\mathsf{const}}(\forall x\, \psi) \le 2(\mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_\forall(\Phi))$, which suffices by our assumption on the size of C.

Since d does not appear in either p^+ or $\forall x\, \psi$, we can use Lemma 4.2.2.5 to conclude $p^+ \nvdash \psi[x{\leftarrow}d]$, and consequently $\psi[x{\leftarrow}d] \in q^-$, as desired. $\square$

The next step is to link maximal consistent and fully witnessed pairs through a relation that respects the diamond formulas in the pair. To that end we define $\hat{R}$ and prove some properties about it.

Definition 4.4.5 ($p\hat{R}q$)**.** The relation $\hat{R}$ between pairs is such that $p\hat{R}q$ if and only if both of following hold:

- for any formula $\Diamond\varphi \in p^-$ we have $\{\varphi, \Diamond\varphi\} \subseteq q^-$;
- there is some formula $\Diamond\psi \in p^+ \cap q^-$.

Lemma 4.4.6. *The relation $\hat{R}$ restricted to consistent pairs is transitive and irreflexive.*

Proof. In order to see that $\hat{R}$ is transitive, assume that $p\hat{R}q\hat{R}r$. We wish to see that $p\hat{R}r$. Let $\Diamond\varphi \in p^-$ be arbitrary. Then $\Diamond\varphi \in q^-$ because $p\hat{R}q$, and then $\{\varphi, \Diamond\varphi\} \subseteq r^-$ because $q\hat{R}r$. Let now $\Diamond\psi \in p^+ \cap q^-$. Since $q\hat{R}r$ we know that $\Diamond\psi \in r^-$. Then $\Diamond\psi \in p^+ \cap r^-$.

Regarding irreflexivity, suppose that there is a pair p such that $p\hat{R}p$. Then there must be $\Diamond\psi \in p^+ \cap p^-$, which contradicts the consistency of p. $\square$

For maximal and consistent pairs, we can replace the first condition on $\hat{R}$ with one that looks at the positive parts.

Lemma 4.4.7. *Given a set of formulas Γ and Γ-maximal consistent pairs p, q, we have that $p\hat{R}q$ if and only if both of the following hold:*

- *for every formula $\Diamond\varphi \in \Gamma$, if either $\varphi \in q^+$ or $\Diamond\varphi \in q^+$, then $\Diamond\varphi \in p^+$;*
- *there is some formula $\Diamond\psi \in p^+ \cap q^-$.*

Proof. Assume that $p\hat{R}q$ and let $\Diamond\varphi \in \Gamma$ be such that either $\varphi \in q^+$ or $\Diamond\varphi \in q^+$. Assume by contradiction that $\Diamond\varphi \notin p^+$. Then we know that $\Diamond\varphi \in p^-$ by maximality. Thus, since $p\hat{R}q$, we obtain both $\varphi \in q^-$ and $\Diamond\varphi \in q^-$. But this contradicts the consistency of q. The last condition holds by the definition of $\hat{R}$.

Assume now that these conditions hold, towards checking that $p\hat{R}q$. Only the first condition is in question. Let $\Diamond\varphi \in p^-$ and assume that $\varphi \notin q^-$. It must then be that $\varphi \in q^+$ by maximality. Then $\Diamond\varphi \in p^+$, which contradicts the consistency of p. Assume now that $\Diamond\varphi \notin q^-$. By the same token, $\Diamond\varphi$ must be in q^+. Then $\Diamond\varphi \in p^+$, reaching a contradiction again. $\square$

We now see that, if C is large enough and w is a $\mathcal{C}\ell_C(\Phi)$-MCW pair with $\Diamond\varphi \in w^+$, we can find a $\mathcal{C}\ell_C(\Phi)$-MCW pair u with $\varphi \in u^+$ and $w\hat{R}u$.

Lemma 4.4.8 (Pair existence)**.** *Let Σ be a signature with a finite set of constants C, and Φ be a finite set of closed formulas in the language of Σ such that $|C| > 2(\mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_\forall(\Phi))$. If p is a $\mathcal{C}\ell_C(\Phi)$-MCW pair and $\Diamond\varphi \in p^+$, then there is a $\mathcal{C}\ell_C(\Phi)$-MCW pair q such that $p\hat{R}q$, $\varphi \in q^+$, and $\mathsf{d}_\Diamond(q^+) < \mathsf{d}_\Diamond(p^+)$.*

Proof. Consider the pair r defined as $r^+ := \{\varphi\}$ and $r^- := \{\delta, \Diamond\delta \mid \Diamond\delta \in p^-\} \cup \{\Diamond\varphi\}$. Assume that r is not consistent, and thus that there is a formula $\psi \in r^-$ such that $\varphi \vdash \psi$. It cannot be that ψ is $\Diamond\varphi$ due to Corollary 4.2.5. Thus there is $\Diamond\delta \in p^-$ such that either $\varphi \vdash \delta$ or $\varphi \vdash \Diamond\delta$. By necessitation we get either $\Diamond\varphi \vdash \Diamond\delta$ or $\Diamond\varphi \vdash \Diamond\Diamond\delta$, which also implies $\Diamond\varphi \vdash \Diamond\delta$ by transitivity. This contradicts the consistency of p, which leads us to conclude that r is consistent.

It is clear that $p\hat{R}q$ by the definition of r: for every $\Diamond\delta \in p^-$, the formulas δ and $\Diamond\delta$ are in r^- (and hence in q^-), and the formula $\Diamond\varphi$ is both in p^+ and in q^-.

We then use Lemma 4.4.4 to obtain a $\mathcal{C}\ell_C(\Phi)$-MCW pair $q \supseteq r$ such that $\mathsf{d}_\Diamond(q^+) = \mathsf{d}_\Diamond(r^+) = \mathsf{d}_\Diamond(\varphi) < \mathsf{d}_\Diamond(p^+)$. We obtain $p\hat{R}q$ as a straightforward consequence of $p\hat{R}r$. $\square$

We can now define an adequate and constant domain model $\mathcal{M}[p]$ from any given finite and consistent pair p such that $\mathcal{M}[p]$ satisfies the formulas in p^+ and doesn't satisfy the formulas in p^-. The idea is to build a term model where each world w is a $\mathcal{C\ell}_M(p)$-MCW pair (where M is the model's domain), and the worlds are related by (a sub-relation of) $\hat{R}$.

Definition 4.4.9. Let Σ be a signature. Given a finite consistent pair p of closed formulas in Σ such that p^+ is a singleton (or, if p^+ is not a singleton, take the conjunction of every formula in p^+ and use that as p^+ instead), we define an adequate model $\mathcal{M}[p]$. Here we use $\Phi := p^+ \cup p^-$.

Let C be a set of at least $2(\mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_\forall(\Phi)) + 1$ different constants, including all of the ones appearing in Σ and adding more if necessary. The pairs of formulas we work with are in the signature Σ extended by C.

We start by defining the underlying Kripke sheaf in an iterative manner. Each of the following sheaves has a constant domain C and identity compatibility functions.

- The root is given by Lemma 4.4.4 applied to C and p, obtaining the $\mathcal{C\ell}_C(\Phi)$-MCW pair q. The sheaf $\mathcal{S}^0$ is then defined such that its set of worlds is $W^0 := \{q\}$ and its relation R^0 is empty.
- Assume now that we already have a sheaf $\mathcal{S}^i$, and we set out to define $\mathcal{S}^{i+1}$ as an extension of $\mathcal{S}^i$. For each leaf w of $\mathcal{S}^i$, i.e., each world w such that there is no world $u \in \mathcal{S}^i$ with wR^iu, and for each formula $\Diamond\varphi \in w^+$, use Lemma 4.4.8 to obtain a $\mathcal{C\ell}_C(\Phi)$-MCW pair u such that $w\hat{R}u$, $\varphi \in u^+$, and $\mathsf{d}_\Diamond(u^+) < \mathsf{d}_\Diamond(w^+)$. Now add u to W^{i+1} and add $\langle w, u\rangle$ to R^{i+1}.

The process described above terminates because each pair is finite and the modal depth of $\mathcal{C\ell}_C(\Phi)$ (and consequently of w^+, for any $w \subseteq \mathcal{C\ell}_C(\Phi)$) is also finite. Thus there is a final sheaf $\mathcal{S}^m$, for some natural number m. This sheaf is constant domain by construction, but not transitive. We obtain $\mathcal{S}[p]$ as the transitive closure of $\mathcal{S}^m$, which is clearly still constant domain.

In order to obtain the model $\mathcal{M}[p]$ based on the sheaf $\mathcal{S}[p]$, let I_q take constants in Σ to their corresponding version as domain elements. If w is any other world, let I_w coincide with I_q. This is necessary to make sure that the model is concordant, because q is related to every other world, and it is sufficient to see that $\mathcal{M}[p]$ is adequate. Finally, given an n-ary predicate letter S and a world w, define S^{J_w} as the set of n-tuples $\langle d_0, \ldots, d_{n-1}\rangle \subseteq C^n$ such that $S(d_0, \ldots, d_{n-1}) \in w^+$.

We now collect some straightforward consequences of the definition of $\mathcal{M}[p]$.

Lemma 4.4.10. *Let p be as above. The following are properties of the $\mathcal{S}[p] = \langle W, R, C, \{\eta_{w,u}\}_{wRu}\rangle$ sheaf and the $\mathcal{M}[p] = \langle \mathcal{S}[p], \{I_w\}_{w\in W}, \{J_w\}_{w\in W}\rangle$ model:*

1. *The set of worlds W is finite.*
2. *The model is rooted.*
3. *The domain C is the same for every world.*
4. *The domain C is finite.*
5. *The compatibility function $\eta_{w,u}$ is the identity for any pair of worlds w, u such that wRu.*

6. *Every world $w \in W$ is closed, $\mathcal{C}\ell_C(p)$-maximal consistent, and fully witnessed.*

7. *For every world $w \in W$, we have $\top \in w^+$.*

8. *For any two worlds $w, u \in W$, if wRu, then $w\hat{R}u$.*

Proof. These are simple consequences of the definition of $\mathcal{M}[p]$. The finiteness of the domain is possible due to Lemma 4.4.4, while the finiteness of W comes directly from Definition 4.4.9. We have $\top \in w^+$ for every world $w \in W$ due to the definition of closure, noting that C is non-empty (this is needed because $\top \notin \mathcal{C}\ell_{\{\}}(\forall x\, \varphi)$). For the last property, note that R is the transitive closure of R^m. If $wR^m u$, then $w\hat{R}u$ by construction. The result then follows by the transitivity of $\hat{R}$ (Lemma 4.4.6). $\square$

Since everything up until now was meant for closed formulas, and furthermore we are potentially adding new constants to the signature of the formulas we care about, we provide a way of replacing the free variables of a formula with constants.

Definition 4.4.11 (φ^g)**.** Given a formula φ in a signature Σ and a function g from the set of variables to a set of constants in some signature Σ' that has the same relation symbols as Σ and such that $\textsc{Const}_{\Sigma'} \supseteq \textsc{Const}_{\Sigma}$, we define the formula φ^g in the signature Σ' as φ with each free variable x simultaneously replaced by $g(x)$.

Given a variable x, the formula $\varphi^{g\setminus x}$ is as above, except that the variable x is not replaced even if it is free in φ.

We are now ready to prove the Truth lemma, which roughly states that the formulas in the positive part of a given world in $\mathcal{M}[p]$ are precisely those satisfied at that world.

Lemma 4.4.12 (Truth)**.** *Let Σ be a signature. For any finite non-empty consistent pair p of closed formulas in the language of Σ, world $w \in \mathcal{M}[p]$, w-assignment g, and formula φ in the language of Σ such that $\varphi^g \in \mathcal{C}\ell_C(p)$ (where C is the domain of $\mathcal{M}[p]$), we have that:*

$$\mathcal{M}[p], w \Vdash^g \varphi \iff \varphi^g \in w^+.$$

Proof. By induction on φ. The cases of $\top$ and conjunction are straightforward, so we focus on the other ones.

In the case of the relation symbols, take φ as $S(x, c)$ where $c \in C$ as a simplification, since other arities and arrangements of terms are analogous. Note that $\mathcal{M}[p], w \Vdash^g S(x, c)$ if and only if $\langle g(x), c^{I_w} \rangle \in S^{J_w}$ if and only if $S(g(x), c^{I_w}) \in w^+$. Since $c \in C$, we know by the definition of $\mathcal{M}[p]$ that $c^{I_w} = c$. Thus, we conclude that $\mathcal{M}[p], w \Vdash^g S(x, c)$ if and only if $S(g(x), c) \in w^+$, as desired.

Consider now the case of the universal quantifier. For the left to right implication, suppose that $\mathcal{M}[p], w \Vdash^g \forall x\, \varphi$. Then for every w-assignment $h \sim_x g$ we have $\mathcal{M}[p], w \Vdash^h \varphi$. Thus for each such h we know that $\varphi^h \in w^+$ by the induction hypothesis ($\varphi^h \in \mathcal{C}\ell_C(p)$ because $(\forall x\, \varphi)^g \in \mathcal{C}\ell_C(p)$). We want to show that $(\forall x\, \varphi)^g \in w^+$, i.e., that $\forall x\, \varphi^{g\setminus x} \in w^+$. Assume by contradiction that this is not the case. Then, since w is $\mathcal{C}\ell_C(p)$-maximal consistent, it must be that $\forall x\, \varphi^{g\setminus x} \in w^-$. Let $c \in C$ be a witness such that $\varphi^{g\setminus x}[x\leftarrow c] \in w^-$, which exists because w is fully witnessed. Let h be the w-assignment that coincides with g everywhere except at x, where $h(x) := c$. Then $g \sim_x h$ and $\varphi^{g\backslash x}[x\leftarrow c] = \varphi^h$. But this contradicts our earlier observation that for every such h the formula φ^h is in w^+.

For the right to left implication, let $\forall x\, \varphi^{g\backslash x} \in w^+$, and let $h \sim_x g$ be any w-assignment. We want to show that $\mathcal{M}[p], w \Vdash^h \varphi$. By the induction hypothesis this is the same as showing that $\varphi^h \in w^+$. But $\varphi^h = \varphi^{g\backslash x}[x\leftarrow h(x)]$, and this is in w^+ by the completeness and consistency of w.

Finally, consider the case of the diamond. For the left to right implication, assume $\mathcal{M}[p], w \Vdash^g \Diamond\varphi$. Then there is some world u such that wRu and $\mathcal{M}[p], u \Vdash^{g^u} \varphi$. By the induction hypothesis we obtain $\varphi^{g^u} \in u^+$, and since $\eta_{w,u}$ is the identity function, also $\varphi^g \in u^+$. Now, since wRu, we know that $w\hat{R}u$ by Lemma 4.4.10.8, and thus by Lemma 4.4.7 we obtain $\Diamond\varphi^g \in w^+$, as desired.

For the right to left implication, assume that $(\Diamond\varphi)^g \in w^+$. By the construction of $\mathcal{M}[p]$, there is a world u such that $\varphi^g \in u^+$ (and hence $\varphi^{g^u} \in u^+$) and wRu, and then $\mathcal{M}[p], u \Vdash^{g^u} \varphi$ by the induction hypothesis, from which we finally conclude $\mathcal{M}[p], w \Vdash^g \Diamond\varphi$. $\square$

We can finally prove the constant domain completeness theorem.

Theorem 4.4.13 (Constant domain completeness). *Let Σ be a signature and φ, ψ formulas in Σ. If $\varphi \nvdash \psi$, then there is an adequate, finite, irreflexive, rooted, and constant domain model $\mathcal{M}$, a world $w \in W$, and a w-assignment g such that:*

$$\mathcal{M}, w \Vdash^g \varphi \quad \textit{and} \quad \mathcal{M}, w \nVdash^g \psi.$$

Proof. Define a new constant c_x for each free variable x of φ and ψ and let Σ' be the signature Σ augmented with these new constants and a dummy constant c_0. Let g be the assignment taking each free variable x of φ and ψ to c_x and every other variable to c_0. Note that $\varphi^g \nvdash_{\Sigma'} \psi^g$ by the constant elimination rule and Lemma 4.2.6. Build $\mathcal{M} := \mathcal{M}[\langle\{\varphi^g\}, \{\psi^g\}\rangle]$ as described in Definition 4.4.9, with root w. Then by Lemma 4.4.12 we have both $\mathcal{M}, w \Vdash^g \varphi$ and $\mathcal{M}, w \nVdash^g \psi$, as desired. $\square$

We observe that QRC_1 has the finite model property with respect to domains and number of worlds (as a consequence of Lemma 4.4.10 and Theorem 4.4.13). It is interesting that QRC_1 is this well behaved while, say, predicate intuitionistic logic doesn't enjoy the finite model property with respect to either of these.

Corollary 4.4.14. QRC_1 *is decidable.*

Proof. Since QRC_1 is recursively axiomatized, has the finite model property, and it is easy to check whether a given finite model is adequate, Post's theorem [184] allows us to decide whether $\varphi \vdash \psi$. See Section 10.7 for a more detailed proof. $\square$

We briefly comment on the complexity of QRC_1. We take the maximum number of nested connectives (the depth, denoted by d) of a formula as the relevant complexity measure. We have both $\mathsf{d}_\Diamond(\varphi) \leq \mathsf{d}(\varphi)$ and $\mathsf{d}_\forall(\varphi) \leq \mathsf{d}(\varphi)$. We do not take $\mathsf{d}_{\mathsf{const}}(\varphi)$ into account.

Given two closed formulas[2] φ and ψ in a signature Σ where we want to decide whether $\varphi \vdash \psi$, let $p := \langle\{\varphi\}, \{\psi\}\rangle$ and $\Phi := p^+ \cup p^-$. By soundness and completeness, we know that $\varphi \vdash \psi$ if and only

[2]We assume the formulas are closed without loss of generality, since the trick of the completeness proof can be used to get rid of any free variables.

if $\mathcal{M}[p], w \Vdash^g \varphi$ implies $\mathcal{M}[p], w \Vdash^g \psi$, where w and g are as in the completeness theorem. Using big $\mathcal{O}$ notation (see [73]), we can bound the size of $\mathcal{M}[p]$ as follows:

- the domain M of $\mathcal{M}[p]$ has size at most $\max\{2(\mathsf{d}_{\mathsf{const}}(\Phi) + \mathsf{d}_{\forall}(\Phi)) + 1, |\textsc{Const}_\Sigma|\}$, which is $\mathcal{O}(\mathsf{d}(\Phi))$ because we are disregarding the complexity arising from $\mathsf{d}_{\mathsf{const}}$ and hence $|\textsc{Const}_\Sigma|$ as well;
- for any formula χ and set of constants C, we have that $|\mathcal{C}\ell_C(\chi)|$ is $\mathcal{O}(|C|^{\mathsf{d}(\chi)})$, so since $\mathcal{C}\ell_M(\Phi) = \mathcal{C}\ell_M(\varphi) \cup \mathcal{C}\ell_M(\psi)$, we have that $|\mathcal{C}\ell_M(\Phi)|$ is $\mathcal{O}(\mathsf{d}(\Phi)^{\mathsf{d}(\Phi)})$;
- the number of worlds of $\mathcal{M}[p]$ is at most the number of subsets of $\mathcal{C}\ell_M(\Phi)$, so the number of worlds is $\mathcal{O}\left(2^{\mathsf{d}(\Phi)^{\mathsf{d}(\Phi)}}\right)$;
- the relation R can be described using $\mathcal{O}\left(\left(2^{\mathsf{d}(\Phi)^{\mathsf{d}(\Phi)}}\right)^2\right)$ symbols.

Thus the current approximation for the space needed to describe $\mathcal{M}[p]$ is double exponential on the depth of p. By enumerating all models with size at most the size of $\mathcal{M}[p]$ and checking whether they validate φ and/or ψ, we can determine whether $\varphi \vdash \psi$. This gives us an upper-bound of double exponential space for the complexity of QRC_1. A more dedicated study might lead to a tighter bound.

4.5 Fragment of QK4 and QGL

Dashkov [78] showed that RC_1 is the strictly positive fragment of both $\mathsf{K4}$ and GL (and of any logic between them), meaning that, if φ and ψ are built up from $\top$, propositional symbols, conjunctions, and diamonds, then $\varphi \vdash_{\mathsf{RC}_1} \psi$ if and only if $\mathsf{K4} \vdash \varphi \to \psi$, and similarly for GL. In this section we show that this equivalence also holds in the quantified case.

Since QRC_1 includes constants in its language and quantified modal logics are often presented without them (a decision we repeated in Section 2.5), we define a map from the language of QRC_1 to its constant-free fragment (i.e., a map from $\mathcal{L}^+_{\Box\forall\mathsf{c}}$ to $\mathcal{L}^+_{\Box\forall}$), replacing constants with fresh variables. Thus, given a finite set of QRC_1 formulas Φ where the different constants appearing in Φ are $\boldsymbol{c} = c_0, \ldots, c_{n-1}$, let $\boldsymbol{x} = x_0, \ldots, x_{n-1}$ be fresh variables with respect to Φ. Then we define $\varphi^\Phi := \varphi[c_0 \leftarrow x_0] \cdots [c_{n-1} \leftarrow x_{n-1}]$ for each $\varphi \in \Phi$, and abbreviate this long substitution with the notation $\varphi[\boldsymbol{c} \leftarrow \boldsymbol{x}]$. Thus we also have $\varphi = \varphi^\Phi[\boldsymbol{x} \leftarrow \boldsymbol{c}]$.

We confirm that the above translation is harmless when it comes to provability.

Lemma 4.5.1. *Let Φ be a finite set of QRC_1 formulas such that $\varphi, \psi \in \Phi$. Then:*

$$\varphi \vdash \psi \iff \varphi^\Phi \vdash \psi^\Phi.$$

Proof. If $\varphi \vdash \psi$, then $\varphi^\Phi[\boldsymbol{x} \leftarrow \boldsymbol{c}] \vdash \psi^\Phi[\boldsymbol{x} \leftarrow \boldsymbol{c}]$, and since no constant appears in either φ^Φ nor ψ^Φ and the constants c_i are all different, we can iterate the constant elimination rule to obtain $\varphi^\Phi \vdash \psi^\Phi$. If, on the other hand, we have $\varphi^\Phi \vdash \psi^\Phi$, then we can repeatedly use the term instantiation rule to obtain $\varphi^\Phi[\boldsymbol{x} \leftarrow \boldsymbol{c}] \vdash \psi^\Phi[\boldsymbol{x} \leftarrow \boldsymbol{c}]$, which is $\varphi \vdash \psi$. $\Box$

Recall that QK4 is K4 adapted to the quantified modal language together with $\forall$**I** and $\forall$**E**, and similarly for QGL (see Section 2.5). We are now ready to prove the desired result of this section.

Theorem 4.5.2. *Let φ and ψ be* QRC$_1$ *formulas and let QL be any logic in the quantified modal language between* QK4 *and* QGL.[3] *Then $\varphi \vdash_{\mathsf{QRC}_1} \psi$ if and only if there is a finite set Φ such that $\varphi, \psi \in \Phi$ and $QL \vdash \varphi^\Phi \to \psi^\Phi$.*

Proof. For the left-to-right implication, let Φ be the set of formulas appearing in the proof of $\varphi \vdash_{\mathsf{QRC}_1} \psi$. We check by induction on the length of the QRC$_1$ proof that QK4 $\vdash \varphi^\Phi \to \psi^\Phi$ for this particular Φ, which then implies that $QL \vdash \varphi^\Phi \to \psi^\Phi$.

The axioms $\varphi \vdash \top$, $\varphi \vdash \varphi$, and $\varphi \wedge \psi \vdash \varphi$ and the conjunction introduction and cut rules are straightforward.

For the QRC$_1$ necessitation rule, assume that $\varphi \vdash \psi$, which by the induction hypothesis gives us QK4 $\vdash \varphi^\Phi \to \psi^\Phi$. Then by taking the contrapositive and then applying the QK4 necessitation rule, we obtain QK4 $\vdash \Box(\neg\psi^\Phi \to \neg\varphi^\Phi)$. This implies QK4 $\vdash \Box\neg\psi^\Phi \to \Box\neg\varphi^\Phi$, and taking the contrapositive again gives us QK4 $\vdash \Diamond\varphi^\Phi \to \Diamond\psi^\Phi$, as desired.

The translation of the QRC$_1$ transitivity axiom, $\Diamond\Diamond\varphi^\Phi \to \Diamond\varphi^\Phi$, is just the contrapositive of an instance of **4**.

We turn to the quantifier introduction on the right rule. If $\varphi \vdash \psi$ and $x \notin \mathsf{fv}(\varphi)$, then by the induction hypothesis QK4 $\vdash \varphi^\Phi \to \psi^\Phi$ and we can be assured that $x \notin \mathsf{fv}(\varphi^\Phi)$ because all the new variables introduced by the $\cdot^\Phi$ translation are fresh with respect to $\forall x\, \psi$ and are consequently different from x. Then QK4 $\vdash \varphi^\Phi \to \forall x\, \psi^\Phi$ by $\forall$**I**.

In the case of the quantifier introduction on the left rule, suppose that $\varphi[x{\leftarrow}t] \vdash \psi$, with t free for x in φ. Defining t^Φ as either t itself (when t is a variable), or whichever variable replaces t by the $\cdot^\Phi$ translation (when t is a constant), observe that $(\varphi[x{\leftarrow}t])^\Phi$ is $\varphi^\Phi[x{\leftarrow}t^\Phi]$. Thus we have that QK4 $\vdash \varphi^\Phi[x{\leftarrow}t^\Phi] \to \psi^\Phi$ by the induction hypothesis, and we wish to show QK4 $\vdash \forall x\, \varphi^\Phi \to \psi^\Phi$. We know QK4 $\vdash \forall x\, \varphi^\Phi \to \varphi^\Phi[x{\leftarrow}t^\Phi]$ by $\forall$**E**, and so we are done.

For the term instantiation rule, suppose that $\varphi \vdash \psi$ and let x and t be such that t is free for x in both φ and ψ. By the induction hypothesis we have established QK4 $\vdash \varphi^\Phi \to \psi^\Phi$. Using $\forall$**I** with $\top$ as the antecedent, we obtain QK4 $\vdash \forall x\, (\varphi^\Phi \to \psi^\Phi)$, and then we may conclude QK4 $\vdash \varphi^\Phi[x{\leftarrow}t^\Phi] \to \psi^\Phi[x{\leftarrow}t^\Phi]$ by $\forall$**E**, as desired.

Finally, in the case of the constant elimination rule, if $\varphi[x{\leftarrow}c] \vdash \psi[x{\leftarrow}c]$, then QK4 $\vdash \varphi^\Phi[x{\leftarrow}c^\Phi] \to \psi^\Phi[x{\leftarrow}c^\Phi]$ by the induction hypothesis. Since c^Φ is a variable, we can generalize it through $\forall$**I**, obtaining QK4 $\vdash \forall c^\Phi\, (\varphi^\Phi[x{\leftarrow}c^\Phi] \to \psi^\Phi[x{\leftarrow}c^\Phi])$. We then use $\forall$**E** to instantiate c^Φ with x, obtaining QK4 $\vdash \varphi^\Phi[x{\leftarrow}c^\Phi][c^\Phi{\leftarrow}x] \to \psi^\Phi[x{\leftarrow}c^\Phi][c^\Phi{\leftarrow}x]$. Since c does not appear in either φ nor ψ, we also know that c^Φ does not appear in either φ^Φ nor ψ^Φ, and so $\varphi^\Phi[x{\leftarrow}c^\Phi][c^\Phi{\leftarrow}x] = \varphi^\Phi$, and similarly for ψ. We conclude that QK4 $\vdash \varphi^\Phi \to \psi^\Phi$, as desired.

For the right-to-left implication, we prove the contrapositive. If $\varphi \nvdash \psi$, then by Lemma 4.5.1 we know that $\varphi^\Phi \nvdash \psi^\Phi$ for any Φ including both φ and ψ. Let $\mathcal{M}$ be a QRC$_1$ model satisfying φ^Φ and not

[3]It suffices that QL be some set of formulas such that QK4 $\vdash \varphi$ implies $\varphi \in QL$ and $\varphi \in QL$ implies QGL $\vdash \varphi$. We write $QL \vdash \varphi$ instead of $\varphi \in QL$.

satisfying ψ^Φ, as given by Theorem 4.4.13. This model is based on a transitive and Noetherian frame (the latter because it is irreflexive and has finitely-many worlds), which implies that it (or rather, its translation to a model for a language without constants in the sense of Definition 2.5.2) is a QGL model by Theorem 2.5.4. We conclude that QGL $\nvdash \varphi^\Phi \to \psi^\Phi$, and consequently that $QL \nvdash \varphi^\Phi \to \psi^\Phi$. $\square$

We end with the following observation. The proof of Theorem 4.5.2 holds for any quantified modal logic that extends QK4 and is sound for any combination of constant-domain, finite, transitive and irreflexive models. The Barcan formula **BF** is always sound in constant-domain models. Thus the proof of Theorem 4.5.2 also serves to see that, for example, $\varphi \vdash_{\mathsf{QRC}_1} \psi$ if and only if there is a finite set Φ such that $\varphi, \psi \in \Phi$ and QK4 + **BF** $\vdash \varphi^\Phi \to \psi^\Phi$. This is curious because **BF** is not provable in QGL and is in fact unsound with the usual provability interpretation of $\Box$. From this we conclude that it would be worthwhile to study QRC$_1$ from points of view unrelated to provability.

5 Quantified Reflection Calculus with one modality as a provability logic

Contents

5.1 Introduction

Having studied the modal properties of QRC_1 in Chapter 4, we now turn to its arithmetical properties. The original goal of QRC_1 was to escape Vardanyan's theorem [203] that $\mathsf{QPL}(\mathsf{PA})$ is Π^0_2-complete, as described in Section 2.5.2. This cannot literally be done, but we show in this chapter that the strictly positive fragment of $\mathsf{QPL}(T)$ is exactly described by the theorems of QRC_1 for a large class of theories T. This gives QRC_1 its status as a provability logic, and furthermore is a significant reduction in complexity from Π^0_2-completeness, since QRC_1 is decidable (Corollary 4.4.14).

We also describe an alternative, infinitary arithmetical semantics for QRC_1, inspired by the usual arithmetical semantics for strictly positive logics described in Section 2.7.2. We show that QRC_1 is sound and complete with respect to this infinitary semantics, as well as for the usual finitary one.

Finally, we briefly delve into intuitionistic arithmetic and show that all the theorems of QRC_1 are always provable in HA, although we leave completeness as future work.

The results of this chapter are published in [14].

5.2 Arithmetical soundness

As mentioned in Section 2.7.2, we can provide both finitary and infinitary arithmetical semantics for strictly positive logics. We use $\cdot^*$ to refer to finitary realizations and $\cdot^\circ$ to refer to infinitary ones.

5.2.1 Finitary

The finitary semantics for QRC_1 are a straightforward adaptation of Definition 5.2.1, with only one addition: since there are constants in the language of QRC_1, we require that each constant c_k always be translated to the arithmetical variable z_k.

Definition 5.2.1 (Finitary arithmetical realization for $\mathcal{L}^+_{\Box\forall\mathsf{c}}$, $\cdot^*$, $\cdot^{*_T}$). A finitary arithmetical realization is a function $\cdot^*$ such that, for a given predicate S and terms $\boldsymbol{t}$, $S(\boldsymbol{t})^*$ is an arithmetical formula with the same arity as S where modal variables x_k are interpreted as y_k and modal constants c_k are interpreted as z_k.

Given a computably enumerable (c.e.) arithmetical theory T extending EA, we extend this realization to QRC_1 formulas as follows:

- $\top^{*_T} := \top$;
- $S(\boldsymbol{t})^{*_T} := S(\boldsymbol{t})^*$;
- $(\varphi \wedge \psi)^{*_T} := \varphi^{*_T} \wedge \psi^{*_T}$;
- $(\Diamond\varphi)^{*_T} := \Diamond_T \varphi^{*_T}$;
- $(\forall x_k\, \varphi)^{*_T} := \forall y_k\, \varphi^{*_T}$.

Remark 5.2.2. Given any realization $\cdot^*$ and formula φ in the language of QRC_1, we have that x_i is a free variable of φ if and only if y_i is a free variable of φ^{*_T}. Similarly, the constant c_i appears in φ if and only if z_i is a free variable of φ^{*_T}.

Recall the definition of quantified provability logic (Definition 2.5.6), where T is a c.e. extension of EA:

$$\mathsf{QPL}(T) := \{\varphi \in \mathcal{L}_{\Box\forall} \mid \text{for any } \cdot^*, \text{ we have } T \vdash \forall \boldsymbol{y}\, \varphi^{*_T}\}.$$

The strictly positive fragment of $\mathsf{QPL}(T)$ is defined as follows:

$$\mathsf{QPL}^+(T) := \{\langle \varphi, \psi \rangle \in \mathcal{L}^+_{\Box\forall\mathsf{c}} \times \mathcal{L}^+_{\Box\forall\mathsf{c}} \mid \text{for any } \cdot^*, \text{ we have } T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T})\},$$

where $\forall \boldsymbol{y}, \boldsymbol{z}$ is meant to quantify over all free variables of $\varphi^{*_T} \to \psi^{*_T}$. Clearly, if φ and ψ are formulas in the language of QRC_1, we have that $\langle \varphi, \psi \rangle \in \mathsf{QPL}^+(T)$ if and only if $\langle \varphi^\Phi, \psi^\Phi \rangle \in \mathsf{QPL}^+(T)$ if and only if $\varphi^\Phi \to \psi^\Phi \in \mathsf{QPL}(T)$, where Φ is any finite set of formulas including both φ and ψ and $\cdot^\Phi$ is the translation from $\mathcal{L}^+_{\Box\forall\mathsf{c}}$ to $\mathcal{L}^+_{\Box\forall}$ described in Section 4.5.

We can now prove the finitary soundness of QRC_1.

Theorem 5.2.3 (Finitary arithmetical soundness)**.** *Let T be a c.e. theory of arithmetic extending* EA *and φ and ψ be formulas of* QRC_1*. Then:*

$$\varphi \vdash \psi \implies \langle \varphi, \psi \rangle \in \mathsf{QPL}^+(T).$$

Proof. By induction on the QRC_1 proof of $\varphi \vdash \psi$.

Consider $\varphi \vdash \top$ with $\boldsymbol{x}, \boldsymbol{c}$ the free variables and constants appearing in φ, respectively. Let $\cdot^*$ be any realization. We wish to show that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \top)$, which is clearly the case. We are equally easily convinced that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \varphi^{*_T})$, that $T \vdash \forall \boldsymbol{y}, \boldsymbol{y'}, \boldsymbol{z}, \boldsymbol{z'}\, (\varphi^{*_T} \wedge \psi^{*_T} \to \varphi^{*_T})$, and that $T \vdash \forall \boldsymbol{y}, \boldsymbol{y'}, \boldsymbol{z}, \boldsymbol{z'}\, (\varphi^{*_T} \wedge \psi^{*_T} \to \psi^{*_T})$ (where $\boldsymbol{x'}$ and $\boldsymbol{c'}$ are the variables and constants of ψ not already present in φ).

Let $\boldsymbol{x}$ and $\boldsymbol{c}$ be the constants appearing in either φ, ψ, or χ.

The conjunction introduction rule states that from $\varphi \vdash \psi$ and $\varphi \vdash \chi$ we can obtain $\varphi \vdash \psi \wedge \chi$. Let $\cdot^*$ be any realization. We wish to prove $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T} \wedge \chi^{*_T})$. By the induction hypotheses we know both $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T})$ and $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \chi^{*_T})$, which easily allows us to prove the desired goal.

The cut rule states that if $\varphi \vdash \psi$ and $\psi \vdash \chi$, then $\varphi \vdash \chi$. It is enough to show $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \chi^{*_T})$ for an arbitrary realization $\cdot^*$. By the induction hypotheses, we know both $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T})$ and $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\psi^{*_T} \to \chi^{*_T})$. The result follows handily.

The necessitation rule states that if $\varphi \vdash \psi$, then $\Diamond\varphi \vdash \Diamond\psi$. We work towards showing that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\Diamond_T \varphi^{*_T} \to \Diamond_T \psi^{*_T})$. By the induction hypothesis, we know that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T})$. Taking this under the box (see Lemma 2.3.2), we obtain $T \vdash \Box_T \forall \boldsymbol{y}, \boldsymbol{z}\, (\varphi^{*_T} \to \psi^{*_T})$. This implies that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, \Box_T (\varphi^{*_T} \to \psi^{*_T})$ by the formalized converse Barcan formula, which in turn implies our desired goal.

Consider now the transitivity axiom $\Diamond\Diamond\varphi \to \Diamond\varphi$. We wish to show $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}\, (\Diamond_T \Diamond_T \varphi^{*_T} \to \Diamond_T \varphi^{*_T})$. This is derivable from the contrapositive of one of the derivability conditions (Lemma 2.3.2).

We turn to the quantifier introduction on the right rule, that if $\varphi \vdash \psi$ then $\varphi \vdash \forall x' \psi$, as long as $x' \notin \mathsf{fv}(\varphi)$. Let $\boldsymbol{x}$ be the free variables appearing in either φ or $\forall x' \psi$. We wish to show that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z} (\varphi^{*_T} \to \forall y' \psi^{*_T})$. By the induction hypothesis, $T \vdash \forall \boldsymbol{y}, y', \boldsymbol{z} (\varphi^{*_T} \to \psi^{*_T})$. The goal follows from the observation that $y' \notin \mathsf{fv}(\varphi^{*_T})$.

For the quantifier introduction on the left rule, that if $\varphi[x'{\leftarrow}t] \vdash \psi$ then $\forall x' \varphi \vdash \psi$ (t free for x' in φ), we want to show $T \vdash \forall \boldsymbol{y}, \boldsymbol{z} (\forall y' \varphi^{*_T} \to \psi^{*_T})$. Let w be the arithmetical variable corresponding to t. Then $(\varphi[x'{\leftarrow}t])^{*_T}$ is $\varphi^{*_T}[y'{\leftarrow}w]$ and the induction hypothesis tells us that $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}, w (\varphi^{*_T}[y'{\leftarrow}w] \to \psi^{*_T})$, which readily implies the desired result.

For the term instantiation rule, that if $\varphi \vdash \psi$ then $\varphi[x'{\leftarrow}t] \vdash \psi[x'{\leftarrow}t]$, let w be the arithmetical variable corresponding to t. Our goal is $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}, w (\varphi^{*_T}[y'{\leftarrow}w] \to \psi^{*_T}[y'{\leftarrow}w])$. The induction hypothesis is $T \vdash \forall \boldsymbol{y}, y', \boldsymbol{z} (\varphi^{*_T} \to \psi^{*_T})$, so this is a simple matter of variable renaming.

Finally, consider the constant elimination rule, that if $\varphi[x'{\leftarrow}c'] \vdash \psi[x'{\leftarrow}c']$ then $\varphi \vdash \psi$, with c' not appearing in either φ nor ψ. We wish to show $T \vdash \forall \boldsymbol{y}, y', \boldsymbol{z} (\varphi^{*_T} \to \psi^{*_T})$. We know $T \vdash \forall \boldsymbol{y}, \boldsymbol{z}, z' (\varphi^{*_T}[y'{\leftarrow}z'] \to \psi^{*_T}[y'{\leftarrow}z'])$ by the induction hypothesis. The result follows by renaming z' to y' in the induction hypothesis, noting that z' does not appear in either φ^{*_T} nor ψ^{*_T}. $\square$

5.2.2 Infinitary

We now turn to the infinitary semantics. As in Definition 2.7.5, we interpret strictly positive modal formulas as parameterized axiomatizations of arithmetical theories extending EA.

Definition 5.2.4 (Infinitary arithmetical realization for $\mathcal{L}^{+}_{\Box\forall\mathsf{c}}$, $\cdot^{\circ}$, $\cdot^{\circ_\tau}$)**.** An infinitary realization $\cdot^{\circ}$ is a function mapping each n-ary relation symbol $S(\boldsymbol{x}, \boldsymbol{c})$ to an $(n+1)$-ary Σ^0_1 formula $\sigma(u, \boldsymbol{y}, \boldsymbol{z})$ in the language of arithmetic, where u is a fresh variable, $\boldsymbol{y}$ matches with $\boldsymbol{x}$ and $\boldsymbol{z}$ matches with $\boldsymbol{c}$.

Given a Σ^0_1 axiomatization τ of an extension of EA, we extend $\cdot^{\circ}$ to QRC_1 formulas as follows:

- $\top^{\circ_\tau} := \tau$;
- $S(\boldsymbol{x}, \boldsymbol{c})^{\circ_\tau} := \tau \vee S(\boldsymbol{x}, \boldsymbol{c})^{\circ}$;
- $(\psi \wedge \delta)^{\circ_\tau} := \psi^{\circ_\tau} \vee \delta^{\circ_\tau}$;
- $(\Diamond\psi)^{\circ_\tau} := \tau \vee (u = \ulcorner \Diamond_{\psi^{\circ_\tau}} \top \urcorner)$;
- $(\forall x_i \psi)^{\circ_\tau} := \exists y_i \psi^{\circ_\tau}$.

Remark 5.2.5. Given any realization $\cdot^{\circ}$ and formula φ in the language of QRC_1, x_i is a free variable of φ if and only if y_i is a free variable of φ°_τ}. Similarly, the constant c_i appears in φ if and only if z_i is a free variable of φ°_τ}. Finally, u is always a free variable of φ°_τ}.

As an example, if $S(x, c)^{\circ}$ is $\sigma(u, y, z)$, then $(\Diamond S(x, c))^{\circ_\tau}$ is $\tau(u) \vee (u = \ulcorner \Diamond_{\tau(u) \vee \sigma(u, \dot{y}, \dot{z})} \top \urcorner)$. Recall from Section 2.3 that the dotted variables $\dot{y}$ and $\dot{z}$ in the expression $u = \ulcorner \Diamond_{\tau(u) \vee \sigma(u, \dot{y}, \dot{z})} \top \urcorner$ indicate that y and z are free variables of this expression that, upon being instantiated by natural numbers n and m, shall be replaced by the numerals $\overline{n}$ and $\overline{m}$ instead.

The interpretation of the universal quantifier is an existential quantifier, by analogy with the interpretation of conjunction being a disjunction. Thinking of each $\psi(x)^{\circ_\tau}$ as the axiomatization of a theory and of $(\forall x\, \psi(x))^{\circ_\tau}$ as the axiomatization of the infinite union of those theories for each x, it makes sense that its axiomatization is the infinite disjunction of $\psi(x)^{\circ_\tau}$ for each x, i.e., $\exists y\, \psi^{\circ_\tau}(y)$.

Remark 5.2.6. If φ is a QRC_1 formula and τ is a Σ^0_1 axiomatization of an extension of EA, then φ°_τ} is equivalent to a Σ^0_1 formula, provably in EA.

We deliberately defined $\cdot^{\circ_\tau}$ so that φ°_τ} leads to an extension of the base theory T for every formula φ, as the following lemma shows.

Lemma 5.2.7. *Given a* QRC_1 *formula* φ *and a* Σ^0_1 *axiomatization* τ*, we have that* φ°_τ} *is an axiomatization of an extension of the theory axiomatized by* τ*, provably in* EA.

Proof. We prove that for every modal formula φ there is an arithmetical formula χ with the same free variables as φ°_τ} such that $\mathsf{EA} \vdash \varphi^{\circ_\tau} \leftrightarrow (\tau \vee \chi)$.

We proceed by (external) induction on φ. The base cases of $\top$ and relation symbols are straightforward, as is the $\Diamond$ case. The conjunction case follows easily by the induction hypotheses, so we focus on the quantifier case. We wish to show that there is some formula χ with the same free variables as $(\forall x_i\, \varphi)^{\circ_\tau}$ such that $\mathsf{EA} \vdash (\forall x_i\, \varphi)^{\circ_\tau} \leftrightarrow (\tau \vee \chi)$. By the induction hypothesis, there is some formula δ with the same free variables as φ°_τ} such that $\mathsf{EA} \vdash \varphi^{\circ_\tau} \leftrightarrow (\tau \vee \delta)$, so $\mathsf{EA} \vdash (\forall x_i\, \varphi)^{\circ_\tau} \leftrightarrow \exists y_i\, (\tau \vee \delta)$. Since y_i is not a free variable of τ, we conclude that $\chi := \exists y_i\, \delta$ is a suitable formula, which in fact has the same free variables as $(\forall x_i\, \varphi)^{\circ_\tau}$. $\square$

Given a finitary realization $\cdot^*$, we can interpret it as an infinitary one by taking $S(\boldsymbol{t})^{\circledast} := (u = \ulcorner S(\boldsymbol{t})^* \urcorner)$, where the $\cdot^{\circledast}$ symbol is meant to allude to the fact that this is an infinitary realization (it is a circle) but comes from a finitary one (it has a star inside). The above proof might tempt us to conjecture that $\varphi^{\circledast_\tau}$ is equivalent to $\tau \vee (u = \ulcorner \varphi^{*_T} \urcorner)$. However, such a result is in general false: the best we can do is prove one of the implications.

Lemma 5.2.8. *If* T *is a c.e. extension of* EA *axiomatized by* τ*,* φ *is a formula in the language of* QRC_1 *and* $\cdot^{\circledast}$ *is the infinitary interpretation of* $\cdot^*$ *as defined above, then:*

$$T \vdash \forall \theta\, \forall \boldsymbol{y}, \boldsymbol{z}\, (\Box_{\varphi^{\circledast_\tau}} \theta \to \Box_\tau (\varphi^{*_T} \to \theta)),$$

where θ *is a closed formula and* $\boldsymbol{y}, \boldsymbol{z}$ *are the free variables of* φ^{*_T} *(which are the same as the free variables of* $\varphi^{\circledast_\tau}$*).*

Proof. By external induction on φ. There is nothing to show for $\top$ and the case of relation symbols is a straightforward consequence of the formalized deduction theorem (Theorem 2.3.1).

In the case of $\wedge$, we take $\varphi = \psi \wedge \delta$, and omit the variables $\boldsymbol{y}$ and $\boldsymbol{z}$, as they introduce visual clutter but don't make the proof any more complex. Reason in T and fix an arbitrary θ, assuming $\Box_{\psi^{\circledast_\tau} \vee \delta^{\circledast_\tau}} \theta$. Then there is a finite sequence $\boldsymbol{\pi} = \pi_0, \ldots, \pi_n$ with $\pi_n = \theta$ that is a proof of θ in the theory axiomatized by $\psi^{\circledast_\tau} \vee \delta^{\circledast_\tau}$. Each formula χ_i occurring in $\boldsymbol{\pi}$ that is not a consequence of previous formulas in the sequence through a rule satisfies either $\psi^{\circledast_\tau}$ or $\delta^{\circledast_\tau}$. Then we have in particular that either $\Box_{\psi^{\circledast_\tau}}\chi_i$ or $\Box_{\delta^{\circledast_\tau}}\chi_i$ for each such χ_i, whence by the induction hypotheses either $\Box_\tau(\psi^{*_T} \to \chi_i)$ or $\Box_\tau(\delta^{*_T} \to \chi_i)$. In both cases we have $\Box_\tau(\psi^{*_T} \wedge \delta^{*_T} \to \chi_i)$, and thus the proof of θ can be repeated in T under the assumption of $(\psi \wedge \delta)^{*_T}$, as desired.

The $\forall$ case follows the same idea as the $\wedge$ case. Consider $\varphi = \forall x_0\, \psi$, with $\mathsf{fv}(\psi) = \{x_0, x_1\}$ for simplicity (adding more free variables leads to an analogous proof). Note that x_0 is represented by y_0 in T, and this is always a different variable from any z used to represent QRC_1 constants. As there is no further complication with constants, we omit them in our proof. Reason in T, and let θ be (the Gödel number of) a closed formula and l be an arbitrary number taking the place of y_1. If $\Box_{\exists\, y_0\, \psi^{\circledast_\tau}[y_1 \leftarrow \bar{l}]}\theta$, then there is a proof $\boldsymbol{\pi} = \pi_0, \ldots, \pi_n$ where $\pi_n = \theta$ and each axiom χ_i in $\boldsymbol{\pi}$ satisfies $\psi^{\circledast_\tau}[y_1 \leftarrow l][y_0 \leftarrow k_i]$ for some number k_i, and consequently $\Box_\tau(\psi^{*_T}[y_1 \leftarrow l][y_0 \leftarrow k_i] \to \chi_i)$ by the induction hypothesis for each i. Then by weakening we conclude $\Box_\tau(\forall\, y_0\, \psi^{*_T}[y_1 \leftarrow l] \to \chi_i)$ for each i, and we are done.

Finally, for the case of $\varphi = \Diamond\psi$, we start by observing that the induction hypothesis with $\theta := \bot$ yields $T \vdash \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Diamond_\tau \psi^{*_T} \to \Diamond_{\psi^{\circledast_\tau}} \top)$. Note that $(\Diamond\psi)^{\circledast_\tau} := \tau \vee (u = \ulcorner \Diamond_{\psi^{\circledast_\tau}} \top \urcorner)$, and thus $\Box_{(\Diamond\psi)^{\circledast_\tau}}\theta$ implies $\Box_\tau(\Diamond_{\psi^{\circledast_\tau}} \top \to \theta)$ by the formalized deduction theorem. The previous observation under the box then suffices to finish the proof. $\Box$

We do some musing on the other implication, i.e., on $T \vdash \forall \theta\, \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Box_\tau(\varphi^{*_T} \to \theta) \to \Box_{\varphi^{\circledast_\tau}}\theta)$. Clearly it would follow from $T \vdash \forall\, \boldsymbol{y}, \boldsymbol{z}\, \Box_{\varphi^{\circledast_\tau}} \varphi^{*_T}$, which seems at first glance like a true statement. However, note that in the quantifier case it states that there is a *finite* proof of $\forall\, y\, \psi^{*_T}$ in the theory axiomatized by $\exists\, y\, \psi^{\circledast_\tau}$. Such a proof cannot rely on different arguments for each of the infinitely-many values available for y, there must instead be some pattern in ψ^{*_T} that can be exploited. This is the case for the realization we define in the next section, and in that case we can indeed prove the equivalency (Lemma 5.3.6).

Since we interpret formulas as axiomatizations, the statement $\varphi \vdash \psi$ is to be read as "the theory axiomatized by φ is an extension of the theory axiomatized by ψ", or equivalently, "if the theory axiomatized by ψ proves some formula θ, then so does the theory axiomatized by φ". We can then define the quantified reflection logic of a c.e. theory T extending EA. Let τ be an elementary axiomatization of T. Then:

$$\mathsf{QRL}(T) := \{\langle \varphi, \psi \rangle \in \mathcal{L}^{+}_{\Box\forall\mathsf{c}} \times \mathcal{L}^{+}_{\Box\forall\mathsf{c}} \mid \text{for any } \cdot^\circ, \text{ we have } T \vdash \forall \theta\, \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Box_{\psi^{\circ_\tau}}\theta \to \Box_{\varphi^{\circ_\tau}}\theta)\},$$

where θ is a closed formula and $\varphi^{\circ_\tau}, \psi^{\circ_\tau}$ in general depend on $\boldsymbol{y}$ and $\boldsymbol{z}$.

We finally prove the infinitary soundness of QRC_1 for any c.e theory T extending $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$. We are not aware of a proof of this result for theories where Σ^0_1 collection is not derivable, although rather weak theories such as $\mathrm{I}\Sigma_1$ already suffice (see Lemma 2.2.4).

Theorem 5.2.9 (Infinitary arithmetical soundness)**.** *Let T be a c.e. theory of arithmetic extending* $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$ *and φ and ψ be formulas of* QRC_1*. Then:*

$$\varphi \vdash \psi \implies \langle \varphi, \psi \rangle \in \mathsf{QRL}(T).$$

Proof. By induction on the QRC_1 proof of $\varphi \vdash \psi$. The case of the axiom $\varphi \vdash \top$ is a consequence of Lemma 5.2.7, while axiom $\varphi \vdash \varphi$ and the cut rule are both trivially sound. The conjunction elimination axioms are easily seen to be sound since $(\varphi \wedge \psi)^{\circ_\tau}$ is $\varphi^{\circ_\tau} \vee \psi^{\circ_\tau}$, that is, the formula that defines the union of two axiom sets.

We proceed with the soundness of the conjunction introduction rule, that if $\varphi \vdash \psi$ and $\varphi \vdash \chi$ then $\varphi \vdash \psi \wedge \chi$. Fix a realization $\cdot^\circ$, reason in T and let θ be a closed formula and $\boldsymbol{y}, \boldsymbol{z}$ be arbitrary. If $\Box_{\psi^{\circ_\tau} \vee \chi^{\circ_\tau}} \theta$, there is a proof $\boldsymbol{\pi} = \pi_0, \ldots, \pi_n$ in the sense of $\psi^{\circ_\tau} \vee \chi^{\circ_\tau}$ with $\pi_n = \theta$. Some of the π_i are axioms of ψ°_τ}, some are axioms of χ°_τ}, and some follow from previous steps in the proof through a rule. Let $\{\delta_i\}_{i<I}$ be the finite set of ψ°_τ} axioms appearing in $\boldsymbol{\pi}$. Then $\Box_{\psi^{\circ_\tau}} \forall i < I\, \delta_i$ and by the induction hypothesis for $\varphi \vdash \psi$ we obtain $\Box_{\varphi^{\circ_\tau}} \forall i < I\, \delta_i$. On the other hand, we know $\Box_{\chi^{\circ_\tau}} (\forall i < I\, \delta_i \to \theta)$ by the deduction theorem. Through the induction hypothesis for $\varphi \vdash \chi$ we conclude $\Box_{\varphi^{\circ_\tau}} (\forall i < I\, \delta_i \to \theta)$. Putting these two observations together, we obtain the desired $\Box_{\varphi^{\circ_\tau}} \theta$.

We turn to the necessitation rule, that if $\varphi \vdash \psi$ then $\Diamond\varphi \vdash \Diamond\psi$. Reason in T and let $\theta, \boldsymbol{y}$, and $\boldsymbol{z}$ be arbitrary. Consider the induction hypothesis with $\theta := \neg\top$ (and $\boldsymbol{y}$, $\boldsymbol{z}$ as given by our assumption): $\Box_{\psi^{\circ_\tau}} \neg\top \to \Box_{\varphi^{\circ_\tau}} \neg\top$. Taking the contrapositive, we conclude $\Diamond_{\varphi^{\circ_\tau}} \top \to \Diamond_{\psi^{\circ_\tau}} \top$, and consequently $\Box_\tau (\Diamond_{\varphi^{\circ_\tau}} \top \to \Diamond_{\psi^{\circ_\tau}} \top)$ by Lemma 2.3.2. Assume now that $\Box_{(\Diamond\psi)^{\circ_\tau}} \theta$. By the deduction theorem, we obtain $\Box_\tau (\Diamond_{\psi^{\circ_\tau}} \top \to \theta)$. Thus our previous observation gives us $\Box_\tau (\Diamond_{\varphi^{\circ_\tau}} \top \to \theta)$, which is equivalent to $\Box_{(\Diamond\varphi)^{\circ_\tau}} \theta$ by the deduction theorem again.

Consider the transitivity axiom: $\Diamond\Diamond\varphi \vdash \Diamond\varphi$. We start by observing that $(\Diamond\Diamond\varphi)^{\circ_\tau}$ is equivalent to $\tau \vee (u = \ulcorner \Diamond_\tau \Diamond_{\varphi^{\circ_\tau}} \top \urcorner)$. Note that we can derive $\Box_\tau (\Diamond_\tau \Diamond_{\varphi^{\circ_\tau}} \top \to \Diamond_{\varphi^{\circ_\tau}} \top)$ from Σ^0_1 completeness (Lemma 2.3.2). Assume now $\Box_{(\Diamond\varphi)^{\circ_\tau}} \theta$. By the deduction theorem we have $\Box_\tau (\Diamond_{\varphi^{\circ_\tau}} \top \to \theta)$. Then we obtain $\Box_\tau (\Diamond_\tau \Diamond_{\varphi^{\circ_\tau}} \top \to \theta)$ by our previous observation, and we finish with one more application of the deduction theorem.

The quantifier introduction on the right rule states that if $x_0 \notin \mathsf{fv}(\varphi)$ and $\varphi \vdash \psi$, then $\varphi \vdash \forall x_0\, \psi$. Reason in T and let $\theta, \boldsymbol{y}$, and $\boldsymbol{z}$ be arbitrary, where y_0 does not appear in $\boldsymbol{y}$. Assume $\Box_{\exists\, y_0\, \psi^{\circ_\tau}} \theta$ and let $\boldsymbol{\pi} = \pi_0, \ldots \pi_n$ be a proof of this fact. Let $\{\delta_i\}_{i<I}$ be the finite set of axioms of $\exists\, y_0\, \psi^{\circ_\tau}$ appearing in $\boldsymbol{\pi}$. Note that $\Box_\tau (\forall i < I\, \delta_i \to \theta)$ holds by the deduction theorem. Furthermore, for each $i < I$ there is k_i such that δ_i is an axiom of $\psi^{\circ_\tau}[y_0 {\leftarrow} k_i]$, so in particular $\Box_{\psi^{\circ_\tau}[y_0 \leftarrow \overline{k_i}]} \delta_i$. We can use the induction hypothesis for each $i < I$ to conclude $\Box_{\varphi^{\circ_\tau}[y_0 \leftarrow \overline{k_i}]} \delta_i$, and since $x_0 \notin \mathsf{fv}(\varphi)$, we also know that $y_0 \notin \mathsf{fv}(\varphi^{\circ_\tau})$, and thus we obtain $\forall i < I\, \Box_{\varphi^{\circ_\tau}} \delta_i$. We now use Σ^0_1 collection to obtain $\Box_{\varphi^{\circ_\tau}} \forall i < I\, \delta_i$, and the result follows from our observation that $\Box_\tau (\forall i < I\, \delta_i \to \theta)$, noting that the theory axiomatized by φ°_τ} extends T.

The quantifier introduction on the left rule states that if $\varphi[x_0 {\leftarrow} t] \vdash \psi$, then $\forall x_0\, \varphi \vdash \psi$. Let w be the arithmetical counterpart of t (so if t is x_k then $w := y_k$ and if t is c_k then $w := z_k$). We have $\varphi[x_0 {\leftarrow} t]^{\circ_\tau} = \varphi^{\circ_\tau}[y_0 {\leftarrow} w]$. Reason in T and let $\theta, \boldsymbol{y}$ and $\boldsymbol{z}$ be arbitrary, where y_0 appears in $\boldsymbol{y}$ if and only if x_0 is a free variable of ψ. We assume $\Box_{\psi^{\circ_\tau}} \theta$. If t appears in φ or in ψ, then the value of w was already fixed when we picked arbitrary $\boldsymbol{y}$ and $\boldsymbol{z}$. Otherwise, fix $w := 0$ and in either case use the induction hypothesis to obtain $\Box_{\varphi^{\circ_\tau}[y_0 \leftarrow \overline{w}]} \theta$. It is then clear that $\Box_{\exists\, y_0\, \varphi^{\circ_\tau}} \theta$ holds as well.

Finally, the term instantiation and constant elimination rules boil down to variable renaming. $\Box$

5.3 Arithmetical completeness

We show arithmetical completeness with respect to both kinds of realization, leading to a proof of:[1]

$$\mathsf{QRC}_1 = \mathsf{QPL}^+(T) = \mathsf{QRL}(T),$$

where T is a sound c.e. extension of EA + $\mathbf{COLL}_{\Sigma^0_1}$.

5.3.1 Finitary

The proof of finitary arithmetical completeness, that $\mathsf{QRC}_1 \supseteq \mathsf{QPL}^+(T)$, closely follows the proof of Solovay's theorem (Theorem 2.4.7) presented by Boolos [49]. The idea of Solovay's proof is to take a Kripke model not satisfying the desired unprovable formula and embed it in the language of arithmetic. If the embedding is done correctly, it is possible to prove that a formula is satisfied at a world of the Kripke model exactly when its arithmetical interpretation is a consequence of the representation of that world in the desired theory T. In this subsection, we fix an elementary presented theory T extending EA that is furthermore sound (i.e., $T \vdash \varphi$ implies $\mathbb{N} \models \varphi$).

Given two strictly positive formulas φ and ψ such that $\varphi \not\vdash \psi$ in QRC_1, let $\mathcal{M}_{\varphi,\psi}$ be a finite, irreflexive, and constant domain adequate model satisfying φ and not satisfying ψ at the root 1 under a 1-assignment $g_{\varphi,\psi}$. This model and assignment exist by Theorem 4.4.13. Since $\mathcal{M}_{\varphi,\psi}$ has constant domain, we refer to $g_{\varphi,\psi}$ and any other i-assignments as just assignments, omitting the relevant world.

We assume that the worlds of $\mathcal{M}_{\varphi,\psi}$ are $W = \{1, 2, \ldots, N\}$, where 1 is the root. We define a new (adequate) model $\mathcal{M}$ that is a copy of $\mathcal{M}_{\varphi,\psi}$, except that it has an extra world 0 as the new root. This world 0 is connected to all the other worlds through R and has the same domain, constant interpretation, and relation symbol interpretation as 1. The functions $\eta_{0,i}$ with $0 < i \leq N$ are all defined as the identity function.

Let λ_i be the Solovay sentences for the propositional frame upon which $\mathcal{M}$ is based (i.e., just the set of worlds and the relation R) as defined by Boolos [49]. Since this frame is finite, transitive, and irreflexive, we have the following embedding lemma.

Lemma 5.3.1 (Embedding [49, 79])**.**

1. *$T \vdash \bigvee_{i \leq N} \lambda_i$;*
2. *$T \vdash \lambda_i \to \bigwedge_{j \leq N, j \neq i} \neg\lambda_j$, for $i \leq N$;*
3. *$T \vdash \lambda_i \to \bigwedge_{j \leq N, iRj} \Diamond_T \lambda_j$, for $i \leq N$;*
4. *$T \vdash \lambda_i \to \Box_T \bigvee_{j \leq N, iRj} \lambda_j$, for $0 < i \leq N$;*
5. *$\mathbb{N} \models \lambda_0$.*

Each λ_i is meant to represent the world i of $\mathcal{M}$. For example, if iRj, we have $T \vdash \lambda_i \to \Diamond\lambda_j$.

[1] From now on we abuse notation and write QRC_1 instead of $\{\langle\varphi, \psi\rangle \mid \varphi \vdash \psi\}$.

The domain of every world of $\mathcal{M}$ is M. Let m be the size of M, and $\ulcorner\cdot\urcorner$ be a bijection between M and the set of numerals $\{\overline{0}, \ldots, \overline{m-1}\}$. We now define for a given n-ary relation symbol S and terms $\boldsymbol{t} = t_0, \ldots, t_{n-1}$ the finitary realization $\cdot^*$ as follows:

$$S(\boldsymbol{t})^* := \bigvee_{i \leq N} \left(\lambda_i \wedge \Phi_i^{S(\boldsymbol{t})}\right), \text{ where}$$

$$\Phi_i^{S(\boldsymbol{t})} := \bigvee_{\langle a_0, \ldots, a_{n-1}\rangle \in S^{J_i}} \bigwedge_{l<n} \left(\ulcorner a_l \urcorner = \begin{cases} y_k \bmod \overline{m} & \text{if } t_l = x_k \\ z_k \bmod \overline{m} & \text{if } t_l = c_k \end{cases}\right).$$

Here, x_k is any modal variable, y_k is the corresponding arithmetical variable, c_k is any constant, z_k is the corresponding arithmetical variable, and $\cdot^{J_i}$ is the denotation of a relation symbol at world i. Note that $x_k \in \mathsf{fv}(S(\boldsymbol{t}))$ if and only if $y_k \in \mathsf{fv}(S(\boldsymbol{t})^*)$ and c_k appears in $S(\boldsymbol{t})$ if and only if $z_k \in \mathsf{fv}(S(\boldsymbol{t})^*)$. The $\cdot^*$ interpretation is extended to generic formulas as described in Definition 5.2.1, and the free variables and constants of φ correspond to the free variables of φ^{*_T} just as in Remark 5.2.2.

We further note that φ^{*_T} is invariant under replacing variables by themselves modulo the size of the domain of $\mathcal{M}$, provably in T.

Lemma 5.3.2. *For any* QRC_1 *formula* φ *and any arithmetical variable* w*:*

- $T \vdash \varphi^{*_T} \leftrightarrow \varphi^{*_T}[w \leftarrow w \bmod \overline{m}]$*;*
- $T \vdash \forall w\, \varphi^{*_T} \leftrightarrow \forall w < \overline{m}\, \varphi^{*_T}$.

Proof. The second item is a straightforward consequence of the first, which we prove by external induction on φ. The cases of $\top$ and conjunction are trivial.

For the case of the relation symbols, consider $S(x, c)$ as a reasonable simplification (the general case easily follows). The formula $S(x, c)^*$ has two free variables, namely y and z, so the result is trivial when w is not one of these.

We check first that $T \vdash S(x, c)^* \leftrightarrow S(x, c)^*[y \leftarrow y \bmod \overline{m}]$. We have:

$$S(x, c)^* = \bigvee_{j \leq N} \left(\lambda_j \wedge \bigvee_{\langle a_0, a_1\rangle \in S^{J_j}} (\ulcorner a_0 \urcorner = y \bmod \overline{m} \wedge \ulcorner a_1 \urcorner = z \bmod \overline{m})\right),$$

and hence, noting that y and z are different variables,

$$S(x, c)^*[y \leftarrow y \bmod \overline{m}] = \bigvee_{j \leq N} \left(\lambda_j \wedge \bigvee_{\langle a_0, a_1\rangle \in S^{J_j}} (\ulcorner a_0 \urcorner = (y \bmod \overline{m}) \bmod \overline{m} \wedge \ulcorner a_1 \urcorner = z \bmod \overline{m})\right).$$

These are equivalent because T proves $(y \bmod \overline{m}) \bmod \overline{m} = y \bmod \overline{m}$. If w is z instead, the argument is analogous.

Consider now the case of $\forall x\, \varphi$. If w is y there is nothing to show because y is not a free variable of $(\forall x\, \varphi)^{*_T}$. Thus we may assume that $(\forall x\, \varphi)^{*_T}[w \leftarrow w \bmod \overline{m}]$ is the same as $\forall y\, (\varphi^{*_T}[w \leftarrow w \bmod \overline{m}])$. By the induction hypothesis, we have that $T \vdash \varphi^{*_T} \leftrightarrow \varphi^{*_T}[w \leftarrow w \bmod \overline{m}]$, from which we obtain $T \vdash \forall y\, \varphi^{*_T} \leftrightarrow \forall y\, \varphi^{*_T}[w \leftarrow w \bmod \overline{m}]$, as desired.

Finally, in the case of $\Diamond\varphi$, note that $(\Diamond\varphi)^{*_T}[w \leftarrow w \bmod \overline{m}]$ is the same as $\Diamond_T \varphi^{*_T}[w \leftarrow w \bmod \overline{m}]$. Thus the result is straightforward from the induction hypothesis under the box. $\square$

We now prove two versions of a truth lemma, one for when φ is satisfied at a world i of $\mathcal{M}$, and one for when it isn't. We wish to show that T proves φ^{*_T} (respectively $\neg\varphi^{*_T}$) as a consequence of λ_i, as long as the free variables of φ are interpreted in the same way in both settings.

In order to use concise notation, we shall write $\boldsymbol{y}$ instead of $y_0, \ldots, y_{n-1}$, and similarly for other terms. We also abbreviate iterated substitutions, writing $\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner]$ instead of:

$$\varphi^{*_T}[y_0{\leftarrow}\ulcorner g(x_0) \urcorner] \cdots [y_{n-1}{\leftarrow}\ulcorner g(x_{n-1}) \urcorner],$$

and writing $\varphi^{*_T}[\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$ instead of:

$$\varphi^{*_T}[z_0{\leftarrow}\ulcorner (c_0)^{I_0} \urcorner] \cdots [z_{k-1}{\leftarrow}\ulcorner (c_{k-1})^{I_0} \urcorner].$$

Lemma 5.3.3 (Truth: positive). *Let φ be a QRC_1 formula with free variables $\boldsymbol{x} = x_0, \ldots, x_{n-1}$ and constants $\boldsymbol{c} = c_0, \ldots, c_{k-1}$. Then for any world $i \leq N$ (i.e., $i \in \mathcal{M}$) and any assignment g:*

$$\mathcal{M}, i \Vdash^g \varphi \Longrightarrow T \vdash \lambda_i \to \varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

Proof. By external induction on the complexity of φ. The cases of $\top$ and conjunction are straightforward.

In the case of relational symbols, assume for simplicity's sake that the relevant formula is $S(x, c)$. If $\mathcal{M}, i \Vdash^g S(x, c)$, then $\langle g(x), c^{I_i} \rangle \in S^{J_i}$. Reason in T and assume λ_i. By the definition of $\cdot^*$, it suffices to prove:

$$\Phi_i^{S(x,c)}[y{\leftarrow}\ulcorner g(x) \urcorner][z{\leftarrow}\ulcorner (c)^{I_0} \urcorner],$$

which implies $S(x, c)^*[y{\leftarrow}\ulcorner g(x) \urcorner][z{\leftarrow}\ulcorner (c)^{I_0} \urcorner]$ under the assumption λ_i. By the definition of $\Phi_i^{S(x,c)}$, we need to find a pair $\langle a_0, a_1 \rangle \in S^{J_i}$ such that $\ulcorner a_0 \urcorner = \ulcorner g(x) \urcorner \bmod \overline{m}$ and $\ulcorner a_1 \urcorner = \ulcorner c^{I_0} \urcorner \bmod \overline{m}$. Noting that $\ulcorner b \urcorner \bmod \overline{m}$ is provably equal to $\ulcorner b \urcorner$ for any b in the domain of $\mathcal{M}$, we pick $a_0 := g(x)$ and $a_1 := c^{I_0}$. This concludes this step of the proof because c^{I_0} is equal to c^{I_i}, for any world i.

For $\forall x_0\, \varphi$, assume for simplicity's sake that the free variables of φ are x_0 and x_1. If we have $\mathcal{M}, i \Vdash^g \forall x_0\, \varphi$ then for every assignment $h \sim_{x_0} g$ we have $\mathcal{M}, i \Vdash^h \varphi$. We wish to show:

$$T \vdash \lambda_i \to (\forall y_0\, \varphi^{*_T})[y_1{\leftarrow}\ulcorner g(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

Reason in T and assume λ_i. By Lemma 5.3.2, it is enough to show:

$$(\forall y_0 < \overline{m}\, \varphi^{*_T})[y_1{\leftarrow}\ulcorner g(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

Since y_0, y_1, and $\boldsymbol{z}$ are all different variables, we can push the substitutions inside and prove:

$$\forall y_0 < \overline{m}\, \varphi^{*_T}[y_1{\leftarrow}\ulcorner g(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$$

instead. Let $y_0 < \overline{m}$ be arbitrary. Since $\ulcorner \cdot \urcorner$ is a bijection, there is a in the domain of $\mathcal{M}$ such that $\ulcorner a \urcorner = y_0$. We define an assignment h such that $h \sim_{x_0} g$ and $h(x_0) := a$. By assumption, $\mathcal{M}, i \Vdash^h \varphi$, so by the induction hypothesis we obtain:

$$\varphi^{*_T}[y_0{\leftarrow}\ulcorner h(x_0) \urcorner][y_1{\leftarrow}\ulcorner h(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

This concludes the argument because $\ulcorner h(x_0) \urcorner = y_0$ and $h(x_1) = g(x_1)$.

Finally, consider the case of $\Diamond\varphi$. If $\mathcal{M}, i \Vdash^g \Diamond\varphi$, then there is a world j such that iRj and $\mathcal{M}, j \Vdash^g \varphi$. Reason in T and assume λ_i. By Lemma 5.3.1.3, we obtain $\Diamond_T \lambda_j$. Then the induction hypothesis under the box gives us the desired $\Diamond_T(\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner])$. □

Unlike the positive truth lemma, we only have the negative version for worlds of $\mathcal{M}$ other than its root, i.e., other than 0.

Lemma 5.3.4 (Truth: negative). *Let φ be a QRC_1 formula with free variables $\boldsymbol{x} = x_0, \dots, x_{n-1}$ and constants $\boldsymbol{c} = c_0, \dots, c_{k-1}$. Then for any world $0 < i \leq N$ and any assignment g:*

$$\mathcal{M}, i \nVdash^g \varphi \Longrightarrow T \vdash \lambda_i \to \neg\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

Proof. By external induction on the complexity of φ. The cases of $\top$ and conjunction are straightforward.

For the relation symbols, consider $S(x, c)$ for simplicity's sake. If $\mathcal{M}, i \nVdash^g S(x, c)$, then $\langle g(x), c^{I_i} \rangle \notin S^{J_i}$. Reason in T and assume λ_i. We obtain $\neg\lambda_j$ for every $j \neq i$ by Lemma 5.3.1.2, and hence need only show $\neg\Phi_i^{S(x,c)}[y{\leftarrow}\ulcorner g(x) \urcorner][z{\leftarrow}\ulcorner c^{I_0} \urcorner]$ by the definition of $\cdot^*$. In other words, we need to check that if $\langle a_0, a_1 \rangle \in S^{J_i}$, then either $\ulcorner a_0 \urcorner \neq \ulcorner g(x) \urcorner \bmod \overline{m}$, or $\ulcorner a_1 \urcorner \neq \ulcorner c^{I_0} \urcorner \bmod \overline{m}$. This follows from our observation that $\langle g(x), c^{I_i} \rangle \notin S^{J_i}$, taking into account that $\ulcorner b \urcorner \bmod \overline{m}$ is equal to $\ulcorner b \urcorner$ for any domain element b, that $\ulcorner \cdot \urcorner$ is injective, and that $c^{I_0} = c^{I_i}$.

Consider now the case of $\forall x_0\, \varphi$ and assume as a simplification that $\mathsf{fv}(\varphi) = \{x_0, x_1\}$. If $\mathcal{M}, i \nVdash^g \forall x_0\, \varphi$, then there is an assignment $h \sim_{x_0} g$ such that $\mathcal{M}, i \nVdash^h \varphi$. Reason in T and assume λ_i. From the induction hypothesis we obtain $\neg\varphi^{*_T}[y_0{\leftarrow}\ulcorner h(x_0) \urcorner][y_1{\leftarrow}\ulcorner h(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$, which then implies $\neg\forall y_0\, \varphi^{*_T}[y_1{\leftarrow}\ulcorner h(x_1) \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$. This is what we wanted, taking into account that $h(x_1) = g(x_1)$.

Finally, in the case of $\Diamond\varphi$, assume that $\mathcal{M}, i \nVdash^g \Diamond\varphi$. Then for every j such that iRj, we have $\mathcal{M}, j \nVdash^g \varphi$ and thus the induction hypothesis for j gives us $T \vdash \lambda_j \to \neg\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$, which put together imply $T \vdash \bigvee_{iRj} \lambda_j \to \neg\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$. Reason in T and assume λ_i. By Lemma 5.3.1.4 and our assumption, we obtain $\Box_T \bigvee_{iRj} \lambda_j$. Taking the previous observation under the box, we conclude $\Box_T \neg\varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$, which is $\neg(\Diamond\varphi)^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$. □

With both versions of the truth lemma at hand, we can prove the finitary arithmetical completeness of QRC_1.

Theorem 5.3.5 (Finitary arithmetical completeness). *If φ, ψ are QRC_1 formulas with free variables $\boldsymbol{x}$ and constants $\boldsymbol{c}$ such that $\varphi \nvdash \psi$, we have $T \nvdash (\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g_{\varphi,\psi}(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$.*

Proof. Recall that $\mathcal{M}$ satisfies φ and not ψ at world 1 under the assignment $g := g_{\varphi,\psi}$.

Since $\mathcal{M}, 1 \Vdash^g \varphi$, we obtain $T \vdash \lambda_1 \to \varphi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$ from Lemma 5.3.3. Since $\mathcal{M}, 1 \nVdash^g \psi$, we obtain $T \vdash \lambda_1 \to \neg\psi^{*_T}[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$ from Lemma 5.3.4. Thus:

$$T \vdash \lambda_1 \to \neg(\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner].$$

By Lemma 5.3.1.3 and the fact that $0R1$, we obtain $T \vdash \lambda_0 \to \Diamond_T \lambda_1$, so putting this together with the previous observation under the box, $T \vdash \lambda_0 \to \Diamond_T \neg(\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y}{\leftarrow}\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}{\leftarrow}\ulcorner \boldsymbol{c^{I_0}} \urcorner]$.

By Lemma 5.3.1.5, we know that $\mathbb{N} \vDash \lambda_0$, and thus by the soundness of T, we know that $\mathbb{N} \vDash \Diamond_T \neg(\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y}\leftarrow\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}\leftarrow\ulcorner \boldsymbol{c^{I_0}} \urcorner]$. Then $T \nvdash (\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y}\leftarrow\ulcorner \boldsymbol{g(x)} \urcorner][\boldsymbol{z}\leftarrow\ulcorner \boldsymbol{c^{I_0}} \urcorner]$, as desired. $\square$

We have now proved, via Theorems 5.2.3 and 5.3.5, that for any sound c.e. theory T extending EA:

$$\mathsf{QRC}_1 = \mathsf{QPL}^+(T).$$

5.3.2 Infinitary

The proof of infinitary arithmetical completeness is essentially a corollary of finitary arithmetical completeness, noting that for the specific realization $\cdot^{*_T}$ defined in Section 5.3.1 and in the presence of $\mathbf{COLL}_{\Sigma^0_1}$ we can prove that its translation to an infinitary realization $\cdot^{\circledast}$ behaves exactly like the finitary version.[2] For this reason, we now take T to be a sound c.e. extension of $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$.

Recall that the infinitary arithmetical realization $\cdot^{\circledast}$ that behaves like a given finitary realization $\cdot^*$ is defined such that $S(\boldsymbol{t})^{\circledast} := \tau \vee (u = \ulcorner S(\boldsymbol{t})^{*} \urcorner)$ and extended to non-atomic formulas as described in Definition 5.2.4. Note that $x_k \in \mathsf{fv}(\varphi)$ if and only if $y_k \in \mathsf{fv}(\varphi^{\circledast_\tau})$ and c_k appears in φ if and only if $z_k \in \mathsf{fv}(\varphi^{\circledast_\tau})$, as in Remark 5.2.5.

Lemma 5.3.6. *Let $\cdot^*$ be the arithmetical realization used for the proof of finitary arithmetical completeness in Section 5.3.1. Then for any QRC_1 formula φ with free variables $\boldsymbol{x}$ and constants $\boldsymbol{c}$:*

$$T \vdash \forall \theta \, \forall \boldsymbol{y}, \boldsymbol{z} \, \left(\Box_{\varphi^{\circledast_\tau}} \theta \leftrightarrow \Box_\tau (\varphi^{*_T} \to \theta) \right),$$

where θ is a closed formula.

Proof. The left-to-right implication was already proved in Lemma 5.2.8 for any realization $\cdot^*$. For the right-to-left implication we proceed by external induction on φ. There is nothing to show for $\top$ and the case of relation symbols is a straightforward consequence of the formalized deduction theorem.

In the case of conjunction, we take $\varphi = \psi \wedge \delta$ and omit the variables $\boldsymbol{y}$ and $\boldsymbol{z}$, as they introduce visual clutter but don't make the proof any more complex. Fix θ, $\boldsymbol{y}$ and $\boldsymbol{z}$. By the induction hypothesis (taking θ to be ψ^{*_T}) we see that $\Box_{\psi^{\circledast_\tau}} \psi^{*_T}$ and likewise $\Box_{\delta^{\circledast_\tau}} \delta^{*_T}$. Thus $\Box_{\psi^{\circledast_\tau} \vee \delta^{\circledast_\tau}} (\psi^{*_T} \wedge \delta^{*_T})$. By assumption we have $\Box_\tau(\psi^{*_T} \wedge \delta^{*_T} \to \theta)$, and since $\varphi^{\circledast_\tau}$ extends T, we may conclude $\Box_{\varphi^{\circledast_\tau}} \theta$ as desired.

The $\forall$ case follows the same idea as the $\wedge$ case. Consider $\varphi = \forall x_0 \, \psi$, with $\mathsf{fv}(\psi) = \{x_0, x_1\}$ for simplicity's sake. Note that x_0 is represented by y_0 in T, and this is always a different variable from any z used to represent QRC_1 constants. As there is no further complication with constants, we omit them. Reason in T and let θ and l be arbitrary. Assume $\Box_\tau(\forall y_0 \, \psi^{*_T}[y_1\leftarrow l] \to \theta)$. By Lemma 5.3.2 under the box, we obtain $\Box_\tau(\forall y_0 < \overline{m} \, \psi^{*_T}[y_1\leftarrow l] \to \theta)$, where m is the size of the domain of $\mathcal{M}$. Using the induction hypothesis for each $k < m$ with $\theta := \psi^{*_T}[y_1\leftarrow l][y_0\leftarrow k]$, $y_0 := k$, and $y_1 := l$, we get:

$$\forall k < \overline{m} \, \Box_{\psi^{\circledast_\tau}[y_1\leftarrow \bar{l}][y_0\leftarrow \bar{k}]} \psi^{*_T}[y_1\leftarrow l][y_0\leftarrow k],$$

[2]This is the other implication of Lemma 5.2.8 mentioned in Section 5.2.2.

and in particular:

$$\forall\, k < \overline{m}\, \Box_{\exists\, y_0\, \psi^{\circledast_\tau}[y_1 \leftarrow \bar{l}]}\, \psi^{*_T}[y_1 \leftarrow l][y_0 \leftarrow k].$$

Then by $\mathbf{COLL}_{\Sigma^0_1}$ we can change the order of the quantifier and the box, which allows us to complete this part of the proof with $\Box_{\exists\, y_0\, \psi^{\circledast_\tau}[y_1 \leftarrow \bar{l}]}\, \forall\, y_0 < \overline{m}\, \psi^{*_T}[y_1 \leftarrow l]$.

Finally, for the case of $\varphi = \Diamond\psi$, we start by observing that the induction hypothesis with $\theta := \bot$ yields $T \vdash \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Diamond_{\psi^{\circledast_\tau}} \top \to \Diamond_\tau \psi^{*_T})$. Note that $(\Diamond\psi)^{\circledast_\tau} = \tau \vee (u = \ulcorner \Diamond_{\psi^{\circledast_\tau}} \top \urcorner)$, and thus $\Box_\tau(\Diamond_{\psi^{\circledast_\tau}} \top \to \theta)$ implies $\Box_{(\Diamond\psi)^{\circledast_\tau}}\theta$ by the formalized deduction theorem. The previous observation under the box then suffices to finish the proof. $\Box$

We are ready to prove arithmetical completeness for any sound c.e. theory extending $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$.

Theorem 5.3.7 (Infinitary arithmetical completeness)**.** *Let T be a c.e. theory of arithmetic extending* $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$*. Then* $\mathsf{QRC}_1 \supseteq \mathsf{QRL}(T)$*.*

Proof. Recall the definition of $\mathsf{QRL}(T)$:

$$\mathsf{QRL}(T) := \{\langle \varphi, \psi \rangle \in \mathcal{L}^+_{\Box\forall\mathsf{c}} \times \mathcal{L}^+_{\Box\forall\mathsf{c}} \mid \text{for any } \cdot^\circ,\ \text{we have } T \vdash \forall\,\theta\, \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Box_{\psi^{\circ_\tau}}\theta \to \Box_{\varphi^{\circ_\tau}}\theta)\},$$

where θ is closed.

Given $\mathcal{L}^+_{\Box\forall\mathsf{c}}$ formulas φ and ψ such that $\varphi \not\vdash \psi$, we know from finitary completeness that there is some finitary realization $\cdot^*$ such that $T \not\vdash (\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y} \leftarrow \overline{\boldsymbol{k}}][\boldsymbol{z} \leftarrow \overline{\boldsymbol{l}}]$, where $\boldsymbol{y}, \boldsymbol{z}$ are the free variables of $\varphi^{*_T} \to \psi^{*_T}$ and $\overline{\boldsymbol{k}}, \overline{\boldsymbol{l}}$ are specific terms described in Theorem 5.3.5 whose definition is not relevant at the moment.

We show that the realization $\cdot^\circledast$ defined from $\cdot^*$ as explained above is such that:

$$T \not\vdash \forall\,\theta\, \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Box_{\psi^{\circledast_\tau}}\theta \to \Box_{\varphi^{\circledast_\tau}}\theta).$$

Suppose towards a contradiction that T does prove this formula. Then by Lemma 5.3.6:

$$T \vdash \forall\,\theta\, \forall\, \boldsymbol{y}, \boldsymbol{z}\, (\Box_T(\psi^{*_T} \to \theta) \to \Box_T(\varphi^{*_T} \to \theta)).$$

Taking $\theta := \psi^{*_T}[\boldsymbol{y} \leftarrow \overline{\boldsymbol{k}}][\boldsymbol{z} \leftarrow \overline{\boldsymbol{l}}]$, $\boldsymbol{y} := \overline{\boldsymbol{k}}$, and $\boldsymbol{z} := \overline{\boldsymbol{l}}$, we conclude:

$$T \vdash \Box_T(\varphi^{*_T} \to \psi^{*_T})[\boldsymbol{y} \leftarrow \overline{\boldsymbol{k}}][\boldsymbol{z} \leftarrow \overline{\boldsymbol{l}}].$$

This together with the soundness of T contradicts finitary completeness. $\Box$

We have seen that $\mathsf{QRC}_1 = \mathsf{QPL}^+(T) = \mathsf{QRL}(T)$ for any sound c.e. theory T extending $\mathsf{EA} + \mathbf{COLL}_{\Sigma^0_1}$. Thus, $\mathsf{QPL}^+(T)$ is constant over a large class of theories, and for these theories it does not depend on the specific axiomatization chosen for T. This is similar to the propositional case, but simpler than the full predicate case, where $\mathsf{QPL}(T)$ is known to depend on both T (as shown by Montagna [175]) and τ (as shown by Artemov [21]; see also Kurahashi [146, 147]).

5.4 Heyting Arithmetic

We end this chapter with a foray into intuitionistic arithmetic. The provability logic of Heyting Arithmetic (HA) was somewhat of a mystery for a long time [133, 22], until it was recently axiomatized by Mojtahedi [174]. This axiomatization is both decidable and of considerable conceptual complexity. We show that QRC_1 is sound with respect to Heyting Arithmetic (HA) and announce (but do not prove here) that it is complete as well. Thus, it represents a simple axiomatization of the strictly positive provability logic of HA.

Let $\eta(u)$ be a natural elementary axiomatization of HA. The HA-provability of φ can thus be expressed by $\Box_\eta\varphi$, as defined in Section 2.3. We observe that a number of standard results in the realm of classical provability logic still hold in the intuitionistic case.

Lemma 5.4.1 ([208, Proposition 2.1.10])**.** *Let τ be a Σ^0_1 axiomatization of an arithmetical theory T extending* HA*, and φ, ψ be formulas in the language of arithmetic. Then:*

- *if $T \vdash \varphi$ then* $\mathsf{HA} \vdash \Box_\tau\varphi$*;*
- $\mathsf{HA} \vdash \Box_\tau(\varphi \to \psi) \to (\Box_\tau\varphi \to \Box_\tau\psi)$*;*
- *if σ is a Σ^0_1 formula, then* $\mathsf{HA} \vdash \sigma \to \Box_\tau\sigma$*.*

Lemma 5.4.2 (Collection [104, Proposition 5.13])**.** HA *proves full collection.*

As in the classical case, we define $\Diamond_\tau\varphi$ as $\neg\Box_\tau\neg\varphi$ when τ is an axiomatization of an extension of HA.

We extend a generic realization $\cdot^\circ$ to non-predicate formulas as in the classical case (Definition 5.2.4), using HA as the base theory. Note that φ°_η} is equivalent to a Σ^0_1 formula and extends HA, both of these provably in HA.

Theorem 5.4.3 (Arithmetical soundness w.r.t. HA)**.**

- $\mathsf{QRC}_1 \subseteq \mathsf{QPL}^+(\mathsf{HA})$*;*
- $\mathsf{QRC}_1 \subseteq \mathsf{QRL}(\mathsf{HA})$*.*

Proof. The proofs of Theorems 5.2.3 and 5.2.9 can be repeated for HA, using Lemmas 5.4.1 and 5.4.2 instead of their classical counterparts. □

We have a proof that QRC_1 is also arithmetically complete with respect to HA, but is not yet published.

5.5 Open questions

We have seen how extending RC_1 to a quantified language can lead to a quantified provability logic with nice properties. The obvious next step is to extend QRC_1 to the polymodal setting. We have a sound axiomatization in mind for the polymodal QRC, but completeness is still just a conjecture for both Kripke and arithmetical semantics. We hope to further explore this in the future. Besides being interesting for its own sake, the polymodal language might enable applications to ordinal analysis.

We have provided an upper bound for the complexity of QRC_1, but it is likely not tight. Furthermore, the current decidability proof is not constructive. If we had an axiomatization of QRC_1 with the subformula property, we might obtain a different decision procedure and improve on those two observations. They could also be tackled through other means.

Another avenue for study is strictly positive quantified truth logic. Are there ways of extending QRC_1 to the strictly positive fragment of the quantified truth logic of PA? This could be inspired by how the truth logic of PA, called GLS, is an extension of GL.

Part II

Temporal laws

6
Background

Contents

6.1 Introduction

This part is an outcome of a collaboration with industry (Formal Vindications S.L. and Guretruck S.L.), lawyers, and legislators, where the ultimate goal is to develop verified legal software. Various legal decisions are made on the basis of algorithmic processing of data, but two common issues come into play. First, even though there are written regulations specifying these algorithms, they are sometimes unclear, overly complicated, inefficient, contradictory, or incomplete. Second, even assuming a good specification is available, unformalized implementations are always at risk of being wrong, and often are. These observations are not restricted to the legal context, of course. Specifying and correctly implementing software is hard in general. However, these kinds of issues should be mitigated if at all possible, especially in high-stakes contexts.

Our particular focus are the European regulations on the road transport of public and goods. They are a suitable target because a large part of that regulatory landscape depends on automated checking of drivers' travel logs. These are obtained via tachographs, which are to trucks what black-boxes are to airplanes: they register data on the truck and the driver, such as speed, movement and others. The drivers under these regulations have their records regularly checked, and may be fined or charged with serious infractions if the checker software finds a problem. Unfortunately, such problems do not always exist, and are instead artifacts of hardware or software bugs in either the tachograph or the checker itself [87].

Some of these issues, such as hardware problems, are beyond the scope or our work. However, some software problems can be mitigated through clarification of the intended laws and formal verification [172, 130]. Thus, we analyze the relevant regulations and identify some of their problematic articles in Chapter 7. Then, having settled on acceptable interpretations of these articles, we look for mathematical systems in which they can be expressed, aiming for tractability. A general formalization of the law with its obligations, prohibitions, and permissions, would likely require deontic reasoning [116, 125]. However, we focus on the temporal aspects and as such we are able to work within fragments of Monadic Second-order Logic (see Chapter 8). We also take preliminary steps into formal verification of these articles, although the ones described in this thesis (in Chapter 11) are only a prerequisite and not proper regulation formalization. Formal Vindications S.L. is currently working on the latter.

In this chapter we describe the articles under study and briefly introduce Linear Temporal Logic (LTL) and Monadic First and Second-order Logic (MFO and MSO).

6.2 European road transport regulations

We focus on small parts of two specific European road transport regulations, chosen for being somewhat puzzling.

Regulation (EU) 2016/799 [90] describes the requirements for the use of tachographs. These are digital devices that record the activities of road transport drivers on a second-by-second resolution. This data is used to determine whether drivers have complied with Regulation (EC) 561/2006 [88], which is written assuming a minute-by-minute resolution of the driver's activity. Requirements (51) and (52) of Regulation (EU) 2016/799 [90] describe how the second-by-second data recorded by the tachographs is to be translated into a minute-by-minute format. They read as follows:

(51) Given a calendar minute, if DRIVING is registered as the activity of both the immediately preceding and the immediately succeeding minute, the whole minute shall be regarded as DRIVING.

(52) Given a calendar minute that is not regarded as DRIVING according to requirement (51), the whole minute shall be regarded to be of the same type of activity as the longest continuous activity within the minute (or the latest of the equally long activities).

Note that Requirement (51) seems to be lacking a base case and Requirement (52) explicitly mentions Requirement (51), so it is unlikely that Requirement (52) was meant as the only base case. However, there are no other requirements pertaining to this translation. We investigate this issue in Section 7.2.

Once data has been formatted according to these requirements, it must be checked for compliance with Regulation (EC) 561/2006 [88]. There are various requirements, mostly restraining the amount of time that may be spent driving within a given period of time and providing a lower bound on the amount and frequency of time spent resting. For instance, drivers must have a weekly rest period (similar to a weekend), as prescribed by the following articles:

§4(h) 'regular weekly rest period' means any period of rest of at least 45 hours.

§4(i) 'a week' means the period of time between 00.00 on a Monday and 24.00 on the following Sunday.

§8.6 In any two consecutive weeks, a driver shall take at least:

- two regular weekly rest periods, or
- one regular weekly rest period and one reduced weekly rest period of at least 24 hours. However, the reduction shall be compensated by an equivalent period of rest taken en bloc before the end of the third week following the week in question.

A weekly rest period shall start no later than at the end of six 24-hour periods from the end of the previous weekly rest period.

§8.7 Any rest taken as compensation for a reduced weekly rest period shall be attached to another rest period of at least nine hours.

§8.9 A weekly rest period that falls in two weeks may be counted in either week, but not in both.

We identify some difficulties with the above excerpt. First, the concept of week is strict and depends on a calendar. There is no restriction on ending a weekly rest period before the end of the week, but §8.9 does say that each such period must be assigned to one of the weeks it intercepts.

Determining whether a given configuration of weekly rest periods can be assigned to weeks in a legal way may be non-trivial, and we study it in Sections 7.4.1, 8.2.1 and 8.3.1.

Second, there are hard restrictions on the circumstances in which reduced weekly rest periods are allowed, and in particular they must be compensated by resting extra hours in the near future. Again, checking whether a given configuration of weekly rest periods satisfies these rules can be hard, even assuming there are no complications with the assignments of weekly rest periods to weeks. We further study the compensation requirements in Sections 7.4.2, 8.2.2 and 8.3.2.

6.3 Linear Temporal Logic

Linear Temporal Logic, denoted by LTL, was first proposed by Pnueli [182] as a tool for formal reasoning about temporal properties of systems or programs. It is a propositional modal logic with modalities representing relationships between moments in (linear) time, such as "in the next moment" or "some time in the future." One of the nice things about LTL is that it lends itself well to model checking, and it is widely used in that capacity [128]. The model checking problem for LTL is PSPACE-complete [192], but time-wise it is linear on the size of the model and exponential on the size of the formula being checked [159]. In practice, models are quite large and formulas are small, so model checking in LTL is feasible.

Here, we focus on the language and semantics of LTL, as well as on bisimulation results that are useful during Section 8.3. For a more thorough treatment of LTL in particular and model checking in general, refer to Baier and Katoen [25]. See also Vardi [206] for a history of LTL and relevant milestones.

The language of LTL is denoted by $\mathcal{L}_{\mathsf{OU}}$ and defined as follows:

$$\mathcal{L}_{\mathsf{OU}} ::= \bot \mid p \mid \varphi \to \varphi \mid \mathsf{O}\varphi \mid \varphi \,\mathsf{U}\, \varphi,$$

where $p \in \textsc{Prop}$ is a propositional variable and φ is a $\mathcal{L}_{\mathsf{OU}}$ formula. We read O as "next" and U as "until." The missing Boolean connectives are defined as abbreviations in the usual way. The $\Diamond$ and $\Box$ modalities can be defined as abbreviations as well, setting $\Diamond\varphi := \top \,\mathsf{U}\, \varphi$ and $\Box\varphi := \neg\Diamond\neg\varphi$, although their semantics are different from the one presented in Section 2.4.1 for $\mathcal{L}_\Box$: in this context, $\Box\varphi$ is interpreted as meaning that φ holds in the current world and in all future worlds as well, instead of only in all future ones. Similarly, $\Diamond\varphi$ is interpreted as meaning that there is some world, either current or future, where φ holds.

Another crucial difference is that the relevant LTL models are linear and can be reduced to the structure $\langle \mathbb{N}, S \rangle$, where $S(n) := n + 1$. Thus, for our purposes, an LTL model is merely a function $J\colon \textsc{Prop} \to \wp(\mathbb{N})$. We write p^J instead of $J(p)$, which matches with the notation of Definition 2.5.3. We also use the letters $\mathcal{M}$ and $\mathcal{N}$ to refer to models when convenient. We define the satisfaction relation $\Vdash$ recursively as follows:

- $J, n \nVdash \bot$;
- $J, n \Vdash p$ if and only if $n \in p^J$, where $p \in \textsc{Prop}$;

- $J, n \Vdash \varphi \rightarrow \psi$ if and only if either $J, n \nVdash \varphi$ or $J, n \Vdash \psi$;
- $J, n \Vdash \bigcirc\varphi$ if and only if $J, S(n) \Vdash \varphi$;
- $J, n \Vdash \varphi \mathsf{U} \psi$ if and only if there exists $k \geq 0$ such that $J, S^k(n) \Vdash \psi$ and for every $0 \leq i < k$ we have $J, S^i(n) \Vdash \varphi$.

A formula φ is said to be satisfiable over a set of models Ω is there are $J \in \Omega$ and $n \in \mathbb{N}$ such that $J, n \Vdash \varphi$ (also written $J, n \Vdash_{\mathsf{LTL}} \varphi$ to distinguish it from other satisfiability notions). Furthermore, if $J, n \Vdash \varphi$ for every $n \in \mathbb{N}$, we say that φ is valid in J and write $J \vDash \varphi$. If φ is valid in J for every model $J \in \Omega$, we say that φ is valid in Ω.

We also consider the fragment of $\mathcal{L}_{\bigcirc\mathsf{U}}$ that excludes U but includes $\Box$, written $\mathcal{L}_{\bigcirc\Box}$ and defined as follows:

$$\mathcal{L}_{\bigcirc\Box} ::= \bot \mid p \mid \varphi \rightarrow \varphi \mid \bigcirc\varphi \mid \Box\varphi,$$

where $p \in \textsc{Prop}$, φ is a $\mathcal{L}_{\bigcirc\Box}$ formula and the Boolean connectives and $\Diamond$ are defined as usual.

The semantics for $\mathcal{L}_{\bigcirc\Box}$ are the same as the semantics for $\mathcal{L}_{\bigcirc\mathsf{U}}$, with the clause for $\Box\varphi$ as follows:

- $J, n \Vdash \Box\varphi$ if and only if for every $k \geq 0$ we have $J, S^k(n) \Vdash \varphi$.

We sometimes wish to express counting. This can be achieved via the following abbreviations, where $n, m \in \mathbb{N}$ and an empty disjunction should be read as $\bot$ and an empty conjunction as $\top$:

- $\bigcirc^0\varphi := \varphi$;
- $\bigcirc^{n+1}\varphi := \bigcirc\bigcirc^n\varphi$;
- $\Diamond^{<n}\varphi := \bigvee_{i=0}^{n-1} \bigcirc^n\varphi$;
- $\Box^{<n}\varphi = \bigwedge_{i=0}^{n-1} \bigcirc^n\varphi$.

Variants with $\leq n$ instead of $<n$ are defined by reading $\leq n$ as $<(n+1)$.

Given any formula φ and a modality $\theta \in \{\bigcirc, \Box, \mathsf{U}\}$, the θ-depth of φ (in symbols, $d_\theta(\varphi)$) is defined as the maximum nesting depth of θ in the usual way.

It would be more practical to use sugared versions of LTL such as Metric Temporal Logic (MTL, [178]), which allows for expressions such as $\bigcirc^{n+1}\varphi$ to be represented succinctly. Since the standard translation of MTL into LTL does not increase U-depth, we focus on that and ignore $\bigcirc$-depth and formula size when making succinctness arguments. Thus, these arguments hold both in LTL and in MTL.

We could additionally consider modalities referring to the past, but they do not add expressive power to LTL in models with a starting point [105]. There is also the possibility of using entirely different logics that may better capture the regulations, such as logics based on intervals [1]. However, we have not yet found an option with a substantial technical advantage over LTL.

We now present a version of bounded bisimulation for $\mathcal{L}_{\bigcirc\Box}$ and $\mathcal{L}_{\bigcirc\mathsf{U}}$ proposed by Kurtonina and de Rijke [148]. A bounded bisimulation in this case is a relation between two LTL models that preserves satisfiability of formulas of bounded depth in both directions. Since both languages include Booleans and $\bigcirc$, it is convenient to begin with a basic notion of bisimulation for the $\Box$-free fragment of $\mathcal{L}_{\bigcirc\Box}$.

Definition 6.3.1 (k-$\bigcirc$-bisimulation)**.** Given a natural number k and two LTL models $\mathcal{M}$ and $\mathcal{N}$, a binary relation $Z \subseteq \mathbb{N} \times \mathbb{N}$ is a k-$\bigcirc$-bisimulation (between $\mathcal{M}$ and $\mathcal{N}$) if whenever $x \; Z \; y$, we have $S^j(x) \in p^{\mathcal{M}}$ if and only if $S^j(y) \in p^{\mathcal{N}}$ for every $p \in$ PROP and $j \leq k$.

We now extend the concept of bounded bisimulation to the languages $\mathcal{L}_{\bigcirc\Box}$ and $\mathcal{L}_{\bigcirc\mathsf{U}}$.

Definition 6.3.2 (k-$\Box$-bisimulation, k-U-bisimulation)**.** Fix a natural number k and two LTL models $\mathcal{M}$ and $\mathcal{N}$. Let $\boldsymbol{Z} := (Z_i)_{i=0}^{\infty}$ be a sequence such that for all $i \in \mathbb{N}$, the relation Z_i is a k-$\bigcirc$-bisimulation and $Z_{i+1} \subseteq Z_i$. Then:

- $\boldsymbol{Z}$ is a k-$\Box$-bisimulation (between $\mathcal{M}$ and $\mathcal{N}$) if whenever $x \; Z_{i+1} \; y$ both of the following hold:

  FORTH $\Box$. for all $x' \geq x$ there exists $y' \geq y$ such that $x' \; Z_i \; y'$;

  BACK $\Box$. for all $y' \geq y$ there exists $x' \geq x$ such that $x' \; Z_i \; y'$.

- $\boldsymbol{Z}$ is a k-U-bisimulation (between $\mathcal{M}$ and $\mathcal{N}$) if whenever $x \; Z_{i+1} \; y$ both of the following hold:

  FORTH U. for all $x' \geq x$ there exists $y' \geq y$ and a function $\xi\colon [y, y'] \to [x, x']$ such that every $z \in [y, y']$ satisfies $\xi(z) \; Z_i \; z$ and $\xi(z) = x'$ if and only if $z = y'$;

  BACK U. for all $y' \geq y$ there exists $x' \geq x$ and a function $\eta\colon [x, x'] \to [y, y']$ such that every $z \in [x, x']$ satisfies $z \; Z_i \; \eta(z)$ and $\eta(z) = y'$ if and only if $z = x'$.

Bounded bisimulations are an essential tool in proving inexpressivity or succinctness results, given that they preserve the truth of formulas of small enough nesting depth.

Lemma 6.3.3 (Kurtonina and de Rijke [148])**.**

1. *Given two* LTL *models $\mathcal{M}$ and $\mathcal{N}$ and a k-$\Box$-bisimulation $\boldsymbol{Z}$ between them, for all formulas $\varphi \in \mathcal{L}_{\bigcirc\Box}$ and for all $\langle x, y\rangle \in Z_i$, if φ has $\bigcirc$-depth at most k and $\Box$-depth at most i then $\mathcal{M}, x \Vdash \varphi$ if and only if $\mathcal{N}, y \Vdash \varphi$.*
2. *Given two* LTL *models $\mathcal{M}$ and $\mathcal{N}$ and a k-*U*-bisimulation $\boldsymbol{Z}$ between them, for all formulas $\varphi \in \mathcal{L}_{\bigcirc\mathsf{U}}$ and for all $\langle x, y\rangle \in Z_i$, if φ has $\bigcirc$-depth at most k and* U*-depth at most i then $\mathcal{M}, x \Vdash \varphi$ if and only if $\mathcal{N}, y \Vdash \varphi$.*

In Section 8.3 we use Lemma 6.3.3 to show that certain legal properties are hard or impossible to define in certain fragments of LTL.

6.4 Monadic logic

Both First-order and Second-order Logic are undecidable. However, restricting their language to the monadic fragment interpreted over the natural numbers leads to decidable fragments [162]: Monadic First-order Logic (MFO) and Monadic Second-order Logic (MSO), respectively. These logics are of interest because MFO is expressively equivalent to LTL [142] and most of the systems used for model checking are fragments of MSO.

Let us define the languages of MSO and MFO. A term is either the constant 0, a variable $x \in \mathbb{V}$ (where $\mathbb{V}$ is a fixed set of first-order variables), or $S(t)$ (where t is some term). The language $\mathcal{L}^1_{\forall\forall}$ is defined as follows:

$$\mathcal{L}^1_{\forall\forall} ::= \bot \mid P(t) \mid t = t \mid t < t \mid \varphi \to \varphi \mid \forall\, x\, \varphi \mid \forall\, P\, \varphi,$$

where $x \in \mathbb{V}$ is a first-order variable, t is a term, and $P \in \mathbb{P}$ is a second-order variable ($\mathbb{P}$ being a fixed set of second-order variables). Note that all second-order variables are unary, i.e. the expression $P(t)$ is a valid formula but the expression $P(t, u)$ is not. We define the other Boolean connectives as well as $\exists\, x\, \varphi$ and $\exists\, P\, \varphi$ as standard abbreviations, and define $\mathcal{L}^1_{\forall}$ as the sub-language of $\mathcal{L}^1_{\forall\forall}$ that does not allow quantification over elements of $\mathbb{P}$.

Similarly to the semantics for LTL, we interpret formulas of $\mathcal{L}^1_{\forall\forall}$ in the natural numbers and reduce a model to a function $J : \mathbb{P} \to \wp(\mathbb{N})$. Furthermore, free first-order variables are interpreted according to assignments $g : \mathbb{V} \to \mathbb{N}$. As in Section 2.5.1, we write $g \sim_x h$ when $g(y) = h(y)$ for every $y \neq x$, and extend this notion to models, writing $J \sim_P J'$ when $Q^J = Q^{J'}$ for every $Q \neq P$. Finally, we extend any assignment g to terms by setting $g(0) := 0$ and $g(S(t)) := g(t) + 1$. The satisfaction relation is defined as follows:

- $J \not\Vdash^g \bot$;
- $J \Vdash^g P(t)$ if and only if $g(t) \in P^J$;
- $J \Vdash^g t = u$ if and only if $g(t) = g(u)$;
- $J \Vdash^g t < u$ if and only if $g(t) < g(u)$;
- $J \Vdash^g \varphi \to \psi$ if and only if either $J \not\Vdash^g \varphi$ or $J \Vdash^g \psi$;
- $J \Vdash^g \forall\, x\, \varphi$ if and only if for all $h \sim_x g$ we have $J \Vdash^h \varphi$;
- $J \Vdash^g \forall\, P\, \varphi$ if and only if for all $J' \sim_P J$ we have $J' \Vdash^g \varphi$.

As before, a formula φ is said to be satisfiable over a set of models Ω is there are $J \in \Omega$ and $g : \mathbb{V} \to \mathbb{N}$ such that $J \Vdash^g \varphi$ (also written $J \Vdash^g_{\mathsf{MSO}} \varphi$ to distinguish it from other satisfiability notions). Furthermore, if $J \Vdash^g \varphi$ for every assignment g, we say that φ is valid in J and write $J \vDash \varphi$. If φ is valid in J for every model $J \in \Omega$, we say that φ is valid in Ω.

Monadic Second-order Logic (MSO) is the language $\mathcal{L}^1_{\forall\forall}$ endowed with this semantics. Monadic First-order Logic (MFO) is MSO restricted to $\mathcal{L}^1_{\forall}$.

The semantics of LTL and MSO (and in particular MFO) are similar and can be unified to provide a notion of equivalence between sublanguages of $\mathcal{L}_{\mathsf{OU}}$ and $\mathcal{L}^1_{\forall\forall}$. Towards this goal, we identify the sets of symbols PROP and $\mathbb{P}$ so that an LTL model $J : \textsc{Prop} \to \wp(\mathbb{N})$ can be seen as an MSO model $J : \mathbb{P} \to \wp(\mathbb{N})$ and vice-versa. Given any natural number n, let $g_n : \mathbb{V} \to \mathbb{N}$ be the assignment such that $g_n(x) := n$ for every $x \in \mathbb{V}$. Then, a formula $\varphi \in \mathcal{L}_{\mathsf{OU}}$ is said to be equivalent to a formula $\psi \in \mathcal{L}^1_{\forall\forall}$ over a set of models Ω when for every $J \in \Omega$ and $n \in \mathbb{N}$ we have $J, n \Vdash_{\mathsf{LTL}} \varphi$ if and only if $J \Vdash^{g_n}_{\mathsf{MSO}} \psi$.

With this in mind, we may regard MFO as a temporal logic via the following result.

Theorem 6.4.1 (Kamp [142], see also Rabinovich [186])**.** *Let Ω be any set of models. For every $\varphi \in \mathcal{L}_{\mathsf{OU}}$ there is $\psi \in \mathcal{L}^1_{\forall}$ such that φ and ψ are equivalent over Ω. Conversely, for every $\psi \in \mathcal{L}^1_{\forall}$ with one free variable, there is $\varphi \in \mathcal{L}_{\mathsf{OU}}$ such that φ and ψ are equivalent over Ω.*

When discussing expressivity, we go back and forth between LTL and MFO depending on which is more convenient for the application at hand.

We now briefly introduce the analytical hierarchy, which is useful as a way to distinguish particular fragments of MSO. It is defined analogously to the arithmetical hierarchy (Definition 2.2.1), distinguishing formulas by their second-order quantifiers. It is typically defined for the full (i.e., not monadic) second order language.

Definition 6.4.2 (Analytical hierarchy, $\Pi^1_n, \Sigma^1_n, \Delta^1_n$)**.** A formula is Δ^1_0 if it has no second-order quantifiers. Then the classes of formulas Σ^1_n and Π^1_n are defined as the smallest classes such that:

- $\Sigma^1_0 := \Pi^1_0 := \Delta^1_0$;
- if $\varphi \in \Sigma^1_n$ then $\forall P\, \varphi \in \Pi^1_{n+1}$;
- if $\varphi \in \Pi^1_n$ then $\exists P\, \varphi \in \Sigma^1_{n+1}$.

Furthermore, we say that a formula is Π^1_n in T (respectively Σ^1_n in T) if it is equivalent over T to some formula ψ such that $\psi \in \Pi^1_n$ (respectively $\psi \in \Sigma^1_n$). A formula is Δ^1_n in T if it is simultaneously Π^1_n in T and Σ^1_n in T. The theory T is often omitted when clear from context.

7 Interpreting regulations

Contents

7.1 Introduction

Legislation is often intended to leave room for various interpretations and applications of the law, while mathematical definitions and algorithms disallow ambiguity by design. However, the regulations considered here prescribe algorithms, and as such we believe they should be taken as prose descriptions of such mathematical constructs rather than as deliberately ambiguous statements. We still find many ambiguities even in such cases, perhaps unavoidable due to the format of the regulations. They are still regrettable in that the developers implementing these algorithms must make possibly unnoticed and unexamined decisions on specific implementation details.

In this chapter we show how easily overlooked subtleties in both Regulation (EU) 2016/799 [90] and Regulation (EC) 561/2006 [88] can in principle lead to drastic differences in results, possibly interpreting legal sequences of activities as illegal or vice-versa. These observations are published in [10], which itself is based on [99] and [9].

7.2 From seconds to minutes

Tachographs record the driver's activity on a second-by-second resolution. One of the goals of Regulation (EU) 2016/799 [90] is to specify how to translate those records into a minute-by-minute resolution, more amenable to interpretation by Regulation (EC) 561/2006 [88]. The idea is that activities lasting only a small number of seconds amount to noise (Requirement (52)). Furthermore, single minutes of something other than DRIVING in between periods of DRIVING are to be interpreted as DRIVING as well (Requirement (51)).

In this section we provide explicit interpretations for Requirements (51) and (52) and discuss the consequences of considering them in the order in which they appear or in the opposite order, as well as what happens when they are considered more than once each. We start by defining the concepts of second and minute labelings.

Seconds and minutes are represented by the integers $\mathbb{Z}$. A labeling is a function assigning an activity $a \in \mathbb{A}$ to each number $i \in \mathbb{Z}$, where:

$$\mathbb{A} := \{\text{DRIVING}, \text{REST}, \text{AVAILABILITY}, \text{WORK}, \text{UNKNOWN}\}.$$

In practice, labelings have a finite subset of $\mathbb{Z}$ as domain, but those can be extended to $\mathbb{Z}$ by assigning UNKNOWN to the remaining numbers, so we take the domain of any labeling to be $\mathbb{Z}$.

When $\mathbb{Z}$ is interpreted as a set of seconds we use the symbol $\mathcal{S}$ for labelings and say they are second labelings. Similarly, when it is interpreted as a set of minutes we use the symbol $\mathcal{M}$ for labelings and say they are minute labelings. We only care about extensional equality between labelings, and use the $=$ symbol to denote it in this context.

We interpret the requirements as functions to be applied to minute and possibly second labelings, obtaining minute labelings. Requirement (51) is interpreted as follows.

Definition 7.2.1 (R_{51})**.** Given a minute labeling $\mathcal{M} : \mathbb{Z} \to \mathbb{A}$, the labeling after applying Requirement (51) is defined for each $i \in \mathbb{Z}$ as:

$$\mathrm{R}_{51}(\mathcal{M})(i) := \begin{cases} \textsf{DRIVING} & \text{if } \mathcal{M}(i-1) = \mathcal{M}(i+1) = \textsf{DRIVING} \\ \mathcal{M}(i) & \text{otherwise.} \end{cases}$$

Note that after applying R_{51} to a minute labeling, either there is no change or there are more minutes labeled as DRIVING and less minutes labeled as any other activity. Furthermore, consecutive applications of R_{51} do not lead to new minute labelings, as the following lemma shows.

Lemma 7.2.2. *Requirement (51) is idempotent, i.e.,* $\mathrm{R}_{51}(\mathrm{R}_{51}(\mathcal{M})) = \mathrm{R}_{51}(\mathcal{M})$ *for any minute labeling* $\mathcal{M}$*.*

Proof. Towards a contradiction, let i be such that $\mathrm{R}_{51}(\mathrm{R}_{51}(\mathcal{M}))(i) \neq \mathrm{R}_{51}(\mathcal{M})(i)$. Then by the definition of R_{51}:

$$\mathrm{R}_{51}(\mathcal{M})(i-1) = \mathrm{R}_{51}(\mathcal{M})(i+1) = \textsf{DRIVING}; \text{ and}$$
$$\mathrm{R}_{51}(\mathrm{R}_{51}(\mathcal{M}))(i) = \textsf{DRIVING}.$$

Since we are assuming the second application of R_{51} changed the labeling at i, we conclude that $\mathrm{R}_{51}(\mathcal{M})(i) \neq$ DRIVING and consequently $\mathcal{M}(i) \neq$ DRIVING too. Furthermore, either $\mathcal{M}(i-i) \neq$ DRIVING or $\mathcal{M}(i+1) \neq$ DRIVING.

Assume without loss of generality that $\mathcal{M}(i-1) \neq$ DRIVING. Note that $\mathrm{R}_{51}(\mathcal{M})(i-1) =$ DRIVING, so in applying R_{51} the first case must have been triggered, i.e., it must have been that $\mathcal{M}(i-2) = \mathcal{M}(i) =$ DRIVING. This contradicts our previous observation that $\mathcal{M}(i) \neq$ DRIVING. □

Clearly, Requirement (51) is not enough to translate a second labeling into a minute labeling, since it doesn't so much as mention or interact with second labelings. It is merely a way to "clean up" a minute labeling by interpreting short periods of non-DRIVING between periods of DRIVING as DRIVING too. The only requirement translating second labelings into minute labelings is Requirement (52), which we interpret as follows.

Definition 7.2.3 (R_{52})**.** Given a shift $d \in \mathbb{Z}$, a second labeling $\mathcal{S} : \mathbb{Z} \to \mathbb{A}$ and a minute labeling $\mathcal{M} : \mathbb{Z} \to \mathbb{A}$, the labeling after applying Requirement (52) is defined for each $i \in \mathbb{Z}$ as:

$$\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})(i) := \begin{cases} A(d, \mathcal{S}, i) & \text{if } \mathcal{M}(i) = \textsf{UNKNOWN} \\ \mathcal{M}(i) & \text{otherwise,} \end{cases}$$

where $A(d, \mathcal{S}, i)$ is the activity $a \in \mathbb{A}$ such that $\mathcal{S}^{-1}(a) \cap [d + 60i, d + 60(i+1))$ contains the rightmost interval of maximal length.

In other words, $A(d, \mathcal{S}, i)$ is the activity of the longest constant-activity interval among all constant-activity intervals contained in $[d + 60i, d + 60(i+1))$, or the activity of the rightmost such interval if there is a tie.

The shift d appearing in the definition of R_{52} is meant to represent the calendar. Different calendars differ from each other by a small amount of seconds depending on whether or not leap seconds are considered. For example, as of July 2023, there is a difference of 27 seconds between UTC [138] and Unix time [132], so one could take $d = 27$ to see what would happen if using UTC instead of Unix (or $d = -27$ if starting with UTC).[1] We discuss calendars with and without leap seconds in more detail in Section 11.1. In this section we always use $d = 0$, but in Section 7.3 we show how varying d can drastically change the result of applying R_{52}.

It is easy to check that R_{52} is idempotent.

Lemma 7.2.4. *Requirement (52) is idempotent, i.e.,* $\mathrm{R}_{52}(d, \mathcal{S}, \mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})) = \mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})$ *for any shift* d*, second labeling* $\mathcal{S}$ *and minute labeling* $\mathcal{M}$*.*

Proof. Towards a contradiction, let i be such that $\mathrm{R}_{52}(d, \mathcal{S}, \mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M}))(i) \neq \mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})(i)$. Then by definition of R_{52} it must be that $\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})(i) = \text{UNKNOWN}$ and $\mathrm{R}_{52}(d, \mathcal{S}, \mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M}))(i) = A(d, \mathcal{S}, i)$, with $A(d, \mathcal{S}, i) \neq \text{UNKNOWN}$.

On the other hand, the only situation in which $\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{M})(i) = \text{UNKNOWN}$ is when both $\mathcal{M}(i) = \text{UNKNOWN}$ and $A(d, \mathcal{S}, i) = \text{UNKNOWN}$. This is a contradiction, as $A(d, \mathcal{S}, i)$ cannot be both UNKNOWN and not UNKNOWN. ☐

In order to discuss the translation of a second labeling into a minute labeling, we introduce the unknown minute labeling, upon which we can build with both R_{51} and R_{52}.

Definition 7.2.5 ($\mathcal{U}$)**.** The unknown labeling is the minute labeling $\mathcal{U} : \mathbb{Z} \to \mathbb{A}$ such that $\mathcal{U}(i) :=$ UNKNOWN for all $i \in \mathbb{Z}$.

Requirements (51) and (52) appear in that order in Regulation (EU) 2016/799 [90], which is puzzling. As previously stated, Requirement (51) does not mention how to translate second labelings into minute labelings, and furthermore does not change the unknown labeling, i.e., $\mathrm{R}_{51}(\mathcal{U}) = \mathcal{U}$. However, if applied after R_{52} there is the possibility for change, as the following example illustrates.

Example 7.2.6. Let $\mathcal{S}$ be the second labeling that assigns DRIVING to all seconds within even minutes and REST to all seconds within odd minutes. Then for any $i \in \mathbb{Z}$:

$$\mathrm{R}_{52}(0, \mathcal{S}, \mathcal{U})(i) = \begin{cases} \text{DRIVING} & \text{if } i \text{ is even} \\ \text{REST} & \text{otherwise,} \end{cases}$$

and $\mathrm{R}_{51}(\mathrm{R}_{52}(0, \mathcal{S}, \mathcal{U}))(i) = \text{DRIVING}$.

On the other hand, R_{52} applied to $\mathcal{U}$ and followed by R_{51} leads to a stable configuration, where a further application of either requirement does not produce any changes.

Lemma 7.2.7. *Let* $\mathcal{S}$ *be a second labeling and* d *be a shift. Furthermore, let* $\mathrm{R}_{51\text{-}52}$ *be a notation for* $\mathrm{R}_{51}(\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{U}))$*. The following both hold:*

1. $\mathrm{R}_{51}(\mathrm{R}_{51\text{-}52}) = \mathrm{R}_{51\text{-}52}$*;*

2. $\mathrm{R}_{52}(d, \mathcal{S}, \mathrm{R}_{51\text{-}52}) = \mathrm{R}_{51\text{-}52}$.

[1] Changing the calendar is more complicated than this, since each new leap second changes the shift. For example, the difference between UTC and Unix time is only of 25 seconds if considering records during the year 2014. Notwithstanding, this simplified implementation of R_{52} suffices for our purposes.

Proof. The first observation follows from the idempotency of R_{51} (Lemma 7.2.2). For the second, note that R_{52} is the identity over the non-UNKNOWN part by definition. If $\mathrm{R}_{51\text{-}52}(i) = \text{UNKNOWN}$ for some i, then R_{51} didn't change what happens at i and so $\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{U})(i)$ was UNKNOWN already, which implies that $A(d, \mathcal{S}, i) = \text{UNKNOWN}$. Then, since $\mathrm{R}_{52}(d, \mathcal{S}, \mathrm{R}_{51\text{-}52})(i) = A(d, \mathcal{S}, i)$, it follows that it is UNKNOWN too, which concludes the proof. □

Note that since $\mathrm{R}_{51}(\mathcal{U}) = \mathcal{U}$, starting with an application of R_{51} followed by R_{52} is the same as starting with R_{52}, and thus by Lemma 7.2.7 and the idempotency of both requirements we conclude that all orderings of applications of both R_{51} and R_{52} to $\mathcal{U}$ converge on the same labeling, namely the one obtained by starting with R_{52} followed by R_{51}.

Why then does Requirement (51) precede Requirement (52) in Regulation (EU) 2016/799 [90]? This is especially notorious since Requirement (52) references Requirement (51). If we trace these requirements back in time, we find that there was another sentence immediately preceding Requirement (51), which was removed by an amendment in Commission Regulation (EU) 1266/2009 [89], changing the global meaning of the excerpt. It is possible that the oddness of the resulting order was not noticed or alternatively not deemed relevant enough to change.

Since the internal functioning of commercial tachographs is subject to proprietary software restrictions, we cannot freely check the implementation of the regulation that they have chosen. However, Guretruck S.L. conducted experimental tests and deduced from them that commercial tachographs apply Requirement (52) followed by Requirement (51), which is the only ordering that makes sense, although it is not the one suggested by Regulation (EU) 2016/799 [90]. They also disregard leap seconds, which are part of the UTC time standard prescribed by Regulation (EU) 2016/799 [90]. We discuss the latter issue in the next section.

7.3 Time shifts

In this section we show that different calendar shifts d can lead to completely different minute labelings while still being in accordance with Regulation (EU) 2016/799 [90].

Lemma 7.3.1. *Let d be such that $1 \leq d \leq 59$ and $\mathcal{M}$ and $\mathcal{M}'$ be any two minute labelings. Then there exists a second labeling $\mathcal{S}$ such that $\mathrm{R}_{52}(0, \mathcal{S}, \mathcal{U}) = \mathcal{M}$ and $\mathrm{R}_{52}(d, \mathcal{S}, \mathcal{U}) = \mathcal{M}'$.*

Proof. We can assume without loss of generality that $d \leq 30$, for otherwise $\mathcal{M}$ and $\mathcal{M}'$ could be swapped. For each $s \in \mathbb{Z}$, let $\mathcal{S}(s)$ be as follows:

$$\mathcal{S}(s) := \begin{cases} \mathcal{M}'(\lfloor s/60 \rfloor - 1) & \text{if } s \bmod 60 = 0 \\ \mathcal{M}(\lfloor s/60 \rfloor) & \text{if } 0 < s \bmod 60 \leq 30 \\ \mathcal{M}'(\lfloor s/60 \rfloor) & \text{if } s \bmod 60 > 30. \end{cases}$$

Recall that $\mathrm{R}_{52}(0, \mathcal{S}, \mathcal{U})(i) = A(0, \mathcal{S}, i)$, which is the activity of the rightmost maximal interval of constant activity in $[60i, 60i + 60)$. By the definition of $\mathcal{S}$, in the interval $[60i, 60i + 60)$ there is a first second $60i$ of activity $\mathcal{M}'(i-1)$, then 30 seconds of activity $\mathcal{M}(i)$, and then 29 seconds of activity $\mathcal{M}'(i)$. Therefore, $\mathrm{R}_{52}(0, \mathcal{S}, \mathcal{U})(i) = \mathcal{M}(i)$.

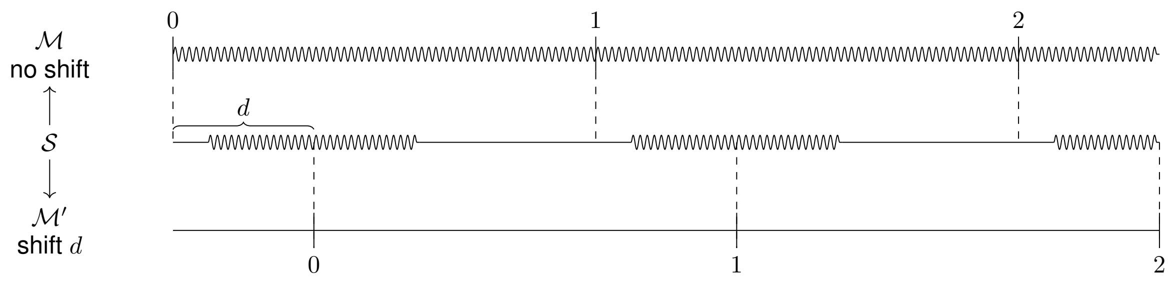

Figure 7.1: A second labeling $\mathcal{S}$ such that Requirement (52) gives rise to the minute labeling $\mathcal{M}$ in the absence of any shift and to $\mathcal{M}'$ in the presence of a shift d. The serpentine and smooth lines represent different activities.

Now for the other condition: $\mathrm{R}_{52}(d,\mathcal{S},\mathcal{U})(i) = A(d,\mathcal{S},i)$, which is the activity of the longest interval in $[d+60i, d+60i+60)$. By the definition of $\mathcal{S}$, in that interval there are first $31-d$ seconds of activity $\mathcal{M}(i)$, then 30 seconds of $\mathcal{M}'(i)$, and then $d-1$ seconds of $\mathcal{M}(i+1)$. If $d > 1$, then the activity with the longest interval is $\mathcal{M}'(i)$. If $d = 1$, then there is a draw between $\mathcal{M}(i)$ and $\mathcal{M}'(i)$, but the latest is $\mathcal{M}'(i)$. In any case, $\mathrm{R}_{52}(d,\mathcal{S},\mathcal{U})(i) = \mathcal{M}'(i)$.

This construction can be visualized in Figure 7.1. □

Informally, Lemma 7.3.1 states that, given two minute labelings and a shift, there is a possible second labeling which leads R_{52} to the first minute labeling in the calendar without any shift and to the second minute labeling in the shifted calendar. In particular, there are sequences of activities that lead to a minute labeling that is 100% DRIVING or 100% REST, depending on the shift. Even though such a drastic difference is unlikely in practice, Guretruck S.L. conducted experimental tests with real-world driver data and found that the minute labelings vary up to 8% in DRIVING time depending on which calendar is used.

Finally, we observe that Requirement (51) does not solve this problem; it only places some mild restrictions on the labelings $\mathcal{M}$ and $\mathcal{M}'$, as defined below.

Definition 7.3.2 (Feasible minute labeling). A minute labeling $\mathcal{M} : \mathbb{Z} \to \mathbb{A}$ is said to be feasible if $\mathcal{M}(i) = \text{DRIVING}$ for every $i \in \mathbb{Z}$ such that $\mathcal{M}(i-1) = \mathcal{M}(i+1) = \text{DRIVING}$.

Feasible minute labelings do not change upon the application of R_{51}, which is easily checked with the reasoning used in the proof of R_{51}'s idempotency (Lemma 7.2.2). Thus, the following is a straightforward corollary of Lemma 7.3.1.

Corollary 7.3.3. *Let d be a time shift such that $1 \leq d \leq 59$ and $\mathcal{M}$ and $\mathcal{M}'$ be two feasible minute labelings. Then there exists a second labeling $\mathcal{S}$ such that $\mathrm{R}_{51}(\mathrm{R}_{52}(0,\mathcal{S},\mathcal{U})) = \mathcal{M}$ and $\mathrm{R}_{51}(\mathrm{R}_{52}(d,\mathcal{S},\mathcal{U})) = \mathcal{M}'$.*

We have seen that the translation of second labelings into minute labelings proposed by Regulation (EU) 2016/799 [90] is especially fragile, since it depends on which calendar is being used. The choice of calendar is also prescribed by the regulations, but is often not heeded (leap seconds are ignored when they shouldn't be, see Section 11.1). Even if that weren't the case, this is an example where the precise moment at which a driver starts their work is relevant in the eyes of the regulations, even though it is our opinion that starting a number of seconds later or earlier should not matter.

With this we conclude our discussion of issues with labelings, and in the sequel consider the problem of verifying that a given minute labeling complies with the regulations, assuming it has been suitably obtained from the original tachograph data.

7.4 Weekly rest periods

Recall from Section 6.2 that according to Regulation (EC) 561/2006 [88] weekly rest periods are periods of uninterrupted rest of at least 24 hours (reduced) or 45 hours (regular). A week in the sense of §4(i) is the period between 00:00 on a Monday and 24:00 on the following Sunday, and each week in that sense must contain at least one weekly rest period. Furthermore, there can be no consecutive reduced weekly rest periods and any reduced weekly rest period must be compensated during the following three weeks. We discuss the assignment of weekly rest periods in Section 7.4.1 and the compensations of reduced weekly rest periods in Section 7.4.2.

7.4.1 Assignments

Consider the following six weeks of weekly rest periods: the driver starts resting on Saturday of the first week at 00:00 and retakes their activity on the following Monday at 20:00, thus resting for 68 consecutive hours. Then, until the fourth week, the driver periodically starts their weekly rest on Sunday at 00:00 and retakes their activity on the following Tuesday at 20:00, again resting for 68 consecutive hours. During the sixth week they only rest for 45 consecutive hours, from Monday at 20:00 to the following Wednesday at 17:00. This is depicted in Figure 7.2.

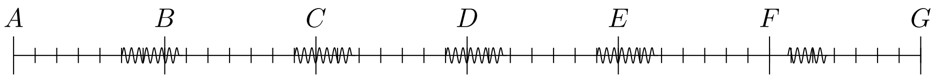

Figure 7.2: Six weeks of activity of a hypothetical driver. Each letter-divided segment denotes a week and the smaller segments denote days. Each serpentine line denotes a weekly rest period of 68 hours except the last one, which lasts only for 45 hours.

Since all but the last of these rest periods fall between two weeks, the application of §8.9 must determine which week each rest period pertains to. This procedure should find a legal placement if at all possible.

There is no legal assignment for this example, because the are six weeks and only five rest periods. In principle, a long rest period could be interpreted as two consecutive weekly rest periods, but since the longest rest periods available are only 68 hours long, any split would lead to either two consecutive reduced weekly rest periods or to a regular weekly rest period and at most 23 hours of rest, which is not enough to count as a weekly rest period. The former would go against §8.6, and the latter is not useful at all. This situation is a bit puzzling, since on average the driver rests well over the minimum required time per week and never spends more than six 24 hour periods without a weekly rest period, which is explicitly mentioned as a requirement.

Here Regulation (EC) 561/2006 [88] does not pose a logical problem, nor is it inconsistently worded. However, the complexity that results from §8.9 coupled with the strict definition of "week" is worthy of note. The situation described in Figure 7.2 can be extended to arbitrarily-many weeks and it is possible that verifying compliance would require checking a large number of week assignments. A more thorough discussion of this issue is provided by del Castillo Tierz [57].

7.4.2 Compensations

Regulation (EC) 561/2006 [88] requires that every reduced weekly rest period be compensated by "an equivalent period of rest taken en bloc before the end of the third week following the week in question" (§8.6) and that "[a]ny rest taken as compensation for a reduced weekly rest period shall be attached to another rest period of at least nine hours" (§8.7). We call these two restrictions together with the requirements that all weeks must have a weekly rest period and that no two consecutive weeks may have reduced weekly rest periods the compensation rules. Checking whether a given driver's activities satisfy the compensation rules is not always trivial, and can even be non-local.

To illustrate, we present an arbitrarily long sequence of weeks s and weeks A and B such that both $A \mid s$ and $s \mid B$ satisfy the compensation rules but $A \mid s \mid B$ does not (where $\cdot \mid \cdot$ denotes week concatenation).

When describing weeks in this section, we focus on the length of the Maximum Period of Consecutive Rest (MPCR) within each week and not on its placement within the week. We disregard shorter periods of rest, including daily rest periods (also part of Regulation (EC) 561/2006 [88] but not explored in this thesis). Including those would increase the opportunities for compensation but ultimately not be enough to avoid the issue presented here.

Definition 7.4.1. We define the following weeks by the lengths of their Maximum Period of Consecutive Rest (MPCR):

- A has an MPCR of 44 hours;
- B has an MPCR of 45 hours;
- C_i has an MPCR of 45 hours for each i;
- D has an MPCR of 24 hours.

Thus, in a vacuum, both A and D have reduced weekly rest periods while B and each C_i have regular weekly rest periods.

We now define the sequences used to exemplify the non-locality of the compensation rules.

Definition 7.4.2. Fix a natural number n. Then:

- $s := C_0 \mid \cdots \mid C_n \mid D$;
- $s_A := A \mid s$;
- $s_B := s \mid B$;

- $s_{AB} := A \mid s \mid B$.

Our interpretation of the compensation rules is that time spent compensating a reduced weekly rest period cannot simultaneously count for the weekly rest period of its week, as otherwise the concept of compensation would be meaningless. Thus, if the length of the MPCR is 45 hours in a given week and one of those hours is used as compensation, the weekly rest period of that week has a length of 44 hours and must be compensated. We further assume that future weeks for which we do not have records (for example, weeks after B in s_{AB}) are spent fully resting, which is as favorable for the compensation rules as possible (following the *in dubio pro reo* principle).

Lemma 7.4.3. *If $n \geq 2$, both s_A and s_B satisfy the compensation rules, but s_{AB} does not.*

Proof. Let w_i^r represent the $(i+1)$-th week of sequence r, so for example $w_0^{s_A} = w_0^{s_{AB}} = A$ and $w_0^{s_B} = C_0$.

Week $w_0^{s_A}$ requires a compensation of one hour. We distinguish the case where n is even from the case where it is odd. If n is even, we use week $w_2^{s_A}$ to compensate week $w_0^{s_A}$. This makes $w_2^{s_A}$ require a compensation, which we assign to week $w_4^{s_A}$ (for $n \geq 4$), and so on. In general, even weeks of s_A require a compensation that can be achieved during the following even week, with the exception of week $w_n^{s_A}$, since the following even week is week D. Week D cannot be used to compensate anything because its MPCR is at minimum length for a reduced weekly rest period. However, in s_A that is also the final week on which we have data, so the compensation of week $w_n^{s_A}$ can be achieved during week $w_{n+3}^{s_A}$, assumed to be a full week of rest. Similarly, the reduced weekly rest period of week D can be compensated during either of the following three full weeks of rest. See Figure 7.3 for an illustration of the $n = 4$ case.

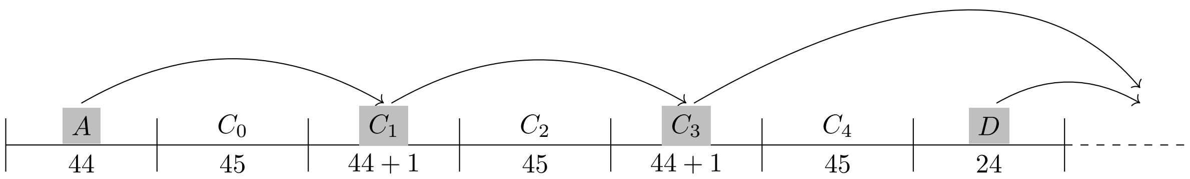

Figure 7.3: The sequence s_A with $n = 4$ together with the compensation scheme proposed during the proof of Lemma 7.4.3 for the even n case. Shaded weeks are the ones with ultimately reduced weekly rest periods.

If n is odd, the reasoning is similar except week $w_0^{s_A}$ is compensated during week $w_3^{s_A}$ instead. Each subsequent odd week then requires a compensation, which can be achieved during the following odd week unless it is week D. As in the even case, such a compensation can be done during week $w_{n+3}^{s_A}$. Note that this argument does not hold for $n = 1$, because then week $w_0^{s_A}$ would only be compensated during week $w_4^{s_A}$, which is too late. See Figure 7.4 for an illustration of the $n = 3$ case.

The only reduced weekly rest periods of s_A in this compensation scheme occur on even weeks when n is even and on $w_0^{s_A}$ or odd weeks ($w_1^{s_A}$ excluded) when n is odd, guaranteeing that they are not consecutive, and all required compensations occur within the required three following weeks. Thus, s_A satisfies the compensation rules.

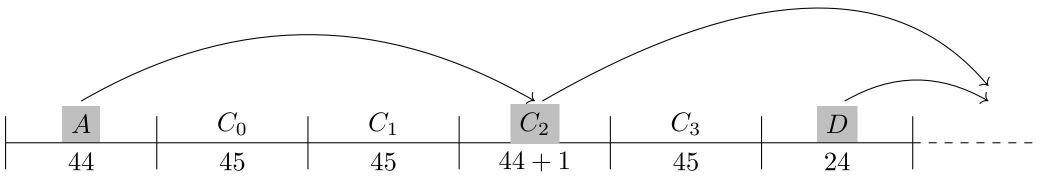

Figure 7.4: The sequence s_A with $n = 3$ together with the compensation scheme proposed during the proof of Lemma 7.4.3 for the odd n case. Shaded weeks are the ones with ultimately reduced weekly rest periods.

It is easy to check that s_B satisfies the compensation rules too, since the only week with a reduced weekly rest period is the next-to-last one, which can be compensated during the weeks of full rest assumed to exist after the end of s_B. See Figure 7.5 for an illustration of the $n = 2$ case.

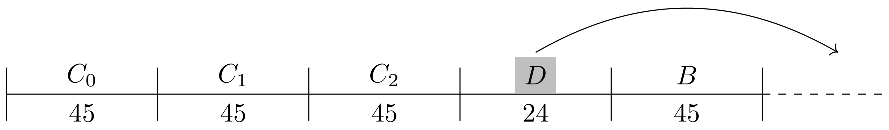

Figure 7.5: The sequence s_B with $n = 2$ together with the compensation scheme proposed during the proof of Lemma 7.4.3. The only week with a reduced weekly rest period is D.

As for s_{AB}, we show how none of the available weeks (other than the ones after B) can be used to compensate even a single hour. Let $l := n + 3$ be the index of B, the last week of s_{AB}. None of $w_l^{s_{AB}}$, $w_{l-1}^{s_{AB}}$ or $w_{l-2}^{s_{AB}}$ can be used to compensate anything, because they are either D, which is at its minimum length of MPCR, or adjacent to a week with a reduced weekly rest period. Then $w_{l-3}^{s_{AB}}$ can also not be used to compensate anything, since it would become reduced itself and in need of compensation during one of the following three weeks, none of which allow compensations. Iterating this argument, we see that none of the weeks of s_{AB} can be used for compensations. However, $w_0^{s_{AB}}$ needs to be compensated and is too far away from the full rest at the end of s_{AB}, which concludes the argument that s_{AB} cannot satisfy the compensation rules. $\square$

We have shown that it is possible to satisfy the compensation rules for an arbitrarily long stretch of time (s_A) only to violate them during a single final week (B). That by itself would not be so surprising, except that the violation does not in fact occur during the new week, as the correctness of s_B shows: it occurs due to the combination of both A and B. This implies that a driver's record would in principle need to be checked in its entirety (since they became a driver under Regulation (EC) 561/2006 [88]) in order to confirm that a given sequence satisfies the compensation rules. This is not feasible for long-term drivers.

8 Expressibility and non-expressibility

Contents

8.1 Introduction

We saw in Section 7.4 how the requirements of Regulation (EC) 561/2006 [88] relating to weekly rest periods have interpretations that lead to unexpected consequences. In this chapter, we focus on representing our interpretations of those same requirements within formal mathematical systems.

Since these legal requirements are checked via automated methods, it is desirable to represent them within simple systems with good complexity properties. A driving record is essentially a sequence of consecutive events, so well-understood logics with ontologies relating to discrete moments in time, such as LTL, and stronger logics that can be related to LTLs such as MSO, are good candidates.

Thus, we start by showing how the two weekly rest period problems (choosing which week a certain rest period is assigned to and choosing in which week to compensate a reduced weekly rest period) can be expressed in the Σ^1_1 fragment of MSO. Then we show that neither is expressible in $\mathcal{L}_{\bigcirc\Box}$ and that the compensation rules are not expressible in $\mathcal{L}_{\bigcirc\mathsf{U}}$ via a formula with a small U-depth.

The upshot is that evaluating whether a given driver's record complies with the relevant part of Regulation (EC) 561/2006 [88] can be reduced to model checking over the Σ^1_1 fragment of MSO. This fragment is well-behaved, since the truth of a given Σ^1_1 MSO formula in a given model is trivially reduced to satisfiability of a MFO formula, which itself can be reduced to satisfiability in LTL, for which many solvers are available [128]. Nevertheless, satisfiability in LTL is PSPACE-complete [192] and moreover the translation of MFO into LTL is non-elementary in the worst case, so we feel that such laws should instead be representable in a computationally tame fragment of MSO.

The contents of this chapter were published in [9] and [10].

8.2 Expressibility

In this section we show how parts of Regulation (EC) 561/2006 [88] pertaining to weekly rest periods, already explored in Section 7.4, can be represented within MSO. Note that the articles in question allow for some leeway in interpretation and thus there are possibly readings different from those we propose. We also make a few simplifying assumptions for the sake of exposition.

We discussed how tachograph data is recorded second-by-second and then translated into a minute-by-minute format in Section 7.2. However, here we consider hour labelings for the sake of readability. Recall that the set of activities is $\mathbb{A} := \{\mathsf{DRIVING}, \mathsf{REST}, \mathsf{AVAILABILITY}, \mathsf{WORK}, \mathsf{UNKNOWN}\}$. Each activity is regarded as a propositional variable. Then a labeling $\mathcal{H}$ (in the sense of Section 7.2) with domain $\mathbb{N}$ may be viewed as an LTL model by setting $a^{\mathcal{H}} := \{n \in \mathbb{N} : \mathcal{H}(n) = a\}$ for $a \in \mathbb{A}$, and $p^{\mathcal{H}} := \emptyset$ for all other propositional variables p. We use $\mathbb{N}$ as domain so that Kamp's theorem (Theorem 6.4.1) is available. It is also possible to work with $\mathbb{Z}$ as in Chapter 7, but an analogue of Kamp's theorem in that setting would require additional modalities to represent the past, which would make some upcoming proofs more tedious. A rest period is any interval whose activity is always REST.

We introduce a predicate symbol WEEK that holds on the first hour of each Monday. This condition can be treated model-theoretically, i.e. models are assumed to be equipped with a correct valuation for WEEK, or syntactically via the axiom:

$$\text{WEEK} \wedge \Box(\text{WEEK} \rightarrow (\bigcirc\Box^{<167}\neg\text{WEEK} \wedge \bigcirc^{168}\text{WEEK})),$$

assuming that the model begins on the first hour of a Monday. LTL models satisfying this formula at zero are called weekly models. With this in mind, we illustrate how a simplified version of the relevant parts of Regulation (EC) 561/2006 [88] can be represented.

8.2.1 Assignments

Our goal in this section is to show how assigning weekly rest periods to weeks can be accomplished within MFO, and hence LTL. The relevant parts of Regulation (EC) 561/2006 [88] were already explored in Section 7.4.1.

Since the required amount of hours in a weekly rest period depends on the context of the surrounding weeks (see Section 7.4.2), we use a parameter d as the minimum amount of hours required for a rest period to be considered a weekly rest period. This is a considerable but reasonable simplification of the overall regulation in this context where we are only interested in studying the part related to assigning weekly rest periods to weeks. Thus, we wish to describe sequences of weeks that allow a rest period assignment of at least d hours to each week such that each period intersects the week it is assigned to.

In this section we assume that variables range over weeks. This can clearly be established in MFO, as a week can be identified with its starting point, already marked by the predicate WEEK. Furthermore, we represent the successor of a week w as $w + 1$ and the predecessor of a week $w > 0$ as $w - 1$.

A week w can be said to have early, late, or internal rest periods, defined as follows.

Definition 8.2.1 (early, late, internal rest periods)**.** An early rest period for week w is a rest period of length at least d that intersects weeks $w - 1$ and w. Similarly, a late rest period for week w is a rest period of length at least d that intersects weeks w and $w + 1$. Finally, an internal rest period for week w is a rest period of length at least d that is disjoint (but possibly contiguous with) any early or late rest periods of w.

We make use of formulas E, L and I such that:

- $E(w)$ if and only if w has an early rest period;
- $L(w)$ if and only if w has a late rest period;
- $I(w)$ if and only if w has an internal rest period.

Clearly, E, I and L are first-order definable (although their definition depends on d). If $d \leq 85$ and $E(w) \wedge L(w)$, the early and late rest periods of w are disjoint. This is because an early rest period has at least one hour on the previous week, so at most $d - 1 \leq 84$ hours of the early rest period can be part of w, and similarly for the late rest period. A week has $2 \cdot 84$ hours, so these two rest periods cannot overlap.

Theorem 8.2.2. *Given $d \in [2, 85]$, there is a formula $\varphi_d \in \mathcal{L}^1_\forall$ such that for any model J, we have $J \models \varphi_d$ if and only if there is an assignment of rest periods such that every week is assigned a rest period of length at least d intersecting that week.*

Proof. Let $\check{E}(w) := E(w) \wedge \neg I(w) \wedge \neg L(w)$, and define $\check{I}(w)$ and $\check{L}(w)$ analogously. Then set:

$$\varphi_d := \forall\, w\, (E(w) \vee I(w) \vee L(w)) \wedge \forall\, w\; \forall\, u\, (w < u \wedge \check{L}(w) \wedge \check{E}(u) \to \exists\, v\, (w < v < u \wedge I(v))).$$

We claim that φ_d holds if and only if there is an assignment such that each week is assigned one rest period of length at least d intersecting that week. First assume that such an assignment exists. Clearly $\forall\, w\, (E(w) \vee I(w) \vee L(w))$ holds, since if w were a counterexample no rest period could be assigned to week w.

Now suppose $w < u$ are such that $\check{L}(w) \wedge \check{E}(u)$, and choose w and u such that $u - w$ is minimal among all such pairs. Note that w must be assigned to its late rest period and u to its early rest period, as these are the only ones available. Let $v \in (w, u]$ be the minimal week not assigned to its late rest period, which must exist because u is such a week. By minimality, $v - 1$ must be assigned to its late rest period, and hence v cannot be assigned to its early rest period (since it is the same as $v - 1$'s late one), and in particular $v \neq u$. However, v must be assigned to some rest period by assumption, and since this rest period is neither early nor late, v must contain some internal rest period, and so $I(v)$ holds.

For the other direction, assume that φ_d holds and define an assignment iteratively on the ordered list of rest periods as follows. Let R be a rest period and suppose that all earlier rest periods have been assigned to some week. If R is internal, assign it to its current week. If R is late for week w and w has not been assigned a rest period yet, assign R to w. Otherwise, assign R to $w + 1$.

We prove by induction that every week is assigned some rest period. Fix u and assume all earlier weeks have an assigned rest period. Note that we have $E(u) \vee I(u) \vee L(u)$ by φ_d. If $I(u)$ or $L(u)$, it is clear that u is assigned some rest period by the algorithm. If $\check{E}(u)$ and u is not assigned a rest period, then the early rest period of u must have been assigned to $u - 1$. Let $w < u$ be minimal such that every $v \in [w, u)$ has a late rest period assigned to it. First note that w must not have an early rest period at all, since otherwise $w > 0$ and either $w - 1$ has had its late rest period assigned to it, contradicting the minimality of w, or else the early rest period of w would have been assigned to the week of w by the algorithm, and then no late rest period would have been assigned to w. Note also that $I(v)$ fails for all $v \in [w, u)$, since any internal rest period is automatically assigned to the current week, again preventing the assignment of a late rest period. We conclude that $\check{L}(w)$ holds and $I(v)$ fails for all $v \in (w, u)$, implying that φ_d does not hold, which is a contradiction. □

8.2.2 Compensations

We now isolate the compensation mechanism from the assignments and analyze it in a similar fashion. We claim that the content of §8.6 admits a representation in $\mathcal{L}^1_{\forall\forall}$ by a Σ^1_1 formula over the class of weekly models.

Recall the compensation rules from Section 7.4.2. A reduced weekly rest period must be compensated in the future, but it may be compensated during the same week as the rest period itself. The compensation must fall entirely within the union of the current week with the subsequent three weeks. We further assume that after compensation each week should have a weekly rest period of exactly 45 hours assigned. This is not an issue since longer rest periods can be partially unassigned. Finally, we assume that each week is assigned a single weekly rest period. Regulation (EC) 561/2006 [88] does not explicitly forbid weeks with two weekly rest periods, but we find this to be the most intuitive interpretation.

Our representation uses second-order variables with the following intended meanings. The notions of early, internal, and late rest periods are as in Definition 8.2.1 except no minimum length d is specified at this point.

- R_E denotes the union of all early rest periods;
- R_I denotes the union of all internal rest periods;
- R_L denotes the union of all late rest periods;
- C_0, C_1, C_2, and C_3 denote periods of compensation such that C_i represents compensations performed during the i-th week after the weekly rest period to be compensated.

We express §8.6 by the formula $\psi_{\S 8.6}$ defined as follows:

$$\psi_{\S 8.6} := \exists R_E \, \exists R_I \, \exists R_L \, \exists C_0 \, \exists C_1 \, \exists C_2 \, \exists C_3 \, \psi^0_{\S 8.6},$$

where $\psi^0_{\S 8.6}$ contains no second-order quantifiers. Note that we can quantify over intervals in MFO, since an interval $[a,b]$ can be identified with its endpoints. Conditions of the form $|X| \geq n$ with fixed n are expressed by stating that there exist distinct $x_1, \dots, x_n$ belonging to X.

The formula $\psi^0_{\S 8.6}$ is a conjunction over formulas expressing the following properties:

- the sets $R_E, R_I, R_L, C_0, C_1, C_2, C_3$ are mutually disjoint and all fully REST;
- both R_E and R_L are unions of non-contiguous intervals of length at least 24 and at most 45;
- for any week w, $R_I \cap w$ is either empty or an interval of length at least 24 and at most 45;
- for any week w, there is a unique maximal interval R_w, the rest period assigned to w, such that $R_w \cap w$ is non-empty and one of the following holds:
  - $R_w \subseteq R_E$, $w > 0$, and R_w intersects $w - 1$;
  - $R_w \subseteq R_I \cap w$;
  - $R_w \subseteq R_L$ and R_w intersects $w + 1$;
- for any week w, R_{w+1} must start no later than $6 \cdot 24$ hours after the end of R_w, and at least one of R_w, R_{w+1} must be a 45 hour interval;

- for any week w, define $C_w := \bigcup_{i=0}^{3}(C_i \cap (w+i))$ such that the following conditions all hold:
  - C_w is an interval disjoint from R_w and it begins after R_w ends;
  - there is a rest interval J such that $C_w \subseteq J$ and $J \setminus C_w$ is an interval of length at least 9;[1]
  - $|R_w \cup C_w| = 45$.

It should be clear that each of these conditions is first-order definable, and hence that $\psi_{§8.6}$ is Σ^1_1. Moreover, some inspection shows that over the class of weekly models, $\psi_{§8.6}$ coincides with our reading of §8.6. Indeed, the variables R_E, R_I, and R_L are used as auxiliary variables to define R_w, the weekly rest period assigned to w. If R_w is a reduced weekly rest period, the variables C_i are used to define the compensation period C_w. Once these auxiliary variables are fixed, checking whether each weekly rest is of suitable length or has been compensated appropriately is a first-order property.

We conclude that the content of §8.6 admits a Σ^1_1 representation over the set of weekly models, as claimed.

8.3 Non-expressibility

We have seen that §8.9 and §8.6 are expressible in $\mathcal{L}^1_\forall$ and $\mathcal{L}^1_{\forall\forall}$, respectively. However, they are not expressible in $\mathcal{L}_{\bigcirc\Box}$, and §8.6 is not reasonably expressible in $\mathcal{L}_{\bigcirc\mathsf{U}}$. In order to see this, we use constructions similar to the examples given in Section 7.4.2. However, since these constructions are somewhat more elaborate, we fix some extra notation first.

Say that a model $\mathcal{M}$ is eventually resting if there is some m such that for all $n > m$ and all $a \in \mathbb{A}$, $n \in a^{\mathcal{M}}$ if and only if $a = \text{REST}$. The end of an eventually resting model $\mathcal{M}$ is the least such value of m that is also a multiple of 168 (i.e., $\mathcal{M}$ has a whole number of weeks). A week-long model is an eventually resting weekly model whose end is 168. We define the concatenation of two eventually resting models $\mathcal{A}$ and $\mathcal{B}$, denoted $\mathcal{A} \mid \mathcal{B}$, as follows. Let m be the end of $\mathcal{A}$. Then, for a predicate symbol P and $n \in \mathbb{N}$, we set:

$$n \in P^{\mathcal{A}|\mathcal{B}} := \begin{cases} n \in P^{\mathcal{A}} & \text{if } n \leq m \\ n - m \in P^{\mathcal{B}} & \text{if } n > m. \end{cases}$$

If k is a natural number then $\mathcal{A}^k$ denotes k concatenated copies of $\mathcal{A}$. If $n \in [24, 168)$, then n denotes a week with a rest period of n hours and no other rest periods of 12 hours or more; we assume that these weekly periods fall in the middle of each week without overlapping with other weeks, with the details being non-essential. However, we do assume that any two instances of the week represented by n are identical.

8.3.1 Assignments

We saw in Section 8.2.1 that the possibility of assigning weekly rest periods to a sequence of weeks is definable in $\mathcal{L}^1_\forall$ (and hence in $\mathcal{L}_{\bigcirc\mathsf{U}}$ via Kamp's theorem (Theorem 6.4.1)). One may then ask

[1]Note that in our reading of §8.7, we allow $J \cap C_{w'} \neq \emptyset$ for some week $w' \neq w$. This is because the regulation does not explicitly state that the nine-hour rest period attached to a compensation period cannot be used to compensate an additional week. However, a more strict reading with $J \cap C_{w'} = \emptyset$ for $w' \neq w$ can be formalized similarly.

whether $\mathcal{L}_{\bigcirc\Box}$ suffices to define it, and the answer is negative. In order to see this, we start by defining a number of models.

Definition 8.3.1 (E, I, L, EL, EIL)**.** Fix $d \in [24, 84]$. The week-long models E, I, and L are defined as follows:

- E is a model whose first $\lfloor d/2 \rfloor$ hours are REST;
- I is a model whose hours $[\lfloor d/2 \rfloor + 1, \lfloor d/2 \rfloor + d]$ are REST;
- L is a model whose last $\lceil d/2 \rceil$ hours are REST.

The remaining hours of each of the above models are not important for our purposes with the exception that they may not include rest periods with a length of 12 or more hours other than the ones explicitly mentioned above. Furthermore, all E weeks must be identical, and similarly for the others.

We use concatenations of letters to denote week-long models with more than one long rest period. Thus, EL denotes a week with the long rest periods of both E and L, while EIL denotes a week with all three long rest periods.

Definition 8.3.2 $(\mathcal{A}_n, \overline{\mathcal{A}}_n)$**.** Fix $d \in [24, 84]$. For each $n \in \mathbb{N}$, the eventually resting models $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ are defined as follows:

$$\mathcal{A}_n := (L \mid EL^n \mid EIL \mid EL^n \mid E)^{n+1}$$
$$\overline{\mathcal{A}}_n := L \mid EL^n \mid E \mid \mathcal{A}_n,$$

where EL^n denotes the EL week concatenated with itself n times; for example, $EL^3 = EL \mid EL \mid EL$.

Given $d \in [24, 84]$ and a model $\mathcal{H}$, we say that $\mathcal{H}$ admits a weekly rest assignment if it is possible to assign a weekly rest period of length at least d to every week in $\mathcal{H}$.

Lemma 8.3.3. *The model $\mathcal{A}_n$ admits a weekly rest assignment. The model $\overline{\mathcal{A}}_n$ does not.*

Proof. It is easy to see that $\mathcal{A}_n$ satisfies the formula φ_d of Theorem 8.2.2 and that $\overline{\mathcal{A}}_n$ does not. $\Box$

However, for any given $\mathcal{L}_{\bigcirc\Box}$ formula φ there is an n such that the models $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ are not distinguishable through φ, as the following lemma suggests.

Lemma 8.3.4. *There is a $168n$-$\Box$-bisimulation Z between $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ such that $0 \; Z_n \; 0$.*

Proof. First we provide some intuition. Consider the following representation of the models, where each number k in subscript marks the beginning of a week, such that $168k$ is the corresponding moment in the model:

$$\mathcal{A}_n = \; {}_0\, L \,|_1\, EL^n \,|_{n+1}\, EIL \,|_{n+2}\, EL^n \,|_{2n+2}\, E \,|_{2n+3} \; \cdots$$

$$\overline{\mathcal{A}}_n = \; {}_0\, L \,|_1\, EL^n \,|_{n+1}\, E \,|_{n+2}\, \mathcal{A}_n.$$

The models are very similar, except that $\overline{\mathcal{A}}_n$ has an extra initial $n + 2$ weeks. Thus, starting at world i in $\mathcal{A}_n$ and finding a bisimilar portion of $\overline{\mathcal{A}}_n$ is easy, just skip to world $168(n + 2) + i$ in $\overline{\mathcal{A}}_n$.

When starting from $\overline{\mathcal{A}}_n$, however, we need to be careful with the first $n+2$ weeks (after that it's a simple matter of going to the corresponding week in $\mathcal{A}_n$). If we are on the first week of $\overline{\mathcal{A}}_n$, we are free to go to the first week of $\mathcal{A}_n$; if we are between weeks 1 and $n+1$ of $\overline{\mathcal{A}}_n$, then we can go to the corresponding world between weeks $n+2$ and $2n+2$ of $\mathcal{A}_n$. In fact, we can even continue for $(n-1)(2n+3)$ more weeks, until we hit the end of $\mathcal{A}_n$ but not the end of $\overline{\mathcal{A}}_n$, at which point the models are different again.

We now make this argument in a more precise form.[2] Define $r := 2n+3$ and for $x = 168w + h$ and $y = 168v + \ell$ with $h, \ell < 168$, let $x \; Z_i \; y$ if and only if both $h = \ell$ and one of the following holds:

(A1) $w = v = 0$ and $i \leq n$;

(A2) $0 < v < n+2$ and $i < n$ and $w = v + n + 1$;

(A3) $v = w + n + 2$;

(A4) $n + 2 \leq v$ and $\max\{w, v - n - 2\} \leq (n - i)r$ and $v \equiv w + n + 2 \pmod{r}$.

We can easily check that $0 \; Z_n \; 0$. We now show that $\boldsymbol{Z}$ is a $168n$-$\Box$-bisimulation.

It is clear that $Z_{i+1} \subseteq Z_i$. Suppose that $x \; Z_0 \; y$ and write $x = 168w + h$ and $y = 168v + \ell$ with $h, \ell < 168$; note that by definition we must have $h = \ell$. Some inspection shows that x and y share the same formulas of the form $\bigcirc^j p$ with $j \leq 168n$, since the current and subsequent n weeks are of the same form. It is sufficient to check this for Z_0 because it contains Z_i for any i.

Now change variables so that $x \; Z_{i+1} \; y$; we check that the remaining clauses hold.

Forth $\Box$. Let $x' \geq x$ and write $x' = 168w' + h'$. We want to find $y' \geq y$ such that $x' \; Z_i \; y'$. By the definition of Z_i, we look for a v' such that $y' := 168v' + h'$ fulfills the desired conditions.

The easiest case is when $v' := w' + n + 2$ suffices, for we immediately obtain $x' \; Z_i \; y'$ via (A3). However, if $168v' + h' < y$ this doesn't suffice.

In that case, we have that either $w' + n + 2 = v$ and $h' < h$, or $w' + n + 2 < v$. It follows that $w + n + 2 < v$, through the following argument. In the first case, since $h' < h$, then $w < w'$; otherwise we would not have $x \leq x'$. Since $w < w'$, so does $w + n + 2 < w' + n + 2$. Then since we are assuming $w' + n + 2 = v$, it follows that $w + n + 2 < v$. In the second case, from $x \leq x'$ we deduce $w \leq w'$ and so $w + n + 2 \leq w' + n + 2$, whence $w + n + 2 < v$ by the assumption that $w' + n + 2 < v$.

From $w + n + 2 < v$ we conclude that (A4) must hold for $x \; Z_{i+1} \; y$, as none of the other options apply. Then $\max\{w, v - n - 2\} \leq (n - i - 1)r$.

Take $v' \in (v, v + r]$ such that $v' \equiv w' + n + 2 \pmod{r}$ and set $y' := 168v' + h'$. Clearly $y' \geq y$. Now, since $w' + n + 2 \leq v$ and $v - n - 2 \leq (n - i - 1)r$, it follows that $w' \leq (n - i - 1)r$ and consequently $w' \leq (n - i)r$. On the other hand, $v' \leq v + r$, so $v' - n - 2 \leq v + r - n - 2$. We already saw that $v - n - 2 \leq (n - i - 1)r$, so $v - n - 2 + r \leq (n - i)r$. From these two observations it follows that $v' - n - 2 \leq (n - i)r$. Then $x' \; Z_i \; y'$ via (A4).

Back $\Box$. Let $y' \geq y$ and write $y' = 168v' + h'$. We want to find w' such that $168w' + h' \geq 168w + h$ and $168w' + h' \; Z_i \; 168v' + h'$.

[2]This corrects the proof presented in [10]: there, the (A1) clause has a typo and the Back $\Box$ case is incomplete, so the bisimulation definition is slightly different here.

We distinguish the case where $n+2 \leq v'$ from the case where $v' < n+2$.

Suppose that $n+2 \leq v'$. As in the FORTH $\Box$ case, things are easy if $w' := v' - n - 2$ suffices, for then the result follows from (A3). If it doesn't suffice, then $168(v'-n-2)+h' < x$, so either $v'-n-2 = w$ and $h' < h$ or $v'-n-2 < w$, but in any case $v'-n-2 \leq w$. Furthermore, by reasoning similar to the one in the FORTH $\Box$ case, $v < w+n+2$.

Pick $w' \in (w, w+r]$ such that $w' \equiv v'-n-2 \pmod r$ and set $x' := 168w'+h'$. Clearly $x \leq x'$. We set out to show that $x' Z_i y'$ via (A4), of which the only non-trivial property is $\max\{w', v'-n-2\} \leq (n-i)r$ in this case. Since $v'-n-2 \leq w$ and $w < w'$, we only really need to check $w' \leq (n-i)r$, which follows from $w+r \leq (n-i)r$. We check this latter property in different ways depending on why $x \; Z_{i+1} \; y$.

Suppose that $x \; Z_{i+1} \; y$ via (A1). Then since $w = 0$, we just need to check $r \leq (n-i)r$, which follows from the (A1) assumption that $i+1 \leq n$.

Note that if $i+1 \leq n$ we can deduce $w+r \leq (n-i)r$ from $w \leq (n-i-1)r$. Thus, if $x \; Z_{i+1} \; y$ via (A2), we set out to prove $w \leq r$ and $r \leq (n-i-1)r$, which together give us the desired $w \leq (n-i-1)r$. The former is a consequence of $w = v+n+1$ and $v < n+2$, recalling the definition $r := 2n+3$. The latter is a straightforward consequence of the (A2) assumption $i+1 < n$.

Finally, if $x \; Z_{i+1} \; y$ via (A4) (the (A3) case being impossible due to $v < w+n+2$), then by assumption we have $w \leq (n-i-1)r$, so by the same reasoning as above we need only prove $i+1 \leq n$. Note that since (A4) holds, we have $n+2 \leq v$, so since we already observed $v < w+n+2$, it follows that $0 < w$. Then $0 < (n-i-1)r$, which implies $i+1 < n$ and consequently $i+1 \leq n$, as desired.

Assume now that $v' < n+2$. Then $x \; Z_{i+1} \; y$ via either (A1) or (A2), as the other two are impossible due to $y \leq y'$. If (A1) holds, then $w = v = 0$ and $i+1 \leq n$. If $v' = 0$, then $x' := h'$ is such that $x \leq x'$ and $x' Z_i y'$ via (A1). Otherwise, $0 < v'$ and we pick $w' := v'+n+1$ with $x' := 168w'+h'$, which is clearly such that both $x \leq x'$ and $x' \; Z_i \; y'$ via (A2). Lastly, if (A2) holds, then the same $x' := 168(v'+n+1)+h'$ works to obtain $x \leq x'$ and $x' \; Z_i \; y'$ via (A2). $\Box$

We are finally ready to prove the main result of this section, that §8.9 is not expressible in $\mathcal{L}_{\bigcirc\Box}$.

Theorem 8.3.5. *Given $d \in [24, 84]$, there is no formula $\varphi \in \mathcal{L}_{\bigcirc\Box}$ such that for every model J, we have $J, 0 \Vdash \varphi$ if and only if J admits a weekly rest assignment.*

Proof. Suppose that $\varphi \in \mathcal{L}_{\bigcirc\Box}$ is such a formula and let $d_{\bigcirc}$ and $d_{\Box}$ be its $\bigcirc$-depth and $\Box$-depth, respectively. Choose n such that $d_{\bigcirc} \leq 168n$ and $d_{\Box} \leq n$. Then by Lemmas 6.3.3 and 8.3.4, we have $\mathcal{A}_n, 0 \Vdash \varphi$ if and only if $\overline{\mathcal{A}}_n, 0 \Vdash \varphi$. However, according to Lemma 8.3.3, $\mathcal{A}_n$ admits a weekly rest assignment and $\overline{\mathcal{A}}_n$ does not. $\Box$

8.3.2 Compensations

Our goals now are to show that §8.6 is not expressible in $\mathcal{L}_{\bigcirc\Box}$ and that it is not expressible by a formula with a small U-depth in $\mathcal{L}_{\bigcirc\mathsf{U}}$. As before, we start by defining a model that complies with the regulation and one that doesn't, and then show that they are bisimilar up to a large enough bound.

Definition 8.3.6 ($\mathcal{B}_n, \overline{\mathcal{B}}_n$)**.** For each $n \in \mathbb{N}$, the eventually resting models $\mathcal{B}_n$ and $\overline{\mathcal{B}}_n$ are defined as follows:

$$\mathcal{B}_n := (44 \mid 45^n \mid 46 \mid 45^n)^n \mid 24 \mid 45 \mid 24$$
$$\overline{\mathcal{B}}_n := 44 \mid 45^n \mid \mathcal{B}_n,$$

where each number $i \in \{24, 44, 45, 46\}$ represents a week-long model with a Maximum Period of Consecutive Rest (MPCR) of length i and no other long periods of consecutive rest, as in Section 7.4.2.

Lemma 8.3.7. *Let $n \in \mathbb{N}$. Then $\mathcal{B}_n$ admits an assignment of compensations and $\overline{\mathcal{B}}_n$ does not. In other words, we have both $\mathcal{B}_n \vDash \psi_{\S 8.6}$ and $\overline{\mathcal{B}}_n \nvDash \psi_{\S 8.6}$.*

Proof. When $n = 0$ the result is straightforward. Assume then that $n > 0$.

In $\mathcal{B}_n$, the first week's missing hour can be compensated during the third week. This creates a chain reaction of compensations, as the third week then also needs to be compensated, since its MPCR is interpreted as a reduced weekly rest period of 44 hours together with a compensation of 1 hour. However, it is always possible to compensate either two weeks after, or on the week with a 46 hour MPCR, if it is close enough. It is thus never necessary to use up hours from the second block of n weeks with an MPCR of 45 hours, which all include regular weekly rest periods. This process repeats n times, until we reach the last three weeks of the model. Two of them need to be compensated, but it is possible to do so using the unlimited hours of rest available after the end.

Consider now $\overline{\mathcal{B}}_n$. The two weeks with an MPCR of 24 hours near the end of the model cannot be used to compensate previous weeks, since 24 is the minimum allowed length for a weekly rest period. The next-to-last week cannot be used to compensate previous weeks either, because then there would be more than one consecutive week with no regular rest period. Thus, we set the last three weeks aside.

There are $m := 2n^2 + 3n + 1$ weeks in the rest of the model, $2n^2 + n$ of which have MPCRs of 45 hours, $n + 1$ of which have MPCRs of 44 rest hours, and n of which have MPCRs of 46 hours, for a total of $45m - 1$ rest hours. Thus there are not enough rest hours to distribute among the period such that each week is assigned 45 hours of rest. $\square$

Lemma 8.3.8. *There is a $168n$-$\square$-bisimulation Z between $\mathcal{B}_n$ and $\overline{\mathcal{B}}_n$ such that $0 \; Z_n \; 0$.*

Proof. The bounded bisimulation and the proof are analogous to those of Lemma 8.3.4. $\square$

Theorem 8.3.9. *There is no $\mathcal{L}_{\bigcirc\square}$ formula equivalent to $\psi_{\S 8.6}$ over the class of eventually resting models.*

Proof. Suppose that $\psi \in \mathcal{L}_{\bigcirc\square}$ is a formula expressing §8.6 with $\bigcirc$-depth $d_{\bigcirc}$ and $\square$-depth $d_{\square}$. Choose n big enough to ensure that $d_{\bigcirc} \leq 168n$ and $d_{\square} \leq n$, and let Z be the bisimulation of Lemma 8.3.8. Then by Lemma 6.3.3, $\mathcal{B}_n, 0 \Vdash \psi$ if and only if $\overline{\mathcal{B}}_n, 0 \Vdash \psi$. This contradicts Lemma 8.3.7. $\square$

Now we show that any $\mathcal{L}_{\bigcirc\mathsf{U}}$ formula equivalent to $\psi_{\S 8.6}$ requires U-depth of at least 19, via the definition of two more eventually resting models.

Definition 8.3.10 ($\mathcal{C}_n, \overline{\mathcal{C}}_n$)**.** For each $n \in \mathbb{N}$, the eventually resting models $\mathcal{C}_n$ and $\overline{\mathcal{C}}_n$ are defined as follows:

$$\mathcal{C}_n := (44 \mid 45^{2n+1})^{21} \mid 66 \mid 24 \mid 45 \mid 24$$
$$\overline{\mathcal{C}}_n := 44 \mid 45^{2n+1} \mid \mathcal{C}_n.$$

Lemma 8.3.11. *Let $n \in \mathbb{N}$. Then $\mathcal{C}_n \models \psi_{\S 8.6}$ and $\overline{\mathcal{C}}_n \not\models \psi_{\S 8.6}$.*

Proof. First we see that $\mathcal{C}_n \models \psi_{\S 8.6}$. Intuitively, even weeks in need are compensated two weeks later, and the size of the compensation increases by one every $2n+2$ weeks. Thus for example one hour of week 0 is compensated by one hour of week 2, which is compensated by one hour of week 4, and so on until we reach week $2n+2$. Note however that this week only has an MPCR of 44 hours and has used one hour to compensate the previous week, so we need to compensate two hours of rest. This is compensated by two hours on week $2n+4$, and so on until we reach the third week with an MPCR of 44 hours. Since two hours of this rest are used to compensate a previous week, now three hours need to be compensated, and so on. On week $21(2n+2)$ we use 21 hours to compensate, which is the maximum allowed, given that each week requires at least a 24 hour rest period. Finally, the week with an MPCR of 66 hours breaks the pattern, allowing a full rest of 45 hours plus the required 21 hours of compensation. As in the case of $\mathcal{B}_n$, the last $24 \mid 45 \mid 24$ block can be compensated with the following unlimited rest.

More formally, every week w numbered $2k$ (including week zero) is reduced and compensated by week $2k+2$, up to and including week $21(2n+2)$. The amount of the compensation is the unique $i > 0$ such that $(i-1)(2n+2) \leq w < i(2n+2)$.

For the second claim, the $24 \mid 45 \mid 24$ block at the end of $\overline{\mathcal{C}}_n$ cannot be used to compensate previous weeks, similarly to what is discussed during the proof of Lemma 8.3.7. There are $m := 22(2n+2)+1$ remaining weeks in $\overline{\mathcal{C}}_n$, of which $22(2n+1)$ have an MPCR of 45 hours, 22 have an MPCR of 44 hours, and 1 has an MPCR of 66 hours, for a total of $45m - 1$ resting hours. Thus there are not enough resting hours to distribute among the weeks. $\square$

Lemma 8.3.12. *There is a $168n$-U-bisimulation $\boldsymbol{Z}$ between $\mathcal{C}_n$ and $\overline{\mathcal{C}}_n$ such that $0\ Z_{19}\ 0$.*[3]

Proof. Let $r := 2n+2$ and for $168w+h \in \mathcal{C}_n$ and $168v+\ell \in \overline{\mathcal{C}}_n$ with $h, \ell < 168$, set $168w+h\ Z_i\ 168v+\ell$ if and only if $h = \ell$ and one of the following properties holds:

(C1) $\max\{w+r, v\} < (21-i)r$ and $w \equiv v \pmod r$;

(C2) $v = w + r$.

Clearly, $0\ Z_{19}\ 0$. We now show that $\boldsymbol{Z}$ is a $168n$-U-bisimulation.

It is clear that $Z_{i+1} \subseteq Z_i$. Suppose that $x\ Z_0\ y$ and write $x = 168w + h$ and $y = 168v + \ell$ with $h, \ell < 168$; note that by definition we must have $h = \ell$. Some inspection shows that x and y share the same formulas of the form $\bigcirc^j p$ with $j \leq 168n$, since the current and subsequent n weeks are of the same form. It is sufficient to check this for Z_0 because it contains Z_i for any i.

[3] In [10] we claimed that $0\ Z_{20}\ 0$, but that is clearly not the case.

Now change variables so that $x\ Z_{i+1}\ y$; we check that the remaining clauses hold.

FORTH U. Let $x' \geq x$ and write $x' = 168w' + h'$ with $h' < 168$. Consider two cases. First assume that $w' \leq w + r$. Set $y' := y + (x' - x)$ and for $z \in [y, y']$ set $\xi(z) := x + (z - y)$. It is then not hard to check that if $x\ Z_{i+1}\ y$ by (C1) then $\xi(z)\ Z_i\ z$ by (C1), and similarly if $x\ Z_{i+1}\ y$ by (C2) then $\xi(z)\ Z_i\ z$ by (C2). The other requirements on ξ are easy to check.

Otherwise $w' > w + r$. We claim that there is v' such that $168v' + h' \geq 168v + h$ and $168w' + h'\ Z_i\ 168v' + h'$. If $168(w'+r)+h' \geq 168v+h$ we may take $v' = w'+r$. Otherwise, we have $v \geq w'+r > w+r$ so that x, y do not satisfy (C2) and thus $\max\{w+r, v\} \leq (21-i-1)r$. Take $v' \in (v, v+r]$ with $v' \equiv w' \pmod r$ and set $y' := 168v' + h'$; from $(21-i-1)r \geq v \geq w'+r$ and $v' \leq v+r \leq (21-i)r$ we obtain $x'\ Z_i\ y'$ by (C1).

We now construct the function $\xi\colon [y, y'] \to [x, x']$. First define $\xi(y') := x'$. For $z = 168u + t \in [y, y')$ with $t < 168$, we consider two cases. If $168(u-r)+t \in \big[x, x'\big)$ take $\xi(z) = 168(u-r)+t$, which in view of (C2) satisfies all desired properties. Otherwise, $168(u-r)+t \notin [x, x')$, and choose $d \in (0, r]$ such that $w+d \equiv u \pmod r$, then set $\xi(z) = 168(w+d)+t$. The assumption that $w' > w+r$ implies $\xi(z) \in [x, x']$. It remains to show that $\xi(z)\ Z_i\ z$, for which it suffices to check that $\max\{w+d+r, u\} < (21-i)r$.

If $168(u-r)+t < x$ then since $z \geq y$, either $u > v$ and hence $v < u \leq w+r$, or else $u = v$ and $t \geq h$, so that forcibly $u-r < w$ and thus $v < w+r$. But $v < w+r$ together with $168v+h'\ Z_{i+1}\ 168w+h$ means that (C1) holds so that $\max\{w+r, v\} < (21-i-1)r$. Thus we have $u-r \leq w < (21-i-1)r$ so that $u < (21-i)r$. Similarly $w+d \leq w+r < (21-i-1)r$ yields $w+d+r < (21-i)r$.

Otherwise $168(u-r)+t \geq x'$. But then since $z < y'$, either $u = v'$ and hence $t < h'$, so that $v'-r = u-r > w'$; or else $u < v'$ and $v'-r > u-r \geq w'$. Thus $w'+r \neq v'$, which together with $168w'+h'\ Z_i\ 168v'+h'$ yields $\max\{w'+r, v'\} < (21-i)r$. From $u \leq v' < (21-i)r$ and $w+d+r \leq (w+r)+r < w'+r < (21-i)r$ we obtain $\max\{w+d+r, u\} < (21-i)r$, as needed.

BACK U. This is essentially symmetric and we omit it. □

Theorem 8.3.13. *All $\mathcal{L}_{\mathsf{OU}}$ formulas equivalent to $\psi_{\S 8.6}$ have* U*-depth at least* 19.

Proof. Suppose that $\psi \in \mathcal{L}_{\mathsf{OU}}$ is a formula expressing §8.6 with O-depth d and U-depth less than 19. Let Z be the bisimulation of Lemma 8.3.12 with $n := d$. Then by Lemma 6.3.3, $\mathcal{C}_n, 0 \Vdash \psi$ if and only if $\overline{\mathcal{C}}_n, 0 \Vdash \psi$, which contradicts Lemma 8.3.11. □

One can ask how Theorem 8.3.13 would differ if we included "since" in the language. In this case, $\mathcal{C}_n, 0$ and $\overline{\mathcal{C}}_n, 0$ are only about 10-bisimilar. However, the nesting depth of 19 is determined only by the resolution of our models. If instead we used a minute-wise resolution (which, as we have mentioned, is the resolution required by Regulation (EC) 561/2006 [88] itself), we could stretch this to $19 \cdot 60$ by replacing the weeks with a MCRP of 44 hours with weeks with a MCRP of 44 hours and 59 minutes. Thus, any LTL definition of $\psi_{\S 8.6}$ would have to exploit the temporal resolution in an essential way, making it arguably unnatural.

We can consider a variant of the regulation where the requirements are: (i) in every two consecutive weeks, the driver must take two weekly rest periods, at least one of which is regular; and (ii) in every four consecutive weeks, the sum of the weekly rest periods must be of at least 180 hours. This formulation is definable by a reasonably-sized LTL formula and maintains the spirit of the original.

Part III

Formalization

9
Background

Contents

9.1 Introduction

In this chapter we discuss interactive theorem proving, both in the context of mathematics and in the context of software. We then describe Coq, our interactive theorem prover of choice, and touch on several strategies and tools relevant for formalization. This sets the stage for the descriptions of the two formalization projects described in Chapters 10 and 11.

9.2 Interactive theorem proving

Proofs as found in most publications, including this thesis, are informal arguments to be read by someone with a large amount of mathematical culture and knowledge. It is understood that these proofs can be formalized in systems such as classical or constructive set theory, or possibly in substantially weaker systems such as PA or EA, although often the specific system and steps toward formalization are left implicit. This is reasonable because people typically think at much higher levels than the axioms of any formalism, having internalized several consequences of the axiomatic systems they use long before producing any proofs worth publishing or reading any non-pedagogical publication. That most published proofs are still correct is a testament to this way of working.

Be that as it may, certain circumstances justify being more careful. Some proofs of hard problems are too complex or long to manually check. For example, the first proof of the four color theorem [19] used a computer to check hundreds of different cases. The community's uncertainty towards this proof was partly assuaged when it was fully formalized several years later [111].

A practical way of formalizing a non-trivial proof is to use a proof assistant, also known as an interactive theorem prover. This kind of software provides a scaffolding for the user to implement definitions, theorem statements, and proofs. Proofs are typically interactive: the user states the result to be proved (known as the goal) and incrementally describes steps towards advancing the proof, possibly introducing assumptions or generating more goals, until all goals are discharged. The user's inputs are internally translated to a proof in a formal system (typically some flavor of higher-order logic, possibly with dependent types). The proof assistant can offer a myriad of tricks to help the user (in particular, automatically solving some goals, see Section 9.6), but critically there should be a final proof object that can be checked correct via a small program (the proof assistant's kernel). This is known as the de Bruijn criterion [27].

When we rely on results verified by a proof assistant we are assuming that the kernel is correct and lacks any bugs. Small kernels can in principle be manually checked for correctness, although empirical tests are also an important part of why we trust them. Most software does include bugs, and critical problems are periodically found in the kernels of existing proof assistants [68]. The key difference with informal proofs is in the kinds of mistakes. A kernel bug that allows for a proof of a false result is very unlikely to exactly match with a reasoning mistake in an informal proof. Thus, even if ill-intentioned users who found a kernel bug could produce a wrong and machine-checked proof, users are unlikely to rely on kernel bugs without realizing it.

There are several well-known interactive theorem provers, such as Agda [2], Coq [66], HOL

Light [127], Isabelle [137], and Lean [151], among others. Several proof assistants are compared by Wiedijk [210] for the purposes of proving the irrationality of $\sqrt{2}$, while Ringer *et al.* [187] provide a more recent overview from the point of view of software verification and Geuvers [108] supplies a general history.

We used Coq for our work and focus on it during the rest of this chapter. Other interactive theorem provers could have been used to achieve similar results, but Coq provides many advantages: its underlying theory is strong enough to prove our results; there are several well-developed libraries for many useful data structures; and the community is large and active [8]. Furthermore, algorithms implemented in Coq can be extracted to other programming languages more suited for computation, such as OCaml. We further expand on Coq and its libraries in Section 9.3.

We now go into more detail on two main uses of interactive theorem provers: the formalization of mathematical theorems and the formalization of software.

9.2.1 Formalizing mathematics

When formalizing mathematics, an informal proof typically already exists and this proof merely needs to be translated into the language of the chosen proof assistant. This translation can be more or less straightforward depending on two factors: the previous existence of a library formalizing all the required ontologies, and the level of detail of the informal proof itself. Informal proofs typically assume a great deal of implicit knowledge that the proof assistant simply doesn't have. New Coq users may be surprised when goals as simple as $n + 0 = n$ are not automatically discharged. Even though there are many ways of solving such a goal, the fact remains that one of those ways must be used.

Furthermore, the language of a theorem prover is often not exactly the same as the one of everyday mathematics. Sets are ubiquitous in mathematics, but not primitive objects in Coq. Even though one can define the concept of set, many times they are not the most appropriate tool for a given job. For example, in informal mathematics $\mathbb{N}$ is the set of natural numbers and we write $n \in \mathbb{N}$ to say that n is a natural number. In Coq, `nat` is an inductive type and we write n : `nat` to say that `nat` is the type of n, which in practice is the same as saying n is a natural number.

This hints at the important art of code design. It is often the case that the precise definition used in an informal proof is not too important, because equivalent definitions can be used whenever relevant. However, the particular implementation of a given concept can fundamentally change the difficulty of a given formalized proof. Often, the definitions used with informal proofs are not suitable for formalization in a given system, and so alternative definitions (and consequently proofs) must be divined. In fact, this can lead to simpler proofs even on paper by translating the formalized proof back. We showcase two such examples in Sections 10.5.2 and 10.6.3.

Finally, formalizing an informal proof does unveil errors in the original, if they exist. Most of the time these are not serious errors: if found by a reader, they might be considered typos. Typical such errors are forgetting a (more or less obvious) assumption, inductions whose induction hypothesis must be strengthened, and other similar mistakes. Sometimes a proof just does not work as stated, but the result in itself continues being provable [115]. Sometimes genuine mistakes are found [52, 51].

Other than the already mentioned formalization of the four color theorem [111], there have been many formalizations of high-profile results, such as the Gödel-Rosser incompleteness theorem [179], the odd order theorem [112], the Kepler conjecture [123], and the Church-Rosser theorem [65], to name only a few.

We formalized most of the results of Chapter 4 and describe this formalization in Chapter 10, focusing on design decisions and remarks of general interest rather than on the exact implementation details or proofs.

9.2.2 Formalizing software

Formalizing software is different from formalizing mathematics in that typically there are no informal proofs to translate. Instead, there is some (formal or informal) specification for how existing software must behave under given conditions, and the formalizer's job is to prove that the existing implementation of that software satisfies the specification.[1] It is also possible to start only with a specification and write an implementation together with a proof that it meets the specification. Regardless of which process is chosen, some challenges must be overcome.

First, writing a good specification is a hard task in itself. Special care must be taken that each corner-case is considered and that the specification is truly what one means. This is unfortunate in that it tends to imply specifications are more complex than we would like them to be, which makes them harder to read and interpret, which in itself introduces room for mistakes. As an example, one can specify a sorting function as follows: "receive a list as input and output a sorted list", but this is of course silly, since the function that always outputs the empty list meets this specification: we forgot to require that the output be a permutation of the input.

Even assuming a perfect specification, there is one other problem. Interactive theorem provers are not known for their efficiency as compiled code, nor their interoperability with typical programming languages. For the purposes of fast code, it is better to use a language like C rather than Coq, so there is often a mismatch between the language we want to use for the implementations and the language where we can talk about correctness. This can be overcome in several ways, one of which is extraction. The implementation can be written in Coq and then automatically translated to OCaml (or other languages) via the Coq extraction plugin. This leads to a nice experience while formalizing, but the extraction process itself is not fully formalized and the code used in the real world is not the code about which there are proofs. Nevertheless, extraction is common in software verification and it is part of the process used for one of our formalizations. See Section 9.7 for details.

Another option is to do the opposite of extraction: start with an implementation already written in the desired language and (either automatically or manually) translate it into the Coq language. This suffers from the same problems as extraction and furthermore can lead to unidiomatic Coq code, about which it is hard to prove anything. The advantages are that the implementation is itself idiomatic of the language in which it is written, which is less than can be said of extracted Coq code

[1]One can have an informal proof of such a statement, and then the process is more similar to the one described in Section 9.2.1.

(but see Section 11.7) and that this can be easily applied to existing codebases. The `coq-of-ocaml` tool [61] automates the translation of OCaml to Coq and has been used to partially verify Tezos smart contracts [46].

Another worry in software formalization is the compilation of the final (possibly extracted) source code. We do not go into detail here, but mention only that the verified C compiler CompCert [154] is itself an important example of a verification project and it has been used in the context of formalizing other C code in Coq, leading to tools such as the Verified Software Toolchain [16] and Œuf [176].

Other verification projects of note are summarized by Appel [18], as well as different formalization strategies and tools. As final examples, we mention the verified implementation of SHA-256 [17], the verified FSCQ file system [59], and the `fiat-crypto` project [86], which verifies standard cryptographic functions and whose code is currently part of several well-known browsers.

We verified UTC calendars and time operations as part of the verified law project described in Part II, and describe the process in Chapter 11.

9.3 Coq and MathComp

Coq [66] is a general purpose interactive proof assistant. It provides a formal language expressive enough to write theorem statements and their proofs, as well as specifications of algorithms and their implementations. These proofs are verified by the Coq kernel, and are thus correct up to hypothetical (and unexpected) errors in the kernel itself.[2]

The core language is based on the Calculus of Inductive Constructions [71, 72], a constructive dependent type theory with inductive types, among other features. Even though the base theory is constructive, it is compatible with several common axioms, including the axiom of excluded middle.

The Mathematical Components libraries, also known as MathComp [166], are libraries of formalized mathematics originally developed for the formalization of the four color theorem [111]. They serve as an alternative to Coq's standard library and provide the theories of basic types such as natural numbers and lists (`mathcomp-ssreflect`), as well as finite sets of so-called choice types (`mathcomp-finmap` [63]), among many others we do not use here.

The proofs in MathComp use the SSReflect proof language [113] and particular definition and proof design principles further explained in the MathComp book [165]. For the projects described here, we used both MathComp and SSReflect, and tried to abide by the same design principles, one of which we detail below. Our naming convention is also inspired by MathComp.

In Coq, every term has a type, and every type is also a term (and thus has a (larger) type itself). We write $t : T$ to say that t has type T, or that T is inhabited by t.

Types of the form $T \to U$ (where T and U are types) represent functions with input of type T and output of type U. Functions with more than one input are typically represented via currying. For example, addition on the natural numbers is defined with type $\mathtt{nat} \to (\mathtt{nat} \to \mathtt{nat})$, i.e., it is a

[2]The MetaCoq project [196] provides a formalization of Coq in Coq (excepting some features such as the module system), which assures us that the implementation of the Coq kernel is both correct and complete with respect to its specification. Proving the soundness of the specification is a different matter, not possible in Coq due to Gödel's second incompleteness theorem.

function that receives a natural number as input and outputs a function from $\texttt{nat}$ to $\texttt{nat}$. Thus, we write $(\text{add } n)\ m$ instead of $\text{add}(n, m)$. The arrow $\rightarrow$ is taken to be right associative, so the parentheses in $\texttt{nat} \rightarrow (\texttt{nat} \rightarrow \texttt{nat})$ can be omitted. Similarly, function application is taken to be left associative, so the parentheses in $(\text{add } n)\ m$ can be omitted too.

It is relevant to distinguish between simple and dependent types. A function has a dependent type when the type of its output depends on the *value* of its input, and a simple type otherwise. For example, addition on the natural numbers has a simple type. To exemplify a dependent type, consider the type of finite ordinals as defined in MathComp. The notation is $\texttt{'I\_}n$ for the ordinal n, which is implemented as the type of natural numbers less than n. The function $\texttt{nat\_of\_ord}$ receives a natural number n and an element i of type $\texttt{'I\_}n$ and outputs the natural number representation of i. This function has the dependent type $\forall(n : \texttt{nat}), \texttt{'I\_}n \rightarrow \texttt{nat}$, where $\forall$ represents the product type or dependent function type. Note that in Coq the $\rightarrow$ binds stronger than the $\forall$ (this is hinted by the comma right after $\forall(n : \texttt{nat})$ in $\forall(n : \texttt{nat}), \texttt{'I\_}n \rightarrow \texttt{nat}$), which is not the typical convention in other contexts and in fact not the convention we use in the other parts of this thesis. We make sure to include the comma after the quantified variables to signal when the Coq convention is being used.

The dual of the dependent function type is the dependent pair type, sometimes represented by $\exists$. A dependent pair is an ordered pair where the type of the second element depends on the value of the first element. For more details on type theory refer to Nederpelt and Geuvers [177].

Coq has a special type, called $\texttt{Prop}$, which is meant to represent logical propositions. Thus, when $P : \texttt{Prop}$ we think of P as the statement of a lemma, and of inhabitants of P as proofs of P.

Terms can be defined directly or via tactics. For example, if $p : P$ and $p2q : P \rightarrow Q$ are known, then $(p2q\ p) : Q$ is a proof of Q, but such a proof could alternatively be obtained via the incantation $\texttt{apply: } p2q\texttt{; exact: } p\texttt{.}$ The tactic language enables a conversation between the user and the proof assistant and can be extended to solve complex goals, which we discuss in Section 9.6.

Most of the time, we don't care which particular proof of P was used to show that P was inhabited. In contrast, when defining a non-$\texttt{Prop}$ object, we often do care about which specific inhabitant was chosen. For example, the statement $0 : \texttt{nat}$ is much more informative than the statement "$\texttt{nat}$ is inhabited". We refer to inhabitants of $\texttt{Prop}$ as proofs or non-informative terms, and to other objects as informative terms.

The richness of Coq's dependent type theory allows one to define complex terms that are not very easy to work with. As an example, compare the standard vector library [53] with the MathComp tuple library [168], both of which define lists of fixed size. Standard library vectors are defined inductively: $\texttt{nil}$ is a vector of size 0, and if w is a vector of size n and x is an element of the vector's type then $\texttt{cons}\ x\ w$ is a vector of size $n + 1$. In contrast, MathComp tuples are inductively defined lists together with a proof that their size is n. These are isomorphic structures, but vectors require their own dedicated theory, while tuples require only a couple of specialized lemmas, inheriting the rest from lists.

More importantly, it is hard to work with vectors because any operation f on a vector must go beyond fulfilling the size invariant: it must do so provably in Coq at the time of f's definition. The proof must be interleaved with the definition of f because there is no way of talking about vectors without explicitly monitoring their size. In contrast, one can define a function g on lists regardless of its consequences of the list's size, prove facts about g's behavior on size separately, and use these proofs to obtain tuples out of g.

It has been our experience that a Coq development becomes much simpler when there is a clear separation between the informative, non-dependent part of a type and the dependent version with non-informative restrictions or assumptions.[3] Thus, we abide by the simple types convention, which states as follows.

Convention 9.3.1 (Simple types)**.** Use simply-typed terms as much as possible. If using dependently-typed terms, avoid interleaving the informative and non-informative parts.

This thesis tries to be accessible to someone who has never used Coq or even other interactive proof assistants. For this reason, we mostly highlight the interesting design decisions and difficulties that would plausibly arise in other formalization efforts and stick to standard mathematical notation. There are some exceptions when we delve into Coq-specific issues or decisions, and we also sometimes provide Coq code for the readers who are already familiar with the syntax and common MathComp terms. This code can safely be ignored in favor of the accompanying text description. Furthermore, when possible we mention the Coq name for each definition and theorem presented here. These names are hyperlinks to an online rendition of their source code.

9.4 Partial functions

A common issue in typed settings is that of defining partial functions. The domain in which a partial function is defined may be known, as is the case of division (undefined only when the divisor is zero) or unknown, as is the case of a function semi-deciding the halting problem. We ran into the former, simpler, problem several times in our development and employed different solutions depending on the circumstance. For a discussion of the latter problem see Bove *et al.* [54].

Consider, as an example of a partial function, Euclidean division on the natural numbers, `div`. Division is mathematically undefined when the divisor is zero. There are three main ways of implementing such partial functions in Coq, illustrated here with the type signature of division.

1. Output an error:

$$\mathtt{div}_{\mathsf{err}} : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat\_or\_error}$$

2. Forbid the input:

$$\mathtt{div}_{\mathsf{gt0}} : \mathtt{nat} \to \mathtt{nat\_greater\_than\_0} \to \mathtt{nat}$$

3. Output a default value:

$$\mathtt{div}_{\mathsf{dflt}} : \mathtt{nat} \to \mathtt{nat} \to \mathtt{nat}$$

[3]This boils down to style. There is a well-known Coq textbook [60] describing how to make heavy use of dependent types.

Outputting an error can be done with an option type. In that case, $\texttt{div}_{\text{err}}\ a\ b$ is `Some` n when b is not 0, and `None` otherwise. This strategy is then felt throughout the development, because every function or theorem mentioning division needs to expect and possibly output an option type as well. However, it is useful when preparing interfaces where we do want error handling.

Forbidding the input can be done by taking advantage of dependent types. In this case, the division function expects a proof that the divisor is greater than 0, and thus there is no need to provide any kind of output for that case. This strategy is mathematically clean, but it also creates ripples throughout the rest of the development. Every time $\texttt{div}_{\text{gt0}}$ is used, a proof that the divisor is greater than 0 must be provided. This goes against the first part of the simple types convention (Convention 9.3.1): "use simply-typed terms as much as possible." However, it can be useful when defining function specifications and stating theorems, and it is still possible to use dependent types while abiding with the second half of the convention ("avoid interleaving informative and non-informative parts").

Finally, there is the option of providing a default value for division by 0, such as 0 itself. This is how both Coq's standard library and MathComp implement Euclidean division. The advantage is that the type signature of $\texttt{div}_{\text{dflt}}$ is as simple as possible and can easily interact with every other function on the natural numbers. A smart choice of default value can even make true some theorems on division without requiring the extra assumption that the divisor be positive, but in general adding this assumption to theorems is not a problem. The disadvantage is that we end up formalizing slightly different structures from the ones used in typical informal proofs.

9.5 Refinements

The refinement technique refers to defining a list of programs $P_0, \ldots, P_{n+1}$ and showing for each $i < n$ that P_i relates to P_{i+1} in some specified way (for example, that they agree on a subset of inputs) under specific assumptions. In the end, P_{n+1} is a refinement of P_0 under the intersection of all assumptions. This is often easier than directly proving that P_0 and P_{n+1} have the desired relationship. This approach has been extensively developed both in specific proof assistant developments, such as Coq [82, 62, 81] and Isabelle [121], and in its purely theoretical aspects [24]. Some other software verification projects have used it too [56].

An example of this strategy is going from an inefficient but easy to understand program P_0 to an efficient but complicated program P_{n+1} that solves the same problem as P_0. This is how formal verification of software is sometimes done, P_0 being the specification and P_{n+1} the final implementation to be used in the real world. These two programs may use different data types and even ontologies, so it is useful to be able to change parameters one by one instead of all at the same time.

As an example, consider the natural numbers. Their default representation in the standard library and MathComp is the unary one: `0` is a `nat` and if n is a `nat` then `S` n is also a `nat`. This representation corresponds to the typical arithmetical one, and in particular the automatically generated structural induction principle corresponds to **IA** (Definition 2.2.2). It is a great type for proving things about natural numbers, and as such it is often used in Coq developments. However, it is a very inefficient representation in terms of space: explicitly representing unary numbers larger than 5000 in Coq makes the code almost unusable. There are ways around this, namely to encode the natural numbers in binary, which is the standard in computer science. The downside is that proofs using binary natural numbers become more complicated (the typical induction principle for the natural numbers is still provable for their binary representation, but it no longer coincides with structural induction).

The usual solution for this kind of problem is to use type refinements. There is an intermediate step where algorithms are defined on top of non-efficient data types and then redefined on top of efficient ones. A proof can then be provided to ensure that the refinement kept the relevant properties of the algorithm.

Some types are isomorphic: for example, every provable result about natural numbers that does not look into their implementation can be proved regardless of whether the numbers are represented in unary or binary form. In contrast, some types are not isomorphic but behave in the same way under certain conditions: for example, modular arithmetic behaves like regular arithmetic as long as no operation goes beyond the module (in other words, as long as there is no overflow), and the non-negative integers behave like the natural numbers so long as we do not subtract larger quantities from smaller ones.

Translations between isomorphic (or merely similar) data types are valuable for type refinements because they are project-independent. The same translation library or tool between unary and binary natural numbers can be used in the type refinement step of any project using natural numbers. The Coq standard library already provides some such translations, for example between the unary and binary versions of the natural numbers, between the binary versions of natural numbers and integers, and between integers and primitive integers. In Section 11.6 we mention a similar translation between MathComp unary natural numbers and unsigned primitive integers, as well as between MathComp unary integers and signed primitive integers.

9.6 Automation

The most natural process for writing proofs in Coq is to use tactics. Each tactic manipulates the proof assumptions or the goal in some way, many of which straightforward. For example, there is a tactic to add a hypothesis to be proved later, and another to observe that the goal coincides with one of the available hypotheses. Other tactics perform more complex manipulations of the goal, such as outright solving problems in Presburger Arithmetic [47] or identifying intuitionistic propositional tautologies [70]. When we mention automation in Coq we are thinking of the latter kind of tactics, although the former are also technically automating part of the proof. See Ringer *et al.* [187] for a history and more thorough overview.

There are many different tactic languages in Coq, which permit the creation of new tactics: the standard one is Ltac [80], but it is slowly being superseded by Ltac2 [181]. Other meta-languages include Mtac [213], Template-Coq [15], and Coq-Elpi [201]. It is also possible to directly write Coq tactics in OCaml.

We barely used any automation in the formalization described in Chapter 10. However, during the project described in Chapter 11 there were many tedious arithmetical goals to solve, and as such we tried a number of automation tools and eventually built our own, which is described in Section 11.9. Below is a brief overview of the state of the art.

Linear integer arithmetic is decidable and the `lia` tactic solves any goal in that fragment. It is part of the micromega [47] set of tactics, which go beyond this small fragment of arithmetic and provide partial solvers for non-linear problems as well as for several Coq and MathComp representations of different kinds of numbers (naturals, integers, and rationals). These different types are handled by the extendable translator tactic `zify` and its MathComp version `mczify` [189].

Also in the realm of arithmetic, `interval` [171] can be used to solve interval arithmetic goals over floating-point numbers, and also combined with external automation such as `gappa` [48].

The `ring` and `field` [118] tactics simplify (semi-)ring and field equations down to a canonical polynomial form, which avoids manual associativity and commutativity rewriting.

Other tactics are more generic, such as `auto` and `eauto` [69], which are lemma aggregators, meaning that they produce proofs by choosing from a set of existing lemmas configurable by the user. It is also possible to connect to external automated theorem provers via Coq Hammer [77], Gappa [48], or others.

9.7 Extraction

Coq provides extraction to OCaml, Haskell, and Scheme [155, 156, 157]. This consists of an automatic translation from the language of Coq to the chosen target language, with two main parts: erasure of non-informative types and translation of dependent types to the simpler types available in the target languages. Thus, lemmas are extracted to empty objects, while types that mix informative and non-informative parts lose the non-informative ones.

Extraction is one of the features that make Coq a prime choice for software verification: software can be written in Coq first, where it is easy to state and prove facts about it, and then be extracted into a more efficient and mainstream language, where it can be combined with non-formalized code and used to solve practical problems [45, 76].

The extraction algorithms are not fully formalized yet (although erasure to an intermediate language is [196], and there are plans for extending it in the near future), and so it is possible that errors in the extraction process lead to unexpected discrepancies between the original and the extracted code. Even if no errors occur, extracted code is typically non-idiomatic and hard to read. Worse, since some Coq types are not expressible in, say, OCaml, extraction often introduces type-casts via `Obj.magic`, a low-level OCaml function that allows casting any type to any other type. These casts are permissible because the functions type-checked in Coq's type theory. However, interoperability with non-formalized code must be done with care.

Extracting dependent types can lead to other problems, even if `Obj.magic` is not needed. Consider, for example, the finite ordinals as defined in MathComp. The notation is `'I_`n for the type of natural numbers smaller than n. Naturally, `'I_`0 is the empty type. We can define a function that, given a proof of $0 < n$, produces an inhabitant of `'I_`n (in this case `ord0`, the first ordinal):

```
From mathcomp Require Import all_ssreflect.

Definition nonempty_ordinal (n : nat) : 0 < n → 'I_n.
  by case: n ⇒ [//|n _]; exact: ord0.
Defined.
```

This function can be extracted,[4] and we see that the case where the input is 0 is translated into `assert false`:

```
(** val nonempty_ordinal : nat → ordinal **)

let nonempty_ordinal = function
| O → assert false (* absurd case *)
| S _ → O
```

However, there is a possible trap:

```
Definition trap := nonempty_ordinal 0.
```

This is a perfectly reasonable Coq term, and so can be extracted, leading to:

```
(** val trap : __ → ordinal **)

let trap _ =
  nonempty_ordinal O
```

Note that if `trap` is called from OCaml it will lead to an error via `assert false`.

Despite its flaws, extraction is an extremely powerful and useful tool, which we rely on to obtain practical formalized software. We describe our particular process in Section 11.7, which led to clean, reasonably short, and readable extracted code.

[4]The code shown here is not literally the result of extracting `nonempty_ordinal`, but the result of extracting a `cbv` simplification of it, which leads to equivalent and much easier to read OCaml code.

10

Formalized QRC_1

Contents

10.1 Introduction

This chapter presents a Coq formalization [5] of the main results of Chapter 4, namely the soundness and completeness of QRC_1 with respect to relational semantics, as well as its decidability. Points where we diverged from the definitions or proofs described in that chapter are particularly highlighted. The first part of this chapter (up to Section 10.5) was published in [4], while the remaining sections are new. There is also a summary of the formalization available online [6], serving as a kind of documentation.

Quantified modal logic has been extensively studied [110], and even formalized. For example, Basin *et al.* [28] describe a modular Isabelle formalization of several quantified modal logics, including Kripke soundness and completeness theorems for them. On the other hand, Benzmüller and Paleo [43] describe a set of Coq tactics to facilitate showing that a user-defined and possibly quantified modal logic proves a given statement. We have not made use of this library as our main goal was to prove meta-theorems of QRC_1. There has also been work on a custom proof assistant for quantified modal logic [158], as well as an automated theorem prover for normal quantified modal logics [109]. Furthermore, there are several implementations of propositional modal logics, both in Coq [84, 83, 3] and in other proof assistants [163], as well as presentations of first-order logics [179, 102].

We briefly present some figures comparing the formalization presented here with mathematical text describing the same definitions, theorems, and proofs. With this information, we calculate this project's de Bruijn factor [209], which is the quotient between the compressed size of the formalization and the compressed size of natural language text describing the same results. The formalization described in this document takes up about 47K of memory when compressed (corresponding to 4862 lines of code excluding comments and empty lines). That is roughly 2.8 times as much as the compressed size of the LaTeX source for the relevant parts of Chapter 4 (corresponding to about 15 typeset pages). This is a smaller factor than many of the ones described by Wiedijk [211], which indicates that the text we chose to formalize was reasonably amenable.

10.2 Language

The language of QRC_1 is that of quantified and strictly positive formulas based on a signature with both constants and predicate symbols. We define the names of variables, `VarName`, as simply the natural numbers, ensuring there is always a fresh variable for any given finite context. We then define the concept of (finite) `signature` as including a finite type of constant names `ConstName`, a finite type of predicate names `PredName`, and a function `arity` from the relation symbols to the natural numbers assigning an arity to each one. The finiteness is not much of a restriction, as signatures can be expanded or restricted without impact on the QRC_1 theorems in the appropriate language (see Lemma 4.2.6 and Corollary 4.2.9).

A `term` is either a variable or a constant. We define the appropriate canonical instances for `eqType` (equality on terms is decidable), `countType` (there is a countable amount of terms), and `choiceType` (there is a choice function for terms). This makes it possible to talk about finite sets of terms using the machinery of the Finite Maps library [63] later on.

A `formula` in a fixed signature is either $\top$, a predicate name together with a tuple of terms of the arity given by the signature, a conjunction of two other formulas, a diamond of one other formula, or a universal quantifier of a variable and another formula. We use standard mathematical notation in this text and reasonable approximations for this notation in the Coq development.

As with terms, we can show that equality on formulas is decidable and that there is a countable number of formulas by reduction to a structure that has previously been shown to be countable, trees [167] in this case.

The `Language.v` file then goes on to define several standard notions and facts, such as free variables (fv(φ) or `fv`), substitution ($\varphi[t_1{\leftarrow}t_2]$ or `sub`), simultaneous substitution ($\varphi[\boldsymbol{t}{\leftarrow}\boldsymbol{u}]$ or `simsub`), and being free for a variable in a formula (no occurrence of a free variable becomes bound after the substitution, or `freefor`).

Our formulas live in a quantified language, and as such there is a distinction between free and bound variables. This distinction is important when dealing with substitution, since it should not impact bound variables. Thus, $(\forall\, x\, \varphi)[x{\leftarrow}y]$ should be exactly $\forall\, x\, \varphi$ because x is not a free variable of $\forall\, x\, \varphi$. There is one tricky issue, though: cases where replacing a free variable by a term lead to a previously free occurrence becoming bound, such as in $(\forall\, y\, S(x, y))[x{\leftarrow}y]$. Here a naive substitution would lead to $\forall\, y\, S(y, y)$, which clearly does not preserve logical strength. In informal mathematics it is common to ignore this issue by observing that the names of the bound variables are ultimately irrelevant: if we wish to replace x by y in $\forall\, y\, S(x, y)$, then this can be achieved by first renaming the bound variable to some fresh name such as z, and then doing the substitution. The final formula would then be $\forall\, z\, S(y, z)$. This is the approach taken by O'Connor [179] in his formalization of the Gödel-Rosser incompleteness theorem. However, the paper cites this decision as having led to many issues in the formalization; although it ultimately works, we did not wish to use the same strategy.

Another common solution for this problem is to use de Bruijn indices [55]. This avoids naming bound variables altogether, so this concern does not appear. However, this approach is complex in its own right and would make the formalization considerably different from Chapter 4 in the details, if not the overall arc. There is a tool named Autosubst [198] that internally uses de Bruijn indices but generates the boilerplate code by itself and thus cuts back on the complexity and size of the developments. We have not made use of Autosubst, although it would be interesting to see how many lines of manually-written code and complications it would save.

The approach we settled on was inspired by [190] and the will to avoid mixing non-informative and informative objects, following the simple types convention (Convention 9.3.1). We define unguarded substitution, `sub`, and add an extra assumption, `freefor`, as needed. This assumption assures us that the replacing term will not be captured under any binders after substitution. We already spoke of terms being free for variables in formulas in Chapter 4, so the formalization is very similar to its informal counterpart. Furthermore, there were no significant complications in using this approach in the formalization of relational soundness. This was no longer the case for the formalization of relational completeness, mostly due to the formalization of Lemma 4.2.6, where we show that new constants can be conservatively added to the signature (see Section 10.6.1).

One downside of our strategy is that variable names must be picked with some foresight. Going back to our example from above, if $(\forall y\, S(x,y))[x{\leftarrow}y]$ ever appears in our development then we can perform the substitution, but won't be able to use any of the results about it because here y is not free for x in $\forall y\, S(x,y)$. Thus it is assumed that in practice the names for the bound variables do not clash with the names for the free variables, or that bound variables are renamed as needed.

Finally, we declare a sufficient condition for a variable to be fresh with respect to a formula (`fresh`), namely being larger than all the variables appearing in the formula. This depends on our implementation of variables as natural numbers, since we use the order relation of `nat`, but it is without loss of generality since any infinite and countable set of variables would be isomorphic to the natural numbers and hence have an associated order relation. This implementation has the following useful property: if x is `fresh` for φ and y is `fresh` for ψ, then $\max\{x, y\}$ is `fresh` for $\varphi \wedge \psi$. The typical notion of freshness ("x is fresh for φ if x does not appear in φ") does not have a similar property and we do not implement it at all.

10.3 Axiomatization

The axioms and rules of QRC_1 are represented as the `QRC1Proof` inductive type. We use a slightly different notation in the Coq implementation, with Latin upper-case letters instead of Greek lower-case letters for formulas and `|- A ~> B` instead of $A \vdash B$ (or $\varphi \vdash \psi$). Each of the axioms and rules described in Definition 4.2.1 has a Coq counterpart. For example, the quantifier introduction on the right rule has the following type:

```
AllIr : ∀ (sig : signature) (x : VarName) (A B : formula sig),
          x \notin fv A → |- A ~> B → |- A ~> All x B.
```

This representation facilitates the proofs of meta-theorems such as relational soundness and completeness, which is our ultimate goal. However, it is still quite easy to prove theorems within QRC_1 itself, as the simple formalization of Lemma 4.2.2 illustrates. Each of this lemma's items can be found in `QRC1.v`, starting with `AllC` and ending with `Diam_All`.

Like other provability logics, QRC_1 is irreflexive, i.e., $\varphi \vdash \Diamond\varphi$ is not provable (Corollary 4.2.5). However, unlike other provability logics, this fact can be shown without semantics. Its formalization is called `Diam_irreflexive`.

10.4 Relational semantics

Recall from Section 4.3 that the relational semantics for QRC_1 is based on Kripke sheaves, which are a generalization of the relational semantics for propositional modal logics where each world is a first-order model and for each two worlds w, u such that wRu there is a compatibility function $\eta_{w,u}$ from the domain of w to the domain of u.

When implementing Kripke models for QRC_1 in Coq, we made small changes to the definitions, which we comment on below. We furthermore used the keyword "frame" instead of "sheaf," which we use in this chapter to match the source code.[1]

The definitions of frame and model are implemented in two steps: first we define simple types with a `raw` prefix (`rawFrame`, `rawModel`) and then we obtain the final types (`frame`, `model`) by including a non-informative restriction that they must be adequate. This strategy is inspired by MathComp, follows the simple types convention (Convention 9.3.1), and is repeated in Section 11.3 in a different context.

Definition 10.4.1 (`rawFrame`, `rawModel`)**.** A `rawModel` $\mathcal{M}$ in a signature Σ is a tuple $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{w,u\in W}, \{I_w\}_{w\in W}, \{J_w\}_{w\in W}\rangle$ where:

- W is a finite set (the set of `world`s, where individual worlds are referred to as w, u, v, etc);
- R is a binary relation on W (the accessibility relation `R`);
- M_w is a finite set for each $w \in W$ (the `domain` of the world w, whose elements are referred to as d, d_0, d_1, etc);
- $\eta_{w,u}$ is a function from M_w to M_u for each $w, u \in W$ (the compatibility function `eta` between w and u);
- for each $w \in W$, the interpretation I_w assigns an element of the domain M_w to each constant $c \in \textsc{Const}_\Sigma$, written c^{I_w} or `I w c`; and
- for each $w \in W$, the interpretation J_w (or `J w`) assigns a set of n-tuples $S^{J_w} \subseteq \wp((M_w)^n)$ to each n-ary predicate symbol $S \in \textsc{Rel}_\Sigma$.

The $\langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{w,u\in W}\rangle$ part of a `rawModel` is called its `rawFrame`.

The main difference between the above definition of `rawFrame` and the definition of sheaf presented in Section 2.5.1 is that above we postulate compatibility functions for every pair of worlds, while in typical sheaves $\eta_{w,u}$ is only defined when wRu. The restriction makes sense because the notion of satisfiability only uses the compatibility functions in those cases. However, including such a restriction in a Coq definition, although possible, leads to noticeable inconveniences and goes against the simple types convention (Convention 9.3.1). Our work-around is to change the notion of frame so that $\eta_{w,u}$ must exist for every pair of worlds w and u. This decision was crucial in the formalization of the soundness of the constant elimination rule (Rule 4.2.1.(x) or `ConstE`), which was the trickiest one (see Section 10.5.2).

Another small difference is that we do not explicitly require non-empty domains. None of the formalized results depend on the domains being non-empty; at worst they become trivial. There is no particular reason to avoid adding this restriction other than the code becoming more complex for no clear benefit.

The final difference is that we only implement finite models, in the sense that both the set of worlds and each domain are finite. This does not impact the completeness proof, since QRC_1 has the finite

[1] When we discovered that our adapted notion of Kripke frame was known in the literature as Kripke sheaf, most of the Coq code had already been released.

model property (Theorem 4.4.13), but it does mean that the formalized soundness proof is slightly weaker than the more general one presented in Section 4.3.

Note how we do not lose generality with the alternative definition. The extra $\eta_{w,u}$ functions can be dropped to obtain the original definition; on the other hand, when the domain M_u is non-empty we can define a function $\eta_{w,u}$ from M_w to M_u. Since this function is not used, it doesn't matter which one we pick. Thus, the original soundness theorem for models with fewer compatibility functions and non-empty models (Theorem 4.3.10) can easily be recovered from the formalized one, and similarly for completeness (Theorem 4.4.13).

Above, there were no restrictions on the validity of the frames and models, and in fact their Coq keywords are `rawFrame` and `rawModel` precisely to highlight that they might not fulfill all the required restrictions. We implement those restrictions as follows.

Definition 10.4.2 (`adequateF`, `adequateM`)**.** A `rawFrame` $\mathcal{F}$ is adequate (`adequateF`, to distinguish it from the similar notion for models) if all of the following hold:

- R is transitive: if wRu and uRv, then wRv (`transitiveF`);
- the compatibility functions respect transitivity: if wRu and uRv, then $\eta_{w,v}(d) = \eta_{u,v}(\eta_{w,u}(d))$ for every d in the domain of w (`transetaF`);
- for every world w, the compatibility function $\eta_{w,w}$ is the identity (`idetaF`).

A `rawModel` is adequate (`adequateM`) if it is based on an adequate frame and it is:

- concordant: if wRu, then $c^{I_u} = \eta_{w,u}(c^{I_w})$ for every constant c (`concordantM`).

Note that in an adequate and rooted model the interpretation of the constants is fully determined by their interpretation at the root.

The notion of `frame` is defined by pairing a `rawFrame` with a proof that it is adequate, and similarly for `model`. When defining specific frames or models or notions that depend on frames or models such as satisfiability, we use the `raw` versions. On the other hand, when stating facts about frames or models we use the adequate versions, if necessary. We make use of implicit coercions (see the next paragraph) in order to smoothly refer to operations that expect a `rawFrame` in a theorem statement about a `frame`, and similarly for models.

A coercion is a function f that is automatically used by Coq when an otherwise ill-typed statement would be well-typed in the presence of f. For example, we declare a coercion from `rawFrame` to the set of worlds that lets us write statements such as $\forall(F : \texttt{rawFrame})(w : F), \ldots$ that closely resemble the common shorthand of stating that a world is part of a frame instead of part of the set of worlds of the frame. In this case Coq automatically infers the implicit coercion `world` necessary to make the statement type-check. Explicitly, it would be $\forall(F : \texttt{rawFrame})(w : \texttt{world}\ F), \ldots$

We use a small number of coercions in our development, the most important of which are represented in Figure 10.1. These coercions serve as a translation between a type and a super-type (in the sense that the former is a sub-type of the latter). We have a very small type hierarchy. Formalizations of, say, mathematical algebra or large libraries such as MathComp include rich hierarchies [188], and there are existing tools to implement and maintain such large hierarchies such as Hierarchy Builder [64].

$$\begin{tikzcd}
& \texttt{model} \arrow[dl, "\texttt{frame\_of\_model}"'] \arrow[dr, "\texttt{rawModel\_of\_model}"] & \\
\texttt{frame} \arrow[dr, "\texttt{rawFrame\_of\_frame}"'] & & \texttt{rawModel} \arrow[dl, "\texttt{rawFrame\_of\_rawModel}"] \\
& \texttt{rawFrame} &
\end{tikzcd}$$

Figure 10.1: A representation of the four datatypes defined to represent frames and models and the coercions between them. Each arrow from `X` to `Y` represents the coercion `Y_of_X`.

Still, even with a small hierarchy we do run into some issues. For example, consider the following unification problem:

$$\texttt{rawFrame\_of\_frame}\ ?F = \texttt{rawFrame\_of\_rawModel}\ (\texttt{rawModel\_of\_model}\ M) \tag{10.1}$$

In words, given a `model` M, a `frame` $?F$ must be found such that its `rawFrame` corresponds to the `rawFrame` of the model. The diamond represented in Figure 10.1 trivially commutes, so we define the canonical coercion `frame_of_model` as the path to solve (10.1) with $?F := \texttt{frame\_of\_model}\ M$.

We use assignments to define truth at a world in a first-order model. Fixing a world w, a w-`assignment` g is a function assigning a member of the domain M_w to each variable in the language.

Two w-assignments g and h are Γ-alternative, written $g \sim_\Gamma h$ (`Xaltern` g h Γ in Coq), if they coincide on all variables other than the ones in Γ. We write $g \sim_x h$ instead of $g \sim_{\{x\}} h$. We define the extension to terms g^+ of a w-assignment g as $g^+(x) := g(x)$ for any variable x and $g^+(c) := c^{I_w}$ for any constant c (named `assignment_of_term` in Coq).

We now define satisfiability at a world.

Definition 10.4.3 (`sat`)**.** Let $\mathcal{M} = \langle W, R, \{M_w\}_{w\in W}, \{\eta_{w,u}\}_{w,u\in W}, \{I_w\}_{w\in W}, \{J_w\}_{w\in W}\rangle$ be a model in some signature Σ, and let $w \in W$ be a world, g be a w-assignment, S be an n-ary predicate symbol of Σ, and φ, ψ be formulas in the language of Σ.

We define $\mathcal{M}, w \Vdash^g \varphi$ (φ is true at w under g) recursively on φ as follows:

- $\mathcal{M}, w \Vdash^g \top$;
- $\mathcal{M}, w \Vdash^g S(t_0, \ldots, t_{n-1})$ iff $\langle g^+(t_0), \ldots, g^+(t_{n-1})\rangle \in S^{J_w}$;
- $\mathcal{M}, w \Vdash^g \varphi \wedge \psi$ iff both $\mathcal{M}, w \Vdash^g \varphi$ and $\mathcal{M}, w \Vdash^g \psi$;
- $\mathcal{M}, w \Vdash^g \Diamond\varphi$ iff there is a $u \in W$ such that wRu and $\mathcal{M}, u \Vdash^{\eta_{w,u}\circ g} \varphi$;
- $\mathcal{M}, w \Vdash^g \forall x\, \varphi$ iff for all w-assignments h such that $h \sim_x g$, we have $\mathcal{M}, w \Vdash^h \varphi$.

Note how we haven't required that $\mathcal{M}$ be adequate in the definition of satisfiability, as it is not needed. We will of course assume the models are adequate when proving facts about them. Note also that the expression $\mathcal{M}, w \Vdash^g \varphi$ is only defined when g is a w-assignment.

The notion of satisfiability is decidable on finite models, due to there being only finitely-many worlds to check in the diamond case and finitely-many domain elements to check in the quantifier case. Since this fact is useful for proving the decidability of QRC_1 (see Section 10.7.2), we also define a boolean version of satisfiability $\Vdash_{\mathtt{b}}$ (`satb`). It is defined very similarly to `sat`, the main difference being the quantifier case, which is as follows:

- $\mathcal{M}, w \Vdash^g_{\mathtt{b}} \forall x\, \varphi$ iff for all $d \in M_w$, we have $\mathcal{M}, w \Vdash^{g[x\leftarrow d]}_{\mathtt{b}} \varphi$,

where $g[x\leftarrow d]$ is the w-assignment that outputs d on input x and $g(y)$ on all other inputs y. This formulation is easier to implement because M_w is already proved to be finite, unlike the set of w-assignments that are x-alternative to g. These two versions of satisfiability are equivalent (`satP`).

10.5 Relational soundness

The relational soundness of QRC_1 (Theorem 4.3.10, `soundness`) is proved by induction on the proof of $\varphi \vdash \psi$.

Theorem 4.3.10 (Relational soundness). *If $\varphi \vdash \psi$, then for any adequate model $\mathcal{M}$, for any world $w \in W$, and for any w-assignment g:*

$$\mathcal{M}, w \Vdash^g \varphi \implies \mathcal{M}, w \Vdash^g \psi.$$

Most of its Coq formalization is straightforward, with the exception of two lemmas. The first is Lemma 4.3.3, stating that satisfiability of a formula φ under an assignment g does not depend on the values $g(x)$ for variables x not free in φ. That this proof poses a problem is perhaps surprising, since it is such an intuitive lemma. We discuss this in Section 10.5.1.

The second is Lemma 4.3.9, stating that the constant elimination rule is sound. This is the most complicated part of the overall soundness proof and it is not surprising that its formalization is non-trivial. We discuss it in Section 10.5.2, where we present a slightly different proof from the one described in Section 4.3.

10.5.1 Finite sets

In our formalization, we restrict ourselves to finite sets. This allows us to make use of the nice Finite Maps library for choice types of MathComp [63] instead of having to prove many basic facts from scratch. However, Lemma 4.3.3 (reproduced below) made us momentarily reconsider this decision.

Lemma 4.3.3 (Absent free variables). *Let $\mathcal{M}$ be an adequate model, w be a world, g, h be Γ-alternative w-assignments, and φ be a formula with no free variables in Γ. Then:*

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}, w \Vdash^h \varphi.$$

This lemma feels intuitive and in fact its proof was omitted in [13] and [14]. However, it is not as straightforward as it looks. A simple induction is underpowered to solve it; one must do induction building with the assumption that g and h are (Vars $\setminus$ fv(φ))-alternative instead. Since Vars $\setminus$ fv(φ) is not a finite set, it cannot be represented by the machinery of the Finite Maps library. In order to get around this, we define the notion of Γ-equivalent assignments.

Definition 10.5.1 (`Xeq`)**.** Two w-assignments g and h are said to be Γ-equivalent if they agree on every variable in Γ.

Clearly, g and h are (Vars $\setminus$ fv(φ))-alternative if and only if they are fv(φ)-equivalent. Then it is easy to prove a version of the absent free variables lemma using the notion of Γ-equivalency, stated as follows.

Lemma 10.5.2 (`sat_Xeqfv`)**.** *Let $\mathcal{M}$ be an adequate model, w be a world, φ be a formula, and g, h be* fv(φ)*-equivalent w-assignments. Then:*

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}, w \Vdash^h \varphi.$$

Then Lemma 4.3.3 (`sat_Xalternfv`) is an easy corollary, and we successfully avoid working with non-finite sets.

10.5.2 Constant elimination rule

Recall the constant elimination rule (Rule 4.2.1.(x), `ConstE`):

if $\varphi[x{\leftarrow}c] \vdash \psi[x{\leftarrow}c]$, then $\varphi \vdash \psi$
(c not in φ nor ψ)

The argument for its soundness goes as follows. Suppose that $\varphi[x{\leftarrow}c] \vdash \psi[x{\leftarrow}c]$ is true and that $\mathcal{M}, w \Vdash^g \varphi$ for some adequate model $\mathcal{M}$, world w, and w-assignment g. We wish to show that $\mathcal{M}, w \Vdash^g \psi$. We build a new model $\mathcal{M}[w, c{\leftarrow}g(x)]$ that is identical to $\mathcal{M}$ except it interprets c as $g(x)$ in w, in hopes that $\mathcal{M}[w, c{\leftarrow}g(x)]$ satisfies $\chi[x{\leftarrow}c]$ at w if and only if $\mathcal{M}$ satisfies χ at w, for any formula χ where c does not appear. We can then deduce that $\mathcal{M}[w, c{\leftarrow}g(x)], w \Vdash \varphi[x{\leftarrow}c]$ from our assumption that $\mathcal{M}, w \Vdash^g \varphi$, and, since $\varphi[x{\leftarrow}c] \vdash \psi[x{\leftarrow}c]$ is true, this implies that $\mathcal{M}[w, c{\leftarrow}g(x)], w \Vdash^g \psi[x{\leftarrow}c]$, and consequently that $\mathcal{M}, w \Vdash^g \psi$.

The above proof sketch should be intuitive enough, but it omits a crucial point: the naive definition of $\mathcal{M}[w, c{\leftarrow}g(x)]$ is not concordant (i.e., the constant interpretation is not in sync over all accessible worlds), because the interpretation of a constant is being changed at w without being changed anywhere else. It is fine to propagate the change to the successors of w through the compatibility functions, and this would restore the concordance if w were the root of the model. However, when w is not the root, there is no clear solution other than dropping every other world from the model, which is what is done in Section 4.3. It works well because the satisfiability of a formula at w depends only on the model restricted to w and its successors.

We originally tried to implement this proof directly: restrict $\mathcal{M}$ to w and its successors and then replace the interpretation of c with $g(x)$ at w and with $\eta_{w,u}(g(x))$ at all the successors u of w. In this proof, the models are adequate every step of the way. However, implementing this strategy proved rather difficult. A model restricted to w and its successors is naturally defined as a regular model together with a non-informative statement to the effect that every world is either w or its successor. Then the next step would be to define a way to change the interpretation of a constant at the root and propagate it to all its successors. However, trying to do this on top of restricted models proved hard, in part because there is no built-in concept of root. Adequate models do not need to be rooted and we didn't want to include this restriction.

Instead, we changed the proof to postpone including non-informative elements as much as possible, in accordance with the simple types convention (Convention 9.3.1). The key insight is that only the final model needs to be adequate, and so we can change the constant interpretation first and only then restrict the worlds to obtain concordance. Here is also where the decision to have compatibility functions for every pair of worlds shines, as we'll soon see. We define the constant interpretation for the new model as follows.

Definition 10.5.3 (`replace_I`)**.** Let $\mathcal{M}$ be a model, w be a world, c be a constant, and d be an element of the domain of w. If I is the constant interpretation of $\mathcal{M}$, we define a new interpretation $I[w, c{\leftarrow}d]$ as follows. For a given world u, $c^{I[w,c\leftarrow d]_u} := \eta_{w,u}(d)$. The interpretation $I[w, c{\leftarrow}d]$ behaves like I for every other constant.

Note that the above definition is well-typed even if $\mathcal{M}$ is not an adequate model, and it won't necessarily lead to an adequate model unless w happens to be the root of $\mathcal{M}$. Note also that, if $\mathcal{M}$ is adequate, then $c^{I[w,c\leftarrow d]_w} = \eta_{w,w}(d) = d$, because $\eta_{w,w}$ is the identity in adequate models.

Finally, observe that if $\eta_{w,u}$ only existed when wRu, we could not have defined $I[w, c{\leftarrow}d]$ like this, for there would be no way to obtain an element of the domain of u in the cases where u was not a successor of w. Recall that we're going to drop these worlds later anyway, so it doesn't matter which domain element this is; only that we have one in hand. Even though this could have been implemented in other ways (for example, by designating a default element for each domain), this particular solution is elegant in its simplicity and symmetry, as there is no need to have a case distinction on whether wRu or not.

The first approximation to $\mathcal{M}[w, c{\leftarrow}g(x)]$ is then a copy of $\mathcal{M}$ with the constant interpretation replaced by $I[w, c{\leftarrow}g(x)]$. We can already prove the desired property about this model (`sat_replace`), namely that $\mathcal{M}[w, c{\leftarrow}g(x)]$ satisfies $\chi[x{\leftarrow}c]$ at w if and only if $\mathcal{M}$ satisfies χ at w, for any formula χ where c does not appear. It now remains to further modify $\mathcal{M}[w, c{\leftarrow}g(x)]$ so that it is adequate, by dropping all spurious worlds, obtaining $\mathcal{M}[w, c{\leftarrow}g(x)]|_w$. The final model, called `restrict_replace`, is finally adequate and allows us to prove the desired result, Lemma 10.5.4, which has the soundness of the constant elimination rule as a corollary.

Lemma 10.5.4 (`sat_restrict_replace`)**.** *Given a constant c, a formula φ where c does not appear,* *an adequate model $\mathcal{M}$, a world w, and a w-assignment g, we have:*

$$\mathcal{M}, w \Vdash^g \varphi \iff \mathcal{M}[w, c \leftarrow g(x)]|_w, w \Vdash^g \varphi[x \leftarrow c].$$

10.6 Relational completeness

The relational completeness theorem (Theorem 4.4.13, `completeness`) is proved via the description of a canonical model, world, and assignment for each pair of formulas φ and ψ such that $\varphi \not\vdash \psi$. The canonical model satisfies φ at the canonical world under the canonical assignment and does not satisfy ψ under the same conditions.

Theorem 4.4.13 (Constant domain completeness)**.** *Let Σ be a signature and φ, ψ formulas in Σ. If $\varphi \not\vdash \psi$, then there is an adequate, finite, irreflexive, rooted, and constant domain model $\mathcal{M}$, a world $w \in W$, and a w-assignment g such that:*

$$\mathcal{M}, w \Vdash^g \varphi \quad \textit{and} \quad \mathcal{M}, w \not\Vdash^g \psi.$$

Our formalization of this theorem mostly follows the proof described in Section 4.4 with the exception of the way of obtaining the set of worlds of the canonical model, which we describe in Section 10.6.3. We also comment on the formalization of the conservativity of QRC_1 when extra constants are added to the signature in Section 10.6.1 and on the implementation of the closure under a set of constants in Section 10.6.2.

10.6.1 Extended signature

For the completeness proof, it is essential to be able to conservatively extend the signature with extra constants. We do this twice: once to translate free variables to new constants and once to obtain an appropriately large domain for the canonical model.

The notion of extended signature by a finite number of new constants (`extend`) is straightforward, but formalizing that it does not increase the strength of the calculus (Lemma 4.2.6, `QRC1Proof_lift`) was tricky. The informal proof argues that a new constant appearing in a QRC_1 proof can be replaced by a fresh variable with no issue. However, this variable must be fresh for the whole QRC_1 proof. Instead of directly translating this requirement, we slightly modify it as follows.

Let $P(x, \varphi)$ be some property of a variable x and formula φ such that P is monotone on x, i.e., such that if $x \leq y$ and $P(x, \varphi)$ then also $P(y, \varphi)$.[2] Consider the following statement, which abstracts some details away in order to highlight the part that we wish to discuss:

> *Given a provable formula φ, there exists a Hilbert-style proof π of φ and a variable x fresh for π such that $P(x, \varphi)$.*

We cannot directly translate statements like this to Coq if we do not have an object representing the full proof π. Although such an object could be defined, the typical formalization of inductive

[2]Recall that we implement variables as natural numbers, so writing $x \leq y$ is allowed. We could alternatively make use of some bijection between the variables and the natural numbers for the same purpose.

axiomatizations does not provide one, it only lets us reason by induction on the construction. Instead, we prove the following adaptation:

> *Given a provable formula φ, there exists a variable x greater than all variables appearing in φ such that we have $P(x, \varphi)$.*

This takes advantage of the representation of variables as natural numbers. If x is greater than every variable appearing in a given formula φ, then x is fresh for φ. Furthermore, given a variable y greater than all variables appearing in a given context (be it a formula or a proof), $\max\{x, y\}$ is greater than the variables of both that context and φ, and so it is fresh for both.

We can now see that the two statements are equivalent. On the one hand, if x is fresh for a proof $\boldsymbol{\pi}$ of φ, then it is also fresh for φ in particular, and so $P(y, \varphi)$ holds for any $y \geq x$ by the monotonicity of P. On the other hand, if φ is provable then there exists a proof $\boldsymbol{\pi}$ of φ. We reason by induction on $\boldsymbol{\pi}$. If the proof $\boldsymbol{\pi}$ is built of a single axiom, then x is also fresh for the whole proof. If the final step of the proof is the consequence of a rule, then by the induction hypothesis there are proofs of the rule's assumptions and respective fresh variables. From x and the fresh variables of the previous proof steps, we can build a new variable fresh for the whole proof.

The above trick can be used in several contexts, as the abstraction suggests, and it was useful for the formalization of the conservativity of adding new constants to the signature. Still, that formalization was long and tedious, making use of several low-level lemmas about circumstances in which a given term is free for substitution in a formula, substitution in general, and free variables. It is likely that a nameless implementation of quantification (such as de Bruijn indices) would have significantly simplified the process.

10.6.2 Closure

During the proof of completeness, we use the notion of closure under a set of constants C, denoted by $\mathcal{C}\ell_C$ (Definition 4.4.1). This coincides with the usual notion of closure under subformulas in all cases except $\forall$. Given a formula $\forall\, x\, \varphi$ and a finite set of constants C, the closure under C of $\forall\, x\, \varphi$ includes the closure under C of $\varphi[x{\leftarrow}c]$ for each $c \in C$:

$$\mathcal{C}\ell_C(\forall\, x\, \varphi) := \{\forall\, x\, \varphi\} \cup \bigcup_{c \in C} \mathcal{C}\ell_C(\varphi[x{\leftarrow}c]).$$

Note that $\varphi[x{\leftarrow}c]$ is not a subformula of $\forall\, x\, \varphi$. Thus, the formalization of $\mathcal{C}\ell_C(\varphi)$ cannot be achieved by structural recursion on φ, even though that is the first method one reaches for in this case. Instead, we define $\mathcal{C}\ell_C(\varphi)$ by recursion on the `depth` of φ, i.e., on the nesting depth of connectives appearing in φ. This works because the depth of $\varphi[x{\leftarrow}c]$ coincides with the depth of φ.

There are several ways of defining non-structurally recursive functions in Coq, such as the FunInd plugin (legacy code, not recommended for new projects [67]), Program [195], and the Equations plugin [197]. The definition of $\mathcal{C}\ell_C$ (`closure` in Coq) is relatively simple, so we used Program. However, this results in a term which is hard to use, because we typically want to reason about the shape of a formula and not about its depth. Thus, we prove two auxiliary results. First, that `closure` is extensionally equal to a direct translation of Definition 4.4.1, i.e., to our intuitive understanding of closure (`closure_eq`). Second, an induction principle for this definition that inducts on the shape of the formula (`closure_ind`). This avoids using the default induction on the depth of the formula, which feels less natural.

The definition of `closure`, the relevant auxiliary results, and the formalization of the observation that the maximum number of distinct constants per formula in $\mathcal{C}\ell_C(\Gamma)$ has upper and lower bounds independent of C (Remark 4.4.3, `constantcount_closurefs`) can be found in the `Closure.v` file.

10.6.3 Canonical model

The formalized proof of completeness follows the ideas presented in Section 4.4 up to the point of defining the canonical model, where it diverges. In Definition 4.4.9, we define the canonical model recursively for a given unprovable sequent as follows: we start by explicitly providing the root (up to a use of the Lindenbaum lemma) and then add new worlds via inspection of the existing ones (again making use of the Lindenbaum lemma). The Lindenbaum lemma (Lemma 4.4.4) roughly states that under some conditions there is a well-formed world extending a given pair of sets, but we do not provide a decidable algorithm to obtain this world, only a classical proof of its existence. Thus, even though we provide an algorithm to define the canonical model, it relies on a non-constructive oracle.

Directly translating the aforementioned definition of the canonical model to Coq posed a challenge. The natural induction principle over our Coq frames uses the leafs as base-cases instead of the root, so we cannot easily use it. In fact, we do not even have a reified notion of root. Even if those problems were overcame via alternative or additional definitions, there was one other issue: not having access to a constructive Lindenbaum lemma, we could not describe a full construction for the canonical model as an informative algorithm. The axiom of excluded middle is non-informative, and so the Lindenbaum lemma is non-informative too.

Since the overall completeness proof is already non-constructive, there is no reason to avoid classical reasoning to define the canonical model. The relation, domain, compatibility functions, constant interpretation, and predicate interpretation are all clear and pose no problem. The hard part is identifying the set of worlds.

Fortunately, we can restrict the worlds to binary partitions of some previously determined finite set Φ with some additional properties. In fact, the property of being a Φ-MCW pair (see either Section 4.4 or `wfpair`) is both necessary and sufficient for being a world. Clearly, there are only finitely-many binary partitions of Φ, so even if we cannot constructively identity which partitions are acceptable worlds, we are able to classically prove that there is an appropriate set of acceptable worlds via the following comprehension lemma.

Lemma 10.6.1 (`fset_seq_comprehension`). *Let Γ be a countable set of objects, Φ be a finite subset of Γ, and $P : \Gamma \to \texttt{Prop}$ be a predicate on Γ. Then there is a set $\Phi_P \subseteq \Phi$ such that for any $\gamma \in \Gamma$ we have $\gamma \in \Phi_P$ if and only if $P(\gamma)$ holds.*

One of the consequences is that our informative Coq definition of canonical model (`crawModel`) is parametrized not only on the formulas φ and ψ for which the model is meant to be canonical, but also on a set of worlds. This set is then instantiated during the completeness proof, where we have access to the non-informative Lindenbaum lemma.

As a final note, we observe that the definition of the set of worlds as the set of well-formed partitions of Φ is similar to the corresponding definition in the proof of relational completeness for RC [36]. It is also a simplification of the proof presented in Section 4.4. We missed it earlier because it only works when the canonical model has a constant domain and our original canonical model (described in [13]) did not. The proof presented in Section 4.4 does not differ from the one presented in [13] on this particular matter, and so misses out on this simplification.

10.7 Decidability

We saw in Corollary 4.4.14 that QRC_1 is decidable via its finite model property. In this section, we go into detail on this proof and on its formalization in Coq. We rely on Post's theorem, which states as follows.

Theorem 10.7.1 (Post [184])**.** *If a set and its complement are both computably enumerable, then both are decidable.*

Post's theorem is not constructively provable (it is equivalent to Markov's principle [202]), but does hold classically. We already used the axiom of excluded middle to prove the finite model property, so accepting Markov's principle for the proof of decidability is inconsequential.

We use a formalization of Post's theorem available in the Library of Undecidability Proofs [103] that assumes the axiom of excluded middle (`bi_rec_enum_t_dec`). In order to make use of it, we need to show that both QRC_1-derivability and QRC_1-non-derivability are enumerable (in the sense of `rec_enum_t`). In other words, we need to find a function e : `nat` $\to$ `formula` $\to$ `formula` $\to$ `bool` such that there is n : `nat` for which $e\ n\ \varphi\ \psi =$ `true` if and only if $\varphi \vdash \psi$, and similarly for $\varphi \nvdash \psi$. Then the decidability of QRC_1 (`QRC1Proof_dec`) easily follows.

10.7.1 Derivability

In order to see that QRC_1 provability is computably enumerable, we define a decidable notion of n-bounded provability $\vdash_n$ and then prove that for any formulas φ and ψ we have $\varphi \vdash \psi$ if and only if there is some n such that $\varphi \vdash_n \psi$. The notion of n-bounded provability is then trivially an appropriate enumerator (a statement that is formalized as `QRC1Proof_enum`).

We first comment on why the calculus we already have (Definition 4.2.1, `QRC1Proof`) is not trivially proved to be decidable. One of the issues is the cut rule:

$$\text{if } \varphi \vdash \psi \text{ and } \psi \vdash \chi, \text{ then } \varphi \vdash \chi.$$

Note that ψ does not appear in the conclusion of the rule, so if we are doing proof search starting from the conclusion, there are *a priori* infinitely-many candidates ψ to try. Some of the other rules have similar issues with formulas, terms, or both. It is possible that there are clever observations that reduce the search space to a finite size, which would lead to a different decidability proof for QRC_1. However, here we simply bound the search space directly. This works for our purposes because any QRC_1 proof is finite and includes formulas and terms in finite languages. Thus, for any QRC_1 proof there is a bound n that would allow us to repeat the proof as an n-bounded proof, and n-bounded proofs obviously work as regular proofs as well.

We now define the concept of n-bounded term and n-bounded formula, for which we first need notions of size for both. The size of a variable is defined as its representation as a natural number. We are already working with a finite signature, so there is no need to restrict the search space for constants, and as such we define the size of any constant as zero. Then an n-bounded term (`bterm`) is simply a term with size less than or equal to n.

The size of a formula is determined by two factors: the maximum size of the terms appearing in the formula, and its maximum depth of nested connectives (`depth`). For ease of use, we define bounded formulas in two steps. A raw n-bounded formula (`rawBformula`) is defined inductively similarly to a regular formula, except all the terms appearing in it must be n-bounded. A (non-raw) n-bounded formula (`bformula`) is a raw n-bounded formula that furthermore has depth at most n.

We are now ready to define the bounded QRC_1 calculus. We write $|t| \leq n$ to denote that t is an n-bounded term and $|\varphi| \leq n$ to denote that φ is an n-bounded formula.

Definition 10.7.2 ($\vdash_n$, `bQRC1Proof`)**.** Let Σ be a finite signature and φ, ψ, and χ be any formulas in $\mathcal{L}^+_{\Box\forall\mathsf{c}}$ with the signature Σ. The axioms and rules of bounded QRC_1 are the following:

(i) if $\varphi \vdash \psi$ is a QRC_1 axiom, then $\varphi \vdash_0 \psi$;

(ii) if $\varphi \vdash_n \psi$, then $\varphi \vdash_{n+1} \psi$ (monotonicity);

(iii) if $\varphi \vdash_n \psi$ and $\varphi \vdash_n \chi$, then $\varphi \vdash_{n+1} \psi \wedge \chi$ (bounded conjunction introduction);

(iv) if $\varphi \vdash_n \psi$ and $\psi \vdash_n \chi$, then $\varphi \vdash_{n+1} \chi$, for $|\psi| \leq n+1$ (bounded cut);

(v) if $\varphi \vdash_n \psi$, then $\Diamond\varphi \vdash_{n+1} \Diamond\psi$ (bounded necessitation);

(vi) if $\varphi \vdash_n \psi$, then $\varphi \vdash_{n+1} \forall x\, \psi$, for $x \notin \mathsf{fv}(\varphi)$ (bounded quantifier introduction on the right);

(vii) if $\varphi[x{\leftarrow}t] \vdash_n \psi$, then $\forall x\, \varphi \vdash_{n+1} \psi$, for $|t| \leq n+1$ and t free for x in φ (bounded quantifier introduction on the left);

(viii) if $\varphi \vdash_n \psi$, then $\varphi[x{\leftarrow}t] \vdash_{n+1} \psi[x{\leftarrow}t]$, for t free for x in φ and ψ and $|\varphi| \leq n+1$, $|\psi| \leq n+1$, $|x| \leq n+1$, $|t| \leq n+1$ (bounded term instantiation);[3]

(ix) if $\varphi[x{\leftarrow}c] \vdash_n \psi[x{\leftarrow}c]$, then $\varphi \vdash_{n+1} \psi$, for c not in φ nor ψ and $|x| \leq n+1$ (bounded constant elimination).

[3]See the proof of Lemma 10.7.3 for motivation.

To a first approximation, one can think of the bound as measuring the proof depth. However, this is not literally the case because the sizes of some terms and formulas are also restricted. We comment on the decisions behind this axiomatization during the proof of its decidability, which follows.

Lemma 10.7.3 (`bQRC1ProofP`). *The n-bounded calculus is decidable, i.e., there is an effective and terminating algorithm deciding whether $\varphi \vdash_n \psi$ or $\varphi \nvdash_n \psi$ for any given n and formulas φ, ψ in a finite signature.*

Proof. By induction on n. Note that $\varphi \vdash_0 \psi$ if and only if $\varphi \vdash \psi$ is an axiom of QRC_1. All the axioms of QRC_1 are easily identifiable from the formulas φ and ψ, which concludes the base case.

The monotonicity, bounded conjunction introduction, bounded necessitation and bounded quantifier introduction on the right rules have only subformulas of φ and ψ in their premises, so deciding whether $\varphi \vdash_{n+1} \psi$ through one of those rules as the last step in an $(n+1)$-bounded proof is straightforward from the induction hypotheses.

The other rules all have some new term or formula appearing in their premise. For example, in order to check whether $\varphi \vdash_{n+1} \psi$ as a consequence of bounded cut, we must check whether there is a formula δ with $|\delta| \leq n+1$ such that both $\varphi \vdash_n \delta$ and $\delta \vdash_n \psi$. This is precisely why we bound the size of δ, as that means there are only finitely many formulas δ available as cut formulas.

Similar reasoning applies to the rest of the rules. We observe that there is no restriction on c in the bounded constant elimination rule because the signature is assumed to be finite, so there are only finitely-many constants to check. Note also that the bounded term instantiation rule includes generic restrictions on the size of the formulas φ and ψ when we already know both $\varphi[x\leftarrow t]$ and $\psi[x\leftarrow t]$, and so we could further reduce the search space for φ and ψ. However, implementing such a restriction would be costly and offer no benefit, since we do not attempt to optimize the decision procedure in any way.

The Coq implementation of the decision procedure sketched above is `bQRC1Proof_bool`. □

We can now easily prove the correctness of bounded QRC_1, which directly implies enumerability, as already noted above.

Lemma 10.7.4 (`QRC1Proof_bQRC1Proof`). *Given formulas φ and ψ, we have $\varphi \vdash \psi$ if and only if there is a number n such that $\varphi \vdash_n \psi$.*

Proof. Each direction follows from a straightforward induction on the respective derivability notion. □

10.7.2 Non-derivability

In order to formalize that the unprovable sequents are computably enumerable, we take advantage of the following formalization of the completeness theorem.

Theorem 10.7.5 (`explicit_completeness`). *Let φ and ψ be QRC_1 formulas such that $\varphi \nvdash \psi$. Then there is a (finite) set of worlds W and a world $w \in W$ such that:*

$$(\texttt{cmodel}\ W), w \Vdash^{\mathsf{cg}\ w} \varphi \quad \textit{and} \quad (\texttt{cmodel}\ W), w \nVdash^{\mathsf{cg}\ w} \psi,$$

where `cmodel` *is the adequate canonical model and* `cg` *is the canonical assignment.*[4]

The helpful thing about this formulation of the completeness theorem is that it provides a counter-model for any unprovable sequent using only a finite set of worlds and one particular world. We already know how to enumerate worlds and finite sets of worlds in Coq because the type of the worlds is trivially a `countType`. These types have a canonical `nat` encoding and decoding. Thus, there is no need to describe how to enumerate models in general, which would be much more cumbersome.

We define the enumerating function e, called `enumerate_cmodels` in Coq, approximately as follows:

$$e\ n\ m\ \varphi\ \psi := ((\texttt{cmodel}\ W_n), w_m \Vdash_{\mathtt{b}}^{\mathtt{cg}\ w_m} \varphi)\ \&\ ((\texttt{cmodel}\ W_n), w_m \nVdash_{\mathtt{b}}^{\mathtt{cg}\ w_m} \psi),$$

where W_n is the n-th finite set of worlds, w_m is the m-th world of that model,[5] $\Vdash_{\mathtt{b}}$ is the boolean version of the satisfaction relation (`satb`), and $\&$ is the boolean conjunction. Thus, $e\ n\ m\ \varphi\ \psi = \texttt{true}$ if and only if n is the code of a finite set of worlds and m is the code of a world such that `cmodel` W_n is a model and w_m is a world of that model where φ is satisfied under the `cg` w_m assignment and ψ is not. Via further coding, we can represent the numbers n and m as a single number.

Finally, to see that non-provable QRC$_1$ sequents are enumerated by e, note that if $\varphi \nvdash \psi$, then there is a counter-model by completeness, so there are n and m such that $e\ n\ m\ \varphi\ \psi = \texttt{true}$. On the other hand, if there are some such n and m, then we have an example of a model where φ is satisfied and ψ is not, so by soundness it must be that $\varphi \nvdash \psi$. The formalization of this result is called `nQRC1Proof_enum`.

[4]Besides the explicitly shown parameters W and w, these functions are both parametrized on φ and ψ as well, and `cg` is additionally parametrized on W.

[5]In practice, either of W_n or w_m may be undefined, in which case $e\ n\ m\ \varphi\ \psi := \texttt{false}$.

11 Formalized UTC

Contents

11.1 Introduction

Coordinated Universal Time (UTC) [138] is the current world standard for keeping time. Although it uses atomic time, it is designed to stay close to solar time, and as such it includes leap seconds. The number of seconds in a minute can be either 59 (if there is a negative leap second), 60, or 61 (if there is a positive leap second). The need for a new leap second is somewhat unpredictable, so the International Earth Rotation and Reference Systems Service announces whether there will be one about six months in advance. The convention is to have at most two leap seconds per year, as the final second of either June 30 or December 31. As of 2023 there have been 27 positive leap seconds and no negative ones [134].

The vast majority of software uses Unix time [132], which is an implementation of UTC without leap seconds. This is fine for many use cases, and understandable given the unpredictability of UTC for future moments. However, it conflicts with regulations that explicitly require UTC, such as Regulation (EU) 2016/799 [90]. For software related with these regulations such as Police Controller [119], using a Unix time implementation such as Microsoft's [173] does not follow the letter of the law. It may seem like 27 seconds are not enough to meaningfully change anything, but they can be, as seen in Section 7.3.

It was in this context that FV Time [7] was developed. It is a Coq library for time management, implementing both Unix time and UTC. The name alludes to Formal Vindications S.L., which sponsored this thesis and is the owner of FV Time and of the related FVTM executable. FV Time includes conversions between time, represented as a 7-tuple of year, month, day, hour, minute, second, and proof of existence in the chosen paradigm (to avoid ill-formed tuples such as the ones including February 30), and timestamps, represented as the number of seconds since a chosen epoch (for example, year 0, or year 1970, which is the Unix epoch). We also define dates and datestamps, which are the respective concepts without information on the hour, minute, and second. Leap seconds are tracked via a modifiable parameter, which can be empty for Unix time or updated as appropriate to keep pace with UTC.

FV Time is not the first library to implement UTC (see for example [114, 74, 152, 169]), but to the best of our knowledge it is the first formally verified one either for Unix time or for UTC. This chapter, based on [11], reports on the development of FV Time from high-level specifications to executable code integrated with other software. It can serve as a roadmap for other similar verification projects.

11.2 FV Time

The main goal of FV Time is to provide verified functions translating between time in a human-readable format and timestamps. It also provides functions for time arithmetic (for example, calculating the number of days between two dates). In this section we give a very brief overview of the plan for the rest of the chapter and of the file structure of the library, summarized in Figure 11.1.

The `calendar.v` file describes the main datatypes, such as what it means to be a (human-readable) date and time (Section 11.3). It also specifies the expected behavior of the translating

$$\begin{tikzcd}
\texttt{calendar.v} & & \\
\texttt{Hinnant.v} \arrow[u, "\text{implements}"] & & \texttt{formalTime.v} \arrow[ll, dotted, "\text{uses}"'] \\
\texttt{HinnantR.v} \arrow[u, "\text{refines}"] & & \texttt{formalTimeR.v} \arrow[u, "\text{refines}"'] \arrow[ll, dotted, "\text{uses}"'] \\
& \texttt{fvtm\_extraction.v} \arrow[ul, "\text{refines}"] \arrow[ur, "\text{refines}"'] & \\
& \texttt{fvtm\_extraction\_correct.v} \arrow[u, "\text{proves correct}"] &
\end{tikzcd}$$

Figure 11.1: The file structure of FV Time. There is an extra file, `doe_of_yoeK.v`, that only contains auxiliary proofs and definitions and doesn't appear in the diagram.

functions by providing simple implementations (Section 11.4.1). In `Hinnant.v`, there are alternative and more efficient implementations of these translating functions (Section 11.4.2) and proofs that they coincide with their specification (Section 11.4.3). The latter implementations of the datestamp algorithm and its inverse are inspired by the ones described by Hinnant [126], hence the name. The time arithmetic functions are specified and implemented in `formalTime.v` (Section 11.5).

Since these algorithms are meant to be extracted from Coq to OCaml for efficient execution, we provide a type refinement for each of them in the `HinnantR.v` and `formalTimeR.v` files (Section 11.6). In other words, there are two versions of each algorithm: one based on proof-friendly datatypes, and one based on extraction and computation-friendly ones. These two versions are proven equivalent under some assumptions with the aid of the FV Prim63 to MathComp library, which describes the precise relationship between the types. It was developed purposefully for this refinement, but can be reused in any other work dealing with the same types. The relevant files are listed in `all_prim63_mathcomp.v`.

Next, `fvtm_extraction.v` redefines every relevant function using simple types and error management, and these new versions are proven correct in `fvtm_extraction_correct.v` (Section 11.7). The extracted code can be used either directly as an OCaml library, or through the FVTM command-line interface (Section 11.8).

Finally, we describe a custom set of automation tactics used throughout the formalization (Section 11.9). These tactics are called FV Check Range and apply to any decidable goal on primitive integer intervals with up to three variables. The relevant files are `check_range_Uint63.v` and `check_range_Sint63.v` for unsigned and signed primitive integers, respectively.

11.3 Main data types

The central data type in FV Time is a representation of moments in time in UTC or Unix (depending on the assumed leap seconds) up to the end of the year 9999,[1] which we call `time`. Under the hood it is simply a 6-tuple of natural numbers representing a given year, month, day, hour, minute, and second,[2] together with a proof that the tuple in question forms an existing time, in which case we say it is valid. What counts as an existing time depends on the parametrized list of leap seconds.

For convenience and modularity's sake, we define three other relevant types: `date`, `rawDate`, and `rawTime`. A `date` is the part of the `time` with only the year, month, and day, together with a proof that it is valid, i.e., that it exists in UTC. The `raw` types are simply the tuples without the proofs. Thus, January 32, 2000 could be represented as a `rawDate` but not as a `date`.

We encode the list of leap seconds as a parameter that can be updated each time a new leap second is announced. The list is actually a list of pairs, where each pair has a `rawDate` (indicating that a leap second occurs on the last moment of that day) and a Boolean value (where `false` means that it's a positive leap second and `true` that it's a negative one). Since we treat the list as a parameter with unknown contents, it can be instantiated in any way. However, we only prove correctness of the time functions when two further assumptions hold: the list must be sorted with respect to the strict (lexicographic) order of `rawDate`s (in particular it doesn't include repeated values), and all the `rawDate`s in it must be valid. Note that we specifically avoid `date`s in the list of leap seconds in accordance with the simple types convention (Convention 9.3.1).

Leap seconds are only ever announced for the last second of June 30 or December 31. However, we accept leap seconds on any day (always on the last second of that day). There is no attempt to verify that the leap seconds provided by a user are the real ones, although we do provide an updated list with each version of FV Time for users who want to use UTC.

The `raw` types are used in the implementation of every function that operates on dates or times, in accordance with the simple types convention. It is then possible to compute the `datestamp` of January 32, 2000, but we do not wish to prove any facts about the datestamps of such ill-formed dates. For that reason, we use the valid (non-`raw`) versions in the specifications and theorem statements. There is then a disconnect between the specification and the implementation, since they refer to different types. This is easily solved using coercions, i.e., automatically inserted translations between one type and another.

We use a number of coercions in our development, mostly between types and their subtypes, as described in Figure 11.2. This leads to ambiguity if a `rawDate` is required and a `time` is provided, as there are two possible (equivalent) ways to obtain a `rawDate` from a `time`. This issue was already described in Section 10.4 and we solve it similarly, declaring `rawDate_of_date` ∘ `date_of_time` as the canonical choice.

[1] It is useful to have a finite number of acceptable `time`s, which is explained further in Section 11.4.1. The year 9999 has no particular significance and different end years could be substituted, although if they were large enough there might be problems with overflow.

[2] We currently take the second to be the smallest unit, but a finer-grained resolution could be easily implemented.

$$\begin{tikzcd}
 & \texttt{time} \arrow[dl, "\texttt{date\_of\_time}"'] \arrow[dr, "\texttt{rawTime\_of\_time}"] & \\
\texttt{date} \arrow[dr, "\texttt{rawDate\_of\_date}"'] & & \texttt{rawTime} \arrow[dl, "\texttt{rawDate\_of\_rawTime}"] \\
 & \texttt{rawDate} &
\end{tikzcd}$$

Figure 11.2: A representation of the four main data types in FV Time and of the coercions between them. Each arrow from `X` to `Y` represents the coercion `Y_of_X`.

11.4 Main functions

The backbone of FV Time are the translations between times and timestamps, which themselves depend on translations between dates and datestamps. In this section we focus on the specification and first implementation of these four functions, as well as on the proofs that they coincide on well-formed inputs. The names of these terms are listed in Table 11.1 with hyperlinks to the relevant online documentation, and their types are made explicit in Table 11.2 on page 164. Further implementations with more practical types are discussed in Section 11.6.

Table 11.1: Main functions in FV Time together with the name of the theorem stating that the implementation meets the specification. See also Table 11.2 on page 164, which includes the types of each of these terms.

	Specification `calendar.v`	Implementation `Hinnant.v`	Correctness `Hinnant.v`
date	`datestamp` `from_datestamp`	`datestamp` `from_datestamp`	`datestampE` `from_datestampE`
time	`timestamp` `from_timestamp`	`timestamp` `from_timestamp`	`timestampE` `from_timestampE`

The relevant files are `calendar.v` and `Hinnant.v`. The former includes the basic definitions of dates and times, as well as the specifications of the main translator functions. The latter includes the efficient implementations and the correctness proofs of those functions. Note that the specification and implementation of a given function have the same name, so we use the name of the file to clarify which we mean at any given time. Similarly, some lemma names coincide and are clarified as well.

11.4.1 Specification

The specifications of the main functions were primarily chosen to be intuitive. Thus, the `calendar.datestamp` of a `date` d is the size of the set of dates strictly smaller than d, and similarly for `calendar.timestamp`, where the order relations on dates and times are defined as expected (lexicographic). These definitions use the notion of cardinality of a finite set, which is defined in MathComp's `fintype` library [164]. We arbitrarily set the maximum date as December 31, 9999 so that our types for valid dates and times could be declared as a `finType` (i.e., a type with finitely-many inhabitants) and hence benefit from this library's theory. This also allowed us to prove the absence of overflow in the refinements of our functions to 63-bit numbers, as explained in Section 11.6.

Given the notions of `next_date` and `next_time`, which are described in the next paragraph, we define `calendar.from_datestamp` for a number n as the n-th iteration of `next_date` after the minimum date, and similarly for `calendar.from_timestamp`.

The definition of `next_date` is the expected one, with its only particularity being that the successor of the maximum date is the minimum date. This cyclic behavior was chosen in order to maintain the invariant that the successor of a valid date is always a valid date. We also define `next_time` to be cyclic, and we underline that this definition depends on the list of leap seconds (see Section 11.3), as do essentially all the functions related to time.

11.4.2 Implementation

`Hinnant.datestamp`

If every month had the same number of days and every year the same number of months, computing the datestamp of a given date would be as straightforward as multiplying each date component by its corresponding number of days and adding everything. Even with different lengths for different months it would not be particularly complicated, but the existence of leap years means that some more care must be taken. Notably, the pattern of leap years repeats every 400 years, so we divide the years into 400-year eras. Furthermore, we internally use shifted years that start on the first day of March and end on the last day of February, as inspired by Hinnant [126]. Thus, the leap day, if it exists, is the last day of the shifted year and doesn't influence the calculation of the datestamp of any day in that year other than itself.

The only other main part of `Hinnant.datestamp` is calculating how many days there are between the start of a (shifted) year and the first day of each (shifted) month. This can be represented as a table, but it turns out there is a simple linear equation interpolating this table, so we use that instead of storing the table in memory.

`Hinnant.from_datestamp`

As in `Hinnant.datestamp`, we divide years into 400-year eras and shift everything so that years start on March 1. With this framing, the `Hinnant.from_datestamp` algorithm is more obviously the inverse of `Hinnant.datestamp`, a fact which we rely on for the correctness proofs.

Given the number of days since the beginning of the era, finding the year must take into consideration leap years, and finding the month must take into consideration the varying number of days in each month. For the latter we use a linear interpolant of the table matching days in a year to months instead of using the table directly.

`Hinnant.timestamp`

This is a natural extension of `Hinnant.datestamp`. In the absence of leap seconds, we could simply add the product of each time component by the amount of seconds in that component (60 seconds per minute and so on). With leap seconds, we need to additionally calculate the offset generated by them, i.e., the number of extra seconds that must be added or subtracted to the leap second-less timestamp. Such an offset is relatively easy to calculate for a given date d, for we can simply count the number of leap seconds that happened prior to d, and then check whether each leap second was positive or negative to obtain a final offset. This is accomplished by `offset_rd`.

Since the leap second offset can *a priori* be negative (even though no negative leap seconds have been declared as of July 2023), we first calculate the timestamp over the integers and then take its absolute value. This works because even before taking the absolute value we know that the timestamp is positive due to our restrictions on leap seconds: we allow at most one leap second per day (an unimportant restriction, since the international convention allows at most two leap seconds per year). Since there are less days than seconds in any given amount of time, it is not possible to have enough negative leap seconds to obtain a negative timestamp.

`Hinnant.from_timestamp`

Once we know how to calculate the date corresponding to some datestamp, calculating the time corresponding to some timestamp (i.e., to some number n of seconds since the epoch) is straightforward in the absence of leap seconds. Thus, we first subtract the relevant offset from n and then proceed as if there were no leap seconds.

Obtaining the offsets for this function is slightly more complicated than for `Hinnant.timestamp`, since our list of leap seconds is a list of dates and not of timestamps. Thus, the offset calculator for `Hinnant.from_timestamp`, called `offset_ts`, first computes the `Hinnant.timestamp` of the final second of each date in our list of leap seconds (leap seconds are always the final second of each day by international convention), and then proceeds similarly to the offset computation for `Hinnant.timestamp`.

11.4.3 Correctness

The specification and implementation of the main functions differ significantly, and so it is hard to directly prove that they match. Instead, we make use of lemmas showing that certain functions are the (left) inverse of others (also known as canceling lemmas) and the following simple remark.

Remark 11.4.1. Let T and U be types, and $f_1, f_2 : T \to U$ be functions. If there is a function $g : U \to T$ that is simultaneously a right inverse of f_1 and a left inverse of f_2, then f_1 and f_2 are (extensionally) equal.

We summarize the correctness proof for `timestamp` (Theorem 11.4.2) as an example. The actual Coq statement includes our standard assumptions on the shape of the list of leap seconds (see Section 11.3 and Table 11.2), omitted here. Note that the theorem statement is about valid `time`s; we make no claim about non-existing times such as any moment during January 32 of any year. The other correctness statements can be found in Table 11.2, together with links to their proofs.

Theorem 11.4.2 (`timestampE`). *For every* `time` *t:*

$$\texttt{Hinnant.timestamp}\ t = \texttt{calendar.timestamp}\ t.$$

Proof. By Remark 11.4.1 it suffices to find a suitable function bridging the implementation and specification of timestamp. We used `calendar.from_timestamp` and the canceling lemmas `calendar.timestampK` and `cal_from_timestampK` (see Figure 11.3 for a schematic representation of their statements). □

$$\begin{tikzcd}
\texttt{calendar.timestamp} & \texttt{Hinnant.timestamp} \arrow[dl, "\texttt{cal\_from\_timestampK}"'] \\
\texttt{calendar.from\_timestamp} \arrow[u, "\texttt{calendar.timestampK}"] & \texttt{Hinnant.from\_timestamp} \arrow[u, "\texttt{Hinnant.timestampK}"']
\end{tikzcd}$$

Figure 11.3: The implementation and specification of the main time functions together with the canceling lemmas used to prove their correctness. Each arrow from f to g represents the proof that f is a left inverse of g.

Given the above strategy, the main challenge becomes proving the canceling lemmas. There are three relevant lemmas for time, depicted as the arrows in Figure 11.3, and three analogous ones for dates, used as a stepping stone for the time ones and not shown here. We briefly comment on each of these three results.

On the specification side, `calendar.timestampK` states that `calendar.from_timestamp` is a left inverse of `calendar.timestamp`. Since this is a statement about two specifications that behave nicely with respect to proofs, there were no great difficulties in proving it.

The bridge between specification and implementation is provided by `cal_from_timestampK`, which states that `calendar.from_timestamp` is a right inverse of `Hinnant.timestamp`. The specification for `calendar.from_timestamp` is very simple (just iterating `next_time`), and so this proof follows without too much difficulty using basic arithmetical facts.

Finally, on the implementation side, `Hinnant.timestampK` (needed for proving `from_timestampE`, which is the correctness theorem for `from_timestamp`) states that `Hinnant.from_timestamp` is a left inverse of `Hinnant.timestamp`. Its proof is the most intricate of the three if done with pen and paper, due to the ubiquitous presence of Euclidean division, which doesn't have an inverse. However, once we developed the automation tool FV Check Range (see Section 11.9), this proof was significantly eased.

11.5 Time arithmetic

The usefulness of time libraries comes in part from the ability to perform time arithmetic, such as finding the date three weeks from today, or the amount of hours between two given moments. To a first approximation, these operations can be done by translating the times to timestamps (or the dates to datestamps), doing the desired arithmetical operation on the natural numbers representing those timestamps, and translating back the result. This is one of the reasons timestamps are useful.

The difficulty with time arithmetic is then not the calculus itself, but the corner-cases of the specifications. What should “one month from today” mean, exactly? Presumably, some day in the vicinity of 30 days from today, although it could also be anywhere from 28 to 31 and it would recognizably be “one month”. With leap seconds, not even expressions such as “35 minutes” are fully determined, because some minutes are longer than others.

Thus, we defined two sets of time arithmetic functions, and leave the decision of which one to pick for the library users. The first approach, leading to the so-called shift functions, uses the same specification as Microsoft [173], which allows FV Time to be used as its replacement. The second approach is a definition of a standard for durations called formal time and operations that manipulate fixed amounts of seconds.

11.5.1 Shift functions

The shift functions follow the Microsoft specification [173]. The idea is that adding n months to a given date is the same as adding n to the `month` component of that date, changing the `year` appropriately. This needs to be adapted if the target day does not exist in the resulting month, for example when adding 1 month to January 31. In that case, the result is the previous existing date, February 28 in this case, or 29 if it’s a leap year. Adding other amounts of time is analogous.

More specifically, the shift function changes a component of the time, carrying to the left if necessary, and then if the result is invalid it performs corrections on the wrong component(s) to return the closest previous valid time that shares all the same components to the right of the wrong ones. For example, 2000-01-31 15:30:10 plus 13 months is 2001-02-28 15:30:10, because January plus 13 months is February of the following year, and there is no 31st of February; the closest previous existing day is 28 (since 2001 is not a leap year) and so the result is the same time (15:30:10) on February 28, 2001. The precise specification can be found in the lemma `shift_utc_monthsP` for months, and similarly for the other date-time components (`shift_utc_yearsP`, `shift_utc_daysP`, `shift_utc_hoursP`, etc.).

We accept negative integers as input too, so the shift functions effectively function as both addition and subtraction. Note however that this particular addition and subtraction are not mutual inverses: for example, January 31 plus one month and then minus one month is January 28 in a non-leap year.

11.5.2 Formal time

Sometimes we may want to add or subtract specific amounts of time instead of vague concepts such as “months” or “years”. This reduces to adding and subtracting seconds, since that’s the lowest precision available in our setting, and this is accomplished by `add_formal_seconds` (which also performs subtraction if given a negative number as input and coincides with `shift_utc_seconds`).

In order to spare the user from calculating or inputting $n \cdot 24 \cdot 60 \cdot 60$ when they wish to add n days to a date, we define the concept of base of units. For example, the base $(3600, 60)$ represents two different units: one that is 3600 seconds long and one that is 60 seconds long. These correspond to reasonable units for our concepts of hour and minute. Any strictly decreasing list is accepted as a base, but we define a standard one too, `G_FT_Conf`, which stipulates what we call formal units. A formal second is an atomic second, a formal minute is 60 formal seconds, a formal hour is 60 formal minutes, a formal day is 24 formal hours, a formal month is 30 formal days, and a formal year is 365 formal days. A duration represented in the `G_FT_Conf` base is called a `formalTime`.

Adding durations can then be accomplished in one of three ways: with `add_formal_years` (or `add_formal_months` and so on, depending on the desired units) if there is no need to mix units; with `add_formal` if the duration is expressed as a `formalTime`; or with `add_formal_generic` if the duration is expressed in a different base.

As an example, if one wishes to add two 31-day months to t, one can either:

- use `add_formal_month` to add two formal months and then `add_formal_day` to add the extra two days;
- use `add_formal` with $(0, 2, 2, 0, 0, 0)$ as the `formalTime`; or
- use `add_formal_generic` with $31 \cdot \texttt{dur\_day}$ as a base and 2 as the number of units.

With the concept of duration, we also define subtraction between two moments in time. The result can be expressed either as the number of seconds between the two times (`sec_time_difference`), as a `formalTime` (`time_difference`), or in a user-specified base (`time_difference_generic`).

11.6 Type refinements

As explained in Section 9.5, the unary types for the natural numbers and integers provided by MathComp are less than adequate for computation and memory management. Instead of extracting the FV Time functions directly, we first refine them by changing those types into the 63-bit primitive integers available in Coq (unsigned and signed, respectively). Primitive integers are an ideal type for this task because they are built into OCaml at the processor level. Their Coq instance is defined at the kernel level as the OCaml version, and their extraction is straightforward.

However, the refinement in itself is not trivial because the natural numbers are not isomorphic to the 63-bit unsigned integers: the latter are a finite type behaving like arithmetic modulo 2^{63}. As mentioned in Section 11.3, we have a maximum time, namely December 31, 9999 23:59:59 (or at most 23:59:60, depending on the parametrized list of leap seconds). The timestamp of this time is on the order of 2^{38}, so overflow should not be (and in fact isn't) a concern. However, this argument is not precise enough to prove the correction of the refinement.

The refined main data types and functions are defined and proven correct in `HinnantR.v`, and the functions related with time arithmetic in `formalTimeR.v`. The proofs are quite tedious and in some cases challenging. Ensuring the equivalence between the original and the refined versions implies ensuring that all of the intermediate steps do not overflow, or finding bounds for which they don't. Automation was key for several lemmas, specially ones with nested Euclidean division. We go into more detail on this in Section 11.9.

One other reason for difficulty (or at least cumbersomeness) was the non-existence of a library linking the types we were working with, so the relationship between MathComp natural numbers and unsigned primitive integers (and also between MathComp integers and signed primitive integers) had to be provided essentially from scratch. This led to the development of FV Prim63 to MathComp [7], a generic and stand-alone library designed exactly for this purpose.

11.7 Extraction

We use extraction to go from the verified Coq code to easily executable OCaml code (see Section 9.7). The terms to be extracted are all defined in a stand-alone file `fvtm_extraction.v` where the only imported libraries are the ones for lists, Booleans, and primitive integers. In particular, it doesn't import any MathComp libraries nor any of our several other files. It is as close to OCaml as possible while still being written in Coq. This is a strategy for simplifying the extraction process and obtaining clean and readable extracted code.

Since we are defining all data types and functions from scratch (albeit essentially copying the refined versions discussed in Section 11.6), we give them more sensible names in the context of the future extracted code. With no dependent types around it makes no sense to talk about raw dates, so we might as well call them dates. Thus, `fvtm_extraction.date` is defined like `HinnantR.rawDate`, and similarly for `fvtm_extraction.time`.

The major difference is the introduction of the `possibly` type, which is a more detailed option type that is either `Result` or one of several different errors such as `InvalidDate` or `Overflow`. This allows us to write useful error messages for the OCaml wrapper (see Section 11.8) while keeping the checking for those errors entirely within the verified part of the program. For example, the function `utc_timestamp` is defined as follows:

```
Definition utc_timestamp (t : time) : possibly uint :=
  if valid_time t then Result (utc_timestamp_plain t)
  else InvalidTime t.
```

In other words, it receives a `time` (which in this context is just a 6-tuple of unsigned primitive integers), checks whether it represents a UTC time with `valid_time`, and if so outputs the result of actually computing its UTC timestamp with `utc_timestamp_plain`. If the `time` is invalid, the output is `InvalidTime` instead. The `utc_` prefix highlights that the epoch is the Unix one (1970-01-01 00:00:00) and that the current UTC list of leap seconds is being used. This list is kept up to date manually via new releases of FV Time. The user is free to provide their own epoch and list of leap seconds with `timestamp`, although we do not provide a version of that function with error handling.

The actual extraction occurs from a different file, `extraction_command.v`, which lists all the functions from `fvtm_extraction.v` to be extracted. We import both `ExtrOcamlBasic` and `ExtrOCamlInt63` from the Coq standard library here. The former is a small collection of well-accepted translations, such as mapping Coq's `bool` type to OCaml's, and other such mappings where the types are basically the same in both languages. The latter maps the Coq definitions of the primitive integers to the very same OCaml module used to implement them in the Coq kernel.

Finally, `fvtm_extraction_correct.v` presents the links between the functions to be extracted and their formalized counterparts. Here we do make use of the full FV Time library, showing the correctness of each `fvtm_extraction.v` function. This correctness amounts to both proofs of equality with the type refined versions, such as `timestampE`, and to further proofs of equality with the specifications provided in `calendar.v` and `formalTime.v` under appropriate assumptions on the input, such as `timestampR`. There are also lemmas specifying the intended behavior when the input is incorrect in a specific way, such as `utc_timestamp_InvalidTime`.

11.8 Command-line interface

The extracted code is an OCaml library with all the relevant functions of our development. However, OCaml is not the most popular programming language and communication with other languages is non-trivial. Hence, we wrote a command-line interface in OCaml that allows a user to compile the library as an executable and invoke it from the terminal or from other programming languages. It translates the `possibly` type into either the plain result in the `Result` case or into OCaml exceptions with helpful error messages in the other cases. There is also a parsing phase that can throw parsing-related errors.

This command-line version of the library is named FVTM (which stands for FV Time Manager) and is documented in [101], but the non-extracted source code and the executable are not publicly available. The conversions between UTC times and timestamps and the function calculating the difference between two times can be tried online [100].

FVTM is currently being used as one possible time library for Police Controller [119], which is proprietary software used for checking compliance with the laws mentioned in Part II. The alternative is the Microsoft time library [173], which is about two times faster. The relative slowness of FVTM is attributed to the interface with the FVTM executable and not with the implementation of the functions themselves.

11.9 FV Check Range

FV Check Range [7] is a set of tactics to automatically solve true decidable statements with up to three free variables bounded by a specific primitive integer range. Given a goal with base statement of type `bool` and at most three primitive integer variables[3] and the bounds on which to check them, the tactics identify the desired Boolean statement, generate a list with all the primitive integers in the relevant range, and use `vm_compute` [117] to confirm that the Boolean statement indeed holds for every number in range. The implementation of the tactics is simple and done with Ltac [80].

These tactics work rather fast, checking ranges with sizes on the order of 10^5 in hundredths of seconds and on the order of 10^7 in two or three seconds (showing a linear progression) in one of our machines.

[3]There is a different tactic for each number of variables to be checked, so adding support for more variables is not as simple as changing a parameter. It could be easily done by analogy, though.

We used FV Check Range at several points during the development of FV Time and describe here only a particular example:

$$\begin{gathered}\forall\ (x : \texttt{Uint63.int}),\ x < 146097 \to \frac{h(x)}{365} \cdot 365 + \frac{h(x)}{365 \cdot 4} - \frac{h(x)}{365 \cdot 100} \leq x, \\ \text{with } h(x) := x - \frac{x}{1460} + \frac{x}{36524} - \frac{x}{146096}.\end{gathered} \tag{11.1}$$

Note that the division operation in question is Euclidean division, so it is not always the case that $\frac{x}{y} \cdot y = x$.

As expected, we found that using automation was significantly easier and faster than translating informal proofs. In particular, the translation of our informal proof for (11.1) had 400 lines in Coq and took a non-trivial amount of time to compile, while the proof using FV Check Range is a one-liner and takes hundredths of seconds.

Notably, a translation of (11.1) to `ssrint` integers can be automatically solved with `mczify` [189], an extension of `micromega` designed to work with MathComp numbers. However, it can't yet be solved in the realm of binary or primitive integers. This is likely not a fundamental but a practical shortcoming that could be bridged with some work. Nonetheless, our tactics solve any kind of decidable goals on primitive integers, not only arithmetical expressions, and thus the scope is different from `micromega`'s.

Another relevant tactic is `interval` [171], which solves interval arithmetic goals on real numbers. It doesn't seem like it can solve a translation of (11.1) in particular. Even if it could, it would introduce a significant amount of overhead and unnecessary axioms due to the detour through the (classical) reals. The other automation tactics mentioned in Section 9.6 are also unable to solve (11.1), at least to the best of our attempts.

Table 11.2: Main functions in FV Time together with the name of the theorem stating that the implementation meets the specification and the type of each term.

	Specification calendar.v	Implementation Hinnant.v	Correctness Hinnant.v
date	datestamp : date → 'I_max_datestamp.+1	datestamp : rawDate → nat	datestampE : ∀(d : date), Hinnant.datestamp d = calendar.datestamp d
	from_datestamp : nat → date	from_datestamp : nat → rawDate	from_datestampE : ∀(n : nat), n ≤ max_datestamp → Hinnant.from_datestamp n = calendar.from_datestamp n
time	timestamp : ∀(ls : leapSeconds), time ls → 'I_(max_timestamp ls).+1	timestamp : leapSeconds → rawTime → nat	timestampE : ∀(ls : leapSeconds)(t : time ls), sorted Order.lt (unzip1 ls) → all valid_date (unzip1 ls) → Hinnant.timestamp ls t = @calendar.timestamp ls t
	from_timestamp : ∀(ls : leapSeconds), nat → time ls	from_timestamp : leapSeconds → nat → rawTime	from_timestampE : ∀(ls : leapSeconds)(n : nat), sorted Order.lt (unzip1 ls) → all valid_date (unzip1 ls) → n ≤ max_timestamp ls → Hinnant.from_timestamp ls n = calendar.from_timestamp ls n

References

[1] Aceto, L., Monica, D. D., Goranko, V., Ingólfsdóttir, A., Montanari, A., & Sciavicco, G. (2016). A complete classification of the expressiveness of interval logics of Allen's relations: the general and the dense cases. *Acta Informatica*, *53*(3), 207–246.

[2] Agda Development Team (2007). The Agda wiki.
URL `https://wiki.portal.chalmers.se/agda/pmwiki.php`

[3] de Almeida Borges, A. (2018). Worms in Coq.
URL `https://gitlab.com/ana-borges/WormsCoq`

[4] de Almeida Borges, A. (2022). Towards a Coq formalization of a quantified modal logic. In C. Benzmüller, & J. Otten (Eds.) *Proceedings of the 4th International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2022)*, (pp. 13–27). Haifa, Israel: CEUR.
URL `https://ceur-ws.org/Vol-3326/ARQNL2022_paper1.pdf`

[5] de Almeida Borges, A. (2023). Coq formalization of QRC1. Version 1.0.0.
URL `https://gitlab.com/ana-borges/QRC1-Coq/-/releases/v1.0.0`

[6] de Almeida Borges, A. (2023). QRC1 in Coq: summary. Version 1.0.0.
URL `https://ana-borges.gitlab.io/QRC1-Coq/v1.0.0/Summary.html`

[7] de Almeida Borges, A., Casals Buñuel, J., Conejero Rodríguez, J., González Bedmar, M., & Hermo Reyes, E. (2023). The FormalV Library. Version 1.2.0.
URL `https://gitlab.com/formalv/formalv/-/releases/1.2.0`

[8] de Almeida Borges, A., Casanueva Artís, A., Falleri, J.-R., Gallego Arias, E. J., Martin-Dorel, É., Palmskog, K., Serebrenik, A., & Zimmermann, T. (2023). Lessons for interactive theorem proving researchers from a survey of Coq users. In A. Naumowicz, & R. Thiemann (Eds.) *14th International Conference on Interactive Theorem Proving (ITP 2023)*, vol. 268 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (pp. 12:1–18). Bialystok, Poland: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
URL `https://drops.dagstuhl.de/opus/volltexte/2023/18374/pdf/lipics-vol268-itp2023-complete.pdf#page=189`

[9] de Almeida Borges, A., Conejero Rodríguez, J. J., Fernández-Duque, D., González Bedmar, M., & Joosten, J. J. (2019). The second order traffic fine: Temporal reasoning in European

transport regulations. In J. Gamper, S. Pinchinat, & G. Sciavicco (Eds.) *26th International Symposium on Temporal Representation and Reasoning (TIME 2019)*, vol. 147 of *Leibniz International Proceedings in Informatics (LIPIcs)*, (pp. 6:1–6:16). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
URL `http://drops.dagstuhl.de/opus/volltexte/2019/11364`

[10] de Almeida Borges, A., Conejero Rodríguez, J. J., Fernández-Duque, D., González Bedmar, M., & Joosten, J. J. (2021). To drive or not to drive: A logical and computational analysis of European transport regulations. *Information and Computation*, *280*.
URL `https://www.sciencedirect.com/science/article/pii/S0890540120301243`

[11] de Almeida Borges, A., González Bedmar, M., Conejero Rodríguez, J., Hermo Reyes, E., Casals Buñuel, J., & Joosten, J. J. (2022). FV Time: A formally verified Coq library. arXiv:2209.14227 [cs.SE].
URL `https://arxiv.org/abs/2209.14227`

[12] de Almeida Borges, A., & Joosten, J. J. (2018). The Worm Calculus. In G. Bezhanishvili, G. D'Agostino, G. Metcalfe, & T. Studer (Eds.) *Advances in Modal Logic 12*, (pp. 13–27). College Publications.
URL `http://www.aiml.net/volumes/volume12/deAlmeidaBorges-Joosten.pdf`

[13] de Almeida Borges, A., & Joosten, J. J. (2020). Quantified Reflection Calculus with one modality. In N. Olivetti, R. Verbrugge, S. Negri, & G. Sandu (Eds.) *Advances in Modal Logic 13*, (pp. 13–32). College Publications.
URL `http://www.aiml.net/volumes/volume13/deAlmeidaBorges-Joosten.pdf`

[14] de Almeida Borges, A., & Joosten, J. J. (2022). An escape from Vardanyan's Theorem. *The Journal of Symbolic Logic*, (pp. 1–26).
URL `https://www.cambridge.org/core/journals/journal-of-symbolic-logic/article/an-escape-from-vardanyans-theorem/03E13D4C282563F918AF77CF6E2830DA`

[15] Anand, A., Boulier, S., Cohen, C., Sozeau, M., & Tabareau, N. (2018). Towards certified metaprogramming with typed Template-Coq. In J. Avigad, & A. Mahboubi (Eds.) *Interactive Theorem Proving*, (pp. 20–39). Cham: Springer International Publishing.

[16] Appel, A. W. (2011). Verified Software Toolchain. In G. Barthe (Ed.) *Programming Languages and Systems*, (pp. 1–17). Berlin, Heidelberg: Springer Berlin Heidelberg.

[17] Appel, A. W. (2015). Verification of a cryptographic primitive: SHA-256. *ACM Trans. Program. Lang. Syst.*, *37*(2).

[18] Appel, A. W. (2022). Coq's vibrant ecosystem for verification engineering (invited talk). In *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, (pp. 2–11).

[19] Appel, K., & Haken, W. (1976). Every planar map is four colorable. *Bulletin of the American Mathematical Society*, *82*(5), 711–712.

[20] Artemov, S. N. (1985). Nonarithmeticity of truth predicate logics of provability. *Doklady Akademii Nauk SSSR*, *284*(2), 270–271. In Russian. English translation in Soviet Mathematics Doklady 33:403–405, 1985.

[21] Artemov, S. N. (1986). Numerically correct logics of provability. *Doklady Akademii Nauk SSSR*, *290*(6), 1289–1292. In Russian.

[22] Artemov, S. N., & Beklemishev, L. D. (2004). Provability logic. In D. Gabbay, & F. Guenthner (Eds.) *Handbook of Philosophical Logic*, vol. 13, (pp. 229–403). Dordrecht: Springer, 2nd ed.

[23] Artemov, S. N., & Japaridze, G. K. (1990). Finite Kripke models and predicate logics of provability. *Journal of Symbolic Logic*, *55*(3), 1090–1098.

[24] Back, R. (1981). On correct refinement of programs. *Journal of Computer and System Sciences*, *23*(1), 49–68.
URL `https://www.sciencedirect.com/science/article/pii/0022000081900052`

[25] Baier, C., & Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

[26] Barcan, R. C. (1946). A functional calculus of first order based on strict implication. *The Journal of Symbolic Logic*, *11*(1), 1–16.

[27] Barendregt, H., & Barendsen, E. (2002). Autarkic computations in formal proofs. *Journal of Automated Reasoning*, *28*, 321–336.

[28] Basin, D., Matthews, S., & Viganò, L. (1998). Modal logics: Quantifiers. *Journal of Logic, Language and Information*, *7*, 237–263.

[29] Beklemishev, L. D. (2004). Provability algebras and proof-theoretic ordinals, I. *Annals of Pure and Applied Logic*, *128*, 103–124.

[30] Beklemishev, L. D. (2005). Reflection principles and provability algebras in formal arithmetic. *Russian Mathematical Surveys*, *60*(2).

[31] Beklemishev, L. D. (2005). Veblen hierarchy in the context of provability algebras. In P. Hájek, L. Valdés-Villanueva, & D. Westerståhl (Eds.) *Logic, Methodology and Philosophy of Science, Proceedings of the Twelfth International Congress*, (pp. 65–78). Kings College Publications.

[32] Beklemishev, L. D. (2006). The worm principle. In Z. Chatzidakis, P. Koepke, & W. Pohlers (Eds.) *Logic Colloquium 2002, Lecture Notes in Logic 27*, (pp. 75–95). ASL Publications.

[33] Beklemishev, L. D. (2010). Kripke semantics for provability logic GLP. *Annals of Pure and Applied Logic*, *161*, 756–774.

[34] Beklemishev, L. D. (2011). A simplified proof of the arithmetical completeness theorem for the provability logic GLP. *Trudy Matematicheskogo Instituta imeni V.A. Steklova*, *274*(3), 32–40. English translation: *Proceedings of the Steklov Institute of Mathematics*, 274(3), 25–33, 2011.

[35] Beklemishev, L. D. (2012). Calibrating provability logic: From modal logic to Reflection Calculus. In T. Bolander, T. Braüner, T. S. Ghilardi, & L. Moss (Eds.) *Advances in Modal Logic 9*, (pp. 89–94). London: College Publications.

[36] Beklemishev, L. D. (2014). Positive provability logic for uniform reflection principles. *Annals of Pure and Applied Logic*, *165*(1), 82–105.

[37] Beklemishev, L. D. (2015). *Turing's Revolution*, chap. Proof Theoretic Analysis by Iterated Reflection, (pp. 225–270). Birkhäuser, Cham.

[38] Beklemishev, L. D. (2018). A note on strictly positive logics and word rewriting systems. In S. Odintsov (Ed.) *Larisa Maximova on Implication, Interpolation, and Definability*, vol. 15, (pp. 61–70). Berlin, Heidelberg: Springer.

[39] Beklemishev, L. D. (2021). Strictly positive provability logics: Recent progress and open questions.
URL `http://yongcheng.whu.edu.cn/webPageContent/Goedel2021/Goedel-Lev.pdf`

[40] Beklemishev, L. D., Fernández-Duque, D., & Joosten, J. J. (2014). On provability logics with linearly ordered modalities. *Studia Logica*, *102*, 541–566.

[41] Beklemishev, L. D., & Gabelaia, D. (2013). Topological completeness of the provability logic GLP. *Annals of Pure and Applied Logic*, *164*(12), 1201–1223.

[42] Beklemishev, L. D., & Pakhomov, F. N. (2022). Reflection algebras and conservation results for theories of iterated truth. *Annals of Pure and Applied Logic*, *173*(5).
URL `https://www.sciencedirect.com/science/article/pii/S0168007222000082`

[43] Benzmüller, C., & Woltzenlogel Paleo, B. (2015). Interacting with modal logics in the Coq Proof Assistant. In L. D. Beklemishev, & D. V. Musatov (Eds.) *Computer Science – Theory and Applications*, (pp. 398–411). Cham: Springer International Publishing.

[44] Berarducci, A. (1989). Σ^0_n-interpretations of modal logic. *Bollettino dell'Unione Matematica Italiana*, *7*(3-A), 177–184.

[45] Berger, U., Berghofer, S., Letouzey, P., & Schwichtenberg, H. (2006). Program extraction from normalization proofs. *Studia Logica*, *82*, 25–49.

[46] Bernardo, B., Cauderlier, R., Claret, G., Jakobsson, A., Pesin, B., & Tesson, J. (2020). Making Tezos smart contracts more reliable with Coq. In *ISoLA (3)*, vol. 12478 of *Lecture Notes in Computer Science*, (pp. 60–72). Springer.

[47] Besson, F. (2007). Fast reflexive arithmetic tactics the linear case and beyond. In T. Altenkirch, & C. McBride (Eds.) *Types for Proofs and Programs*, (pp. 48–62). Berlin, Heidelberg: Springer Berlin Heidelberg.

[48] Boldo, S., Filliâtre, J.-C., & Melquiond, G. (2009). Combining Coq and Gappa for certifying floating-point programs. In J. Carette, L. Dixon, C. S. Coen, & S. M. Watt (Eds.) *Intelligent Computer Mathematics*, (pp. 59–74). Springer.

[49] Boolos, G. S. (1993). *The logic of provability*. Cambridge: Cambridge University Press.

[50] Boolos, G. S., & McGee, V. R. (1987). The degree of the set of sentences of predicate provability logic that are true under every interpretation. *The Journal of Symbolic Logic*, *52*, 165–171.

[51] Bordg, A., & He, Y. (2019). Comment on "Quantum Games and Quantum Strategies". arXiv:1911.09354 [quant-ph].

[52] Bordg, A., Lachnitt, H., & He, Y. (2021). Certified quantum computation in Isabelle/HOL. *Journal of Automated Reasoning*, *65*, 691–709.

[53] Boutillier, P. (2023). Vector. Coq Standard Library Version 8.17.1.
URL `https://coq.inria.fr/distrib/V8.17.1/stdlib/Coq.Vectors.Vector.html`

[54] Bove, A., Krauss, A., & Sozeau, M. (2016). Partiality and recursion in interactive theorem provers–an overview. *Mathematical Structures in Computer Science*, *26*(1), 38–88.

[55] de Bruijn, N. G. (1972). Lambda Calculus notation with nameless dummies: A tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indagationes Mathematicae*, *34*, 381–392.

[56] Cansell, D., Gibson, J. P., & Méry, D. (2007). Refinement: A constructive approach to formal software design for a secure e-voting interface. *Electronic Notes in Theoretical Computer Science*, *183*, 39–55. Proceedings of the First International Workshop on Formal Methods for Interactive Systems (FMIS 2006).
URL `https://www.sciencedirect.com/science/article/pii/S1571066107004276`

[57] del Castillo Tierz, J. (2018). *When the laws of logic meet the logic of laws*. Master's thesis, University of Barcelona.
URL `http://diposit.ub.edu/dspace/handle/2445/133778`

[58] Celani, S., & Jansana, R. (2012). A note on the model theory for positive modal logic. *Fundamenta Informaticae*, *114*(1), 31–54.

[59] Chen, H., Ziegler, D., Chajed, T., Chlipala, A., Kaashoek, M. F., & Zeldovich, N. (2015). Using Crash Hoare Logic for certifying the FSCQ file system. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, (p. 18–37). New York, NY, USA: Association for Computing Machinery.

[60] Chlipala, A. (2013). *Certified programming with dependent types: A pragmatic introduction to the Coq Proof Assistant*. MIT Press.

[61] Claret, G. (2018). *Program in Coq*. Ph.D. thesis, Université Sorbonne Paris Cité.
URL `https://inria.hal.science/tel-01890983`

[62] Cohen, C., Dénès, M., & Mörtberg, A. (2013). Refinements for free! In G. Gonthier, & M. Norrish (Eds.) *Certified Programs and Proofs*, vol. 8307 of *Lecture Notes in Computer Science*, (pp. 147–162). Cham: Springer.

[63] Cohen, C., & Sakaguchi, K. (2015). Finite maps.
URL `https://github.com/math-comp/finmap`

[64] Cohen, C., Sakaguchi, K., & Tassi, E. (2020). Hierarchy Builder: Algebraic hierarchies made easy in Coq with Elpi. In *FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction*, no. 167 in 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), (pp. 34:1–34:21). Paris, France.

[65] Copello, E., Szasz, N., & Álvaro Tasistro (2018). Machine-checked proof of the Church-Rosser Theorem for the Lambda Calculus using the Barendregt variable convention in constructive type theory. *Electronic Notes in Theoretical Computer Science*, *338*, 79–95. The 12th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2017).
URL `https://www.sciencedirect.com/science/article/pii/S1571066118300720`

[66] Coq Development Team (1989). The Coq Proof Assistant.
URL `http://coq.inria.fr`

[67] Coq Development Team (2023). Functional induction. Coq Reference Manual Version 8.17.1.
URL `https://coq.github.io/doc/v8.17/refman/using/libraries/funind.html#coq:cmd.Function`

[68] Coq Development Team (2023). Preliminary compilation of critical bugs in stable releases of Coq. Version 8.17.1.
URL `https://github.com/coq/coq/blob/v8.17/dev/doc/critical-bugs`

[69] Coq Development Team (2023). Programmable proof search. Coq Reference Manual Version 8.17.1.
URL `https://coq.github.io/doc/v8.17/refman/proofs/automatic-tactics/auto.html`

[70] Coq Development Team (2023). tauto. Coq Reference Manual Version 8.17.1.
URL `https://coq.github.io/doc/v8.17/refman/proofs/automatic-tactics/logic.html#coq:tacn.tauto`

[71] Coquand, T., & Huet, G. (1988). The Calculus of Constructions. *Information and Computation*, *76*(2–3), 95–120.

[72] Coquand, T., & Paulin, C. (1990). Inductively defined types. In P. Martin-Löf, & G. Mints (Eds.) *COLOG-88*, (pp. 50–66). Berlin, Heidelberg: Springer Berlin Heidelberg.

[73] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press, 3rd ed.

[74] cppreference.com Team (2021). `utc_clock`. C++ Reference Manual.
URL `https://en.cppreference.com/mwiki/index.php?title=cpp/chrono/utc_clock&oldid=134878`

[75] Craig, W. (1953). On axiomatizability within a system. *The Journal of Symbolic Logic*, *18*, 30–32.

[76] Cruz-Filipe, L., & Letouzey, P. (2006). A large-scale experiment in executing extracted programs. *Electronic Notes in Theoretical Computer Science*, *151*(1), 75–91. Proceedings of the 12th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus 2005).
URL `https://www.sciencedirect.com/science/article/pii/S1571066106001101`

[77] Czajka, Ł., & Kaliszyk, C. (2018). Hammer for Coq: Automation for dependent type theory. *Journal of Automated Reasoning*, *61*, 423–453.

[78] Dashkov, E. V. (2012). On the positive fragment of the polymodal provability logic GLP. *Mathematical Notes*, *91*(3–4), 318–333.

[79] de Jongh, D., Jumelet, M., & Montagna, F. (1991). On the proof of Solovay's theorem. *Studia Logica*, *50*, 51–69.

[80] Delahaye, D. (2000). A tactic language for the system Coq. In *Logic for Programming and Automated Reasoning: 7th International Conference*, vol. 1955, (pp. 85–95). Springer.

[81] Delaware, B., Pit-Claudel, C., Gross, J., & Chlipala, A. (2015). Fiat: Deductive synthesis of abstract data types in a proof assistant. *ACM SIGPLAN Notices*, *50*(1), 689–700.

[82] Dénès, M., Mörtberg, A., & Siles, V. (2012). A refinement-based approach to computational algebra in Coq. In L. Beringer, & A. Felty (Eds.) *ITP 2012: Interactive Theorem Proving*, vol. 7406 of *Lecture Notes in Computer Science*, (pp. 83–98). Berlin, Heidelberg: Springer.

[83] Doczkal, C., & Bard, J. (2018). Completeness and decidability of converse PDL in the constructive type theory of Coq. In *Certified Programs and Proofs, (CPP 2018)*, (pp. 42–52). Los Angeles, United States.

[84] Doczkal, C., & Smolka, G. (2011). Constructive formalization of hybrid logic with eventualities. In Z. S. Jean-Pierre Jouannaud (Ed.) *Certified Programs and Proofs, (CPP 2011)*, vol. 7086 of *LNCS*, (pp. 5–20). Springer.

[85] Dunn, J. M. (1995). Positive modal logic. *Studia Logica*, *55*, 301–317.

[86] Erbsen, A., Philipoom, J., Gross, J., Sloan, R., & Chlipala, A. (2020). Simple high-level code for cryptographic arithmetic: With proofs, without compromises. *ACM SIGOPS Operating Systems Review*, *54*(1), 23–30.

[87] Errezil Alberdi, G., Joosten, J. J., García Tarrach, G., Solé Sánchez, A., de Almeida Borges, A., Sancho, E., & Fernández-Duque, D. (2019). Industrial software homologation: theory and case study. Available online.
URL `https://formalv.com/Docs//doc-en.pdf`

[88] European Parliament and Council of the European Union (2006). Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending council regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing council regulation (EEC) No 3820/85 (text with EEA relevance) - declaration. Official Journal of the European Union.
URL `https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX%3A32006R0561`

[89] European Parliament and Council of the European Union (2009). Commission Regulation (EU) No 1266/2009 of 16 December 2009 adapting for the tenth time to technical progress Council Regulation (EEC) No 3821/85 on recording equipment in road transport. Official Journal of the European Union.
URL `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32009R1266&qid=1688983273421`

[90] European Parliament and Council of the European Union (2016). Commission implementing Regulation (EU) 2016/799 of 18 March 2016 implementing Regulation (EU) No 165/2014 of the European Parliament and of the Council laying down the requirements for the construction, testing, installation, operation and repair of tachographs and their components. Official Journal of the European Union.
URL `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0799&qid=1688983367069`

[91] Fagin, R., Halpern, J. Y., Moses, Y., & Vardi, M. Y. (1995). *Reasoning About Knowledge*. Cambridge, MA: MIT Press.

[92] Feferman, S. (1960). Arithmetization of metamathematics in a general setting. *Fundamenta Mathematicae*, *49*, 35–92.

[93] Fernández-Duque, D. (2017). Worms and spiders: Reflection calculi and ordinal notation systems. *IfCoLoG Journal of Logics and their Applications*, *4*(10), 3277–3356.

[94] Fernández-Duque, D., & Hermo Reyes, E. (2019). A self-contained provability calculus for Γ_0. In R. Iemhoff, M. Moortgat, & R. J. G. B. de Queiroz (Eds.) *Logic, Language, Information, and Computation - 26th International Workshop, WoLLIC 2019, Utrecht, The Netherlands, July 2-5, 2019, Proceedings*, vol. 11541 of *Lecture Notes in Computer Science*, (pp. 195–207). Springer.

[95] Fernández-Duque, D., & Joosten, J. J. (2013). Hyperations, Veblen progressions and transfinite iteration of ordinal functions. *Annals of Pure and Applied Logic*, *164*(7–8), 785–801.

[96] Fernández-Duque, D., & Joosten, J. J. (2013). Models of transfinite provability logic. *The Journal of Symbolic Logic*, *78*(2), 543–561.

[97] Fernández-Duque, D., & Joosten, J. J. (2014). Well-orders in the transfinite Japaridze algebra. *Logic Journal of the IGPL*, *22*(6), 933–963.

[98] Fernández-Duque, D., & Joosten, J. J. (2018). The omega-rule interpretation of transfinite provability logic. *Annals of Pure and Applied Logic*, *169*(4), 333–371.

[99] Fernández-Duque, D., González Bedmar, M., Sousa, D., Joosten, J. J., & Errezil Alberdi, G. (2019). To drive or not to drive: A formal analysis of Requirements (51) and (52) from Regulation (EU) 2016/799. In *Personalidades jurídicas difusas y artificiales*, vol. 4 of *TransJus Working Papers Publication*, (pp. 159–171). Institut de Recerca TransJus.

[100] Formal Vindications S.L. (2022). The FV Time Manager.
URL `https://formalv.com/TimeManager/FVTimeCalculation`

[101] Formal Vindications S.L. (2022). How-to: Use of the FV Time Manager on Windows, Linux and other platforms through its command line interface.
URL `https://formalv.gitlab.io/fv-docs/fvtm-tech-spec.pdf`

[102] Forster, Y., Kirst, D., & Wehr, D. (2021). Completeness theorems for first-order logic analysed in constructive type theory: extended version. *Journal of Logic and Computation*, *31*, 112–151.

[103] Forster, Y., Larchey-Wendling, D., Dudenhefner, A., Heiter, E., Hermes, M., Kirst, D., Koch, M., Kunze, F., Smolka, G., Spies, S., Wehr, D., & Wuttke, M. (2018). Coq library of undecidability proofs.
URL `https://github.com/uds-psl/coq-library-undecidability`

[104] Fujiwara, M., & Kurahashi, T. (2022). Refining the arithmetical hierarchy of classical principles. *Mathematical Logic Quarterly*, *68*(3), 318–345.

[105] Gabbay, D., Pnueli, A., Shelah, S., & Stavi, J. (1980). On the temporal analysis of fairness. In *7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, (pp. 163–173).

[106] Gabbay, D. M., Kurucz, A., Wolter, F., & Zakharyaschev, M. (2003). *Many-dimensional modal logics: Theory and applications*. Elsevier.

[107] Gabbay, D. M., Shehtman, V. B., & Skvortsov, D. P. (2009). *Quantification in Nonclassical Logic, Volume 1*. Amsterdam: Elsevier.

[108] Geuvers, H. (2009). Proof assistants: History, ideas and future. *Sādhanā*, *34*, 3–25.

[109] Gleißner, T., Steen, A., & Benzmüller, C. (2017). Theorem provers for every normal modal logic. In T. Eiter, & D. Sands (Eds.) *LPAR-21: 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, vol. 46 of *EPiC Series in Computing*, (pp. 14–30).

[110] Goldblatt, R. (2011). *Quantifiers, propositions and identity, admissible semantics for quantified modal and substructural logics*. Cambridge University Press.

[111] Gonthier, G. (2008). Formal proof – the four-color theorem. *Notices of the American Mathematical Society*, *55*(11), 1382–1393.

[112] Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Le Roux, S., Mahboubi, A., O'Connor, R., Ould Biha, S., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., & Théry, L. (2013). A machine-checked proof of the odd order theorem. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.) *Interactive Theorem Proving*, (pp. 163–179). Berlin, Heidelberg: Springer Berlin Heidelberg.

[113] Gonthier, G., Mahboubi, A., & Tassi, E. (2016). A small scale reflection extension for the Coq system. Research Report RR-6455, Inria Saclay Ile de France.
URL `https://hal.inria.fr/inria-00258384`

[114] Google (2022). Unsmear.
URL `https://github.com/google/unsmear`

[115] Gouëzel, S., & Shchur, V. (2019). A corrected quantitative version of the Morse lemma. *Journal of Functional Analysis*, *277*(4), 1258–1268.
URL `https://www.sciencedirect.com/science/article/pii/S0022123619300801`

[116] Governatori, G. (2015). Thou shalt is not you will. In *Proceedings of the 15th International Conference on Artificial Intelligence and Law, ICAIL 2015, San Diego, CA, USA, June 8-12, 2015*, (pp. 63–68).

[117] Grégoire, B., & Leroy, X. (2002). A compiled implementation of strong reduction. In *Proceedings of the seventh ACM SIGPLAN international conference on Functional programming*, (pp. 235–246).

[118] Grégoire, B., & Mahboubi, A. (2005). Proving equalities in a commutative ring done right in Coq. In J. Hurd, & T. Melham (Eds.) *Theorem Proving in Higher Order Logics*, (pp. 98–113). Berlin, Heidelberg: Springer Berlin Heidelberg.

[119] Guretruck S.L. (2023). Police controller.
URL `https://www.guretruck.com/PController/en`

[120] Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, *38*, 173–198.

[121] Haftmann, F., Krauss, A., Kunčar, O., & Nipkow, T. (2013). Data refinement in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, & D. Pichardie (Eds.) *Interactive Theorem Proving*, (pp. 100–115). Berlin, Heidelberg: Springer Berlin Heidelberg.

[122] Hájek, P., & Pudlák, P. (1993). *Metamathematics of First Order Arithmetic*. Berlin, Heidelberg, New York: Springer-Verlag.

[123] Hales, T., Adams, M., Bauer, G., Dang, T. D., Harrison, J., Hoang, L. T., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T. T., & et al. (2017). A formal proof of the Kepler conjecture. *Forum of Mathematics, Pi*, *5*, 1–29.

[124] Hao, Y., & Tourlakis, G. (2021). An arithmetically complete predicate modal logic. *Bulletin of the Section of Logic*.
URL `https://czasopisma.uni.lodz.pl/bulletin/article/view/8441`

[125] Herrestad, H. (1991). Norms and formalization. In *Proceedings of the Third International Conference on Artificial Intelligence and Law, ICAIL '91*, (pp. 175–184).

[126] Hinnant, H. (2019). chrono-compatible low-level date algorithms.
URL `https://howardhinnant.github.io/date_algorithms.html`

[127] HOL Light Development Team (1996). HOL Light.
URL `https://www.cl.cam.ac.uk/~jrh13/hol-light/`

[128] Holzmann, G. (2011). *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1st ed.

[129] Hughes, G. E., & Cresswell, M. J. (1996). *A New Introduction to Modal Logic*. Routledge.

[130] Huttner, L., & Merigoux, D. (2022). Catala: Moving towards the future of legal expert systems. *Artificial Intelligence and Law*.
URL `https://link.springer.com/article/10.1007/s10506-022-09328-5`

[131] Icard, T. (2008). *Models of the Polymodal Provability Logic*. Master's thesis, Universiteit van Amsterdam.

[132] IEEE and The Open Group (2018). The Open Group Base Specifications Issue 7.
URL `https://pubs.opengroup.org/onlinepubs/9699919799/`

[133] Iemhoff, R. (2003). Preservativity logic: An analogue of interpretability logic for constructive theories. *Mathematical Logic Quarterly*, *49*(3), 230–249.

[134] IETF (2016). Leap seconds list.
URL `https://www.ietf.org/timezones/data/leap-seconds.list`

[135] Ignatiev, K. N. (1992). *The closed fragment of Dzhaparidze's polymodal logic and the logic of* Σ_1*-conservativity*. ITLI Prepublication Series X–92–02, University of Amsterdam.

[136] Ignatiev, K. N. (1993). On strong provability predicates and the associated modal logics. *The Journal of Symbolic Logic*, *58*, 249–290.

[137] Isabelle Development Team (1994). Isabelle.
URL `https://isabelle.in.tum.de/`

[138] ITU Radiocommunication Assembly (2002). Recommendation ITU-R TF.460-6: Standard-frequency and time-signal emissions. *International Telecommunication Union*.
URL `https://www.itu.int/dms_pubrec/itu-r/rec/tf/R-REC-TF.460-6-200202-I!!PDF-E.pdf`

[139] Japaridze, G. K. (1988). The polymodal provability logic. In *Intensional logics and logical structure of theories: material from the fourth Soviet-Finnish symposium on logic, Telavi, May 20–24, 1985*, (pp. 16–48). Metsniereba. In Russian.

[140] Joosten, J. J. (2016). Turing-Taylor expansions for arithmetic theories. *Studia Logica*, *104*(6), 1225–1243.

[141] Joosten, J. J. (2021). Münchhausen provability. *The Journal of Symbolic Logic*, *86*(3), 1006–1034.

[142] Kamp, H. (1968). *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, UCLA.

[143] Kikot, S., Kurucz, A., Tanaka, Y., Wolter, F., & Zakharyaschev, M. (2019). Kripke completeness of strictly positive modal logics over meet-semilattices with operators. *The Journal of Symbolic Logic*, *84*(2), 533–588.

[144] Kikot, S., Kurucz, A., Wolter, F., & Zakharyaschev, M. (2018). On strictly positive modal logics with S4.3 frames. In G. Bezhanishvili, G. D'Agostino, G. Metcalfe, & T. Studer (Eds.) *Advances in Modal Logic 12*, (pp. 427–446). College Publications.

[145] Kripke, S. A. (1963). Semantical considerations on modal logic. *Acta Philosophica Fennica*, *16*, 83–94.

[146] Kurahashi, T. (2013). On predicate provability logics and binumerations of fragments of Peano Arithmetic. *Archive for Mathematical Logic*, *52*, 871–880.

[147] Kurahashi, T. (2022). On inclusions between quantified provability logics. *Studia Logica*, *110*, 165–188.

[148] Kurtonina, N., & de Rijke, M. (1997). Bisimulations for temporal logic. *Journal of Logic, Language and Information*, *6*(4), 403–425.

[149] Kurucz, A., Wolter, F., & Zakharyaschev, M. (2010). Islands of tractability for relational constraints: Towards dichotomy results for the description logic EL. In L. D. Beklemishev, V. Goranko, & V. Shehtman (Eds.) *Advances in Modal Logic 8*, (pp. 271–291). College Publications.

[150] Ladner, R. E. (1977). The computational complexity of provability in systems of modal propositional logic. *SIAM journal on computing*, *6*(3), 467–480.

[151] Lean Development Team (2015). Lean theorem prover.
URL `https://leanprover.github.io/`

[152] Leijen, D. (2016). UTC time calculation. The Koka Programming Language Documentation.
URL `https://koka-lang.github.io/koka/doc/std_time_utc.html`

[153] Leivant, D. (1983). The optimality of induction as an axiomatization of arithmetic. *The Journal of Symbolic Logic*, *48*, 182–184.

[154] Leroy, X. (2009). Formal verification of a realistic compiler. *Commun. ACM*, *52*(7), 107–115.

[155] Letouzey, P. (2003). A new extraction for Coq. In H. Geuvers, & F. Wiedijk (Eds.) *Types for Proofs and Programs*, (pp. 200–219). Berlin, Heidelberg: Springer.

[156] Letouzey, P. (2004). *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*. Ph.D. thesis, Université Paris Sud-Paris XI.

[157] Letouzey, P. (2008). Extraction in Coq: An overview. In *Conference on Computability in Europe*, (pp. 359–369). Springer.

[158] Libal, T. (2018). A simple semi-automated proof assistant for first-order modal logics. In C. Benzmüller, & J. Otten (Eds.) *Proceedings of the 3rd International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL 2018)*, (pp. 34–48).

[159] Lichtenstein, O., & Pnueli, A. (1985). Checking that finite state concurrent programs satisfy their linear specification. In *Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, (pp. 97–107).

[160] Litak, T., & Wolter, F. (2005). All finitely axiomatizable tense logics of linear time flows are coNP-complete. *Studia Logica*, *81*, 153–165.

[161] Löb, M. H. (1955). Solution of a problem of Leon Henkin. *The Journal of Symbolic Logic*, *20*, 115–118.

[162] Löwenheim, L. (1915). Über möglichkeiten im relativkalkül. *Mathematische Annalen*, *76*(4), 447–470.

[163] Maggesi, M., & Brogi, C. P. (2022). A theorem prover and countermodel constructor for provability logic in HOL Light. arXiv:2205.03659 [cs.LO].

[164] Mahboubi, A. (2013). The rooster and the butterflies. In *International Conference on Intelligent Computer Mathematics*, (pp. 1–18). Springer.

[165] Mahboubi, A., & Tassi, E. (2022). *Mathematical Components*. Zenodo. Version 1.0.2.
URL `https://doi.org/10.5281/zenodo.7118596`

[166] Mathematical Components Team (2007). The Mathematical Components library.
URL https://math-comp.github.io/

[167] Mathematical Components Team (2023). Choice. Version 1.17.0.
URL https://math-comp.github.io/htmldoc_1_17_0/mathcomp.ssreflect.choice.html

[168] Mathematical Components Team (2023). Tuple. Version 1.17.0.
URL https://math-comp.github.io/htmldoc_1_17_0/mathcomp.ssreflect.tuple.html

[169] MathWorks (2022). leapseconds. MATLAB Reference Manual.
URL https://www.mathworks.com/help/matlab/ref/leapseconds.html

[170] McGee, V. R. (1985). *Truth and Necessity in Partially Interpreted Languages*. Ph.D. thesis, University of California, Berkeley.

[171] Melquiond, G. (2023). Coq interval.
URL https://coqinterval.gitlabpages.inria.fr/

[172] Merigoux, D., Chataing, N., & Protzenko, J. (2021). Catala: A programming language for the law. *Proceedings of the ACM on Programming Languages*, *5*(ICFP), 77:1–29.

[173] Microsoft (2022). DateTime.Ticks. Microsoft .NET 6 Documentation.
URL https://docs.microsoft.com/en-us/dotnet/api/system.datetime.ticks?view=net-6.0

[174] Mojtahedi, M. (2022). On provability logic of HA. arXiv:2206.00445 [math.LO].

[175] Montagna, F. (1984). The predicate modal logic of provability. *Notre Dame Journal of Formal Logic*, *25*(2), 179–189.

[176] Mullen, E., Pernsteiner, S., Wilcox, J. R., Tatlock, Z., & Grossman, D. (2018). Œuf: Minimizing the Coq extraction TCB. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2018, (pp. 172–185). New York, NY, USA: Association for Computing Machinery.

[177] Nederpelt, R., & Geuvers, H. (2014). *Type Theory and Formal Proof: An Introduction*. New York: Cambridge University Press.

[178] Ouaknine, J., & Worrell, J. (2007). On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, *3*(1).

[179] O'Connor, R. (2005). Essential incompleteness of arithmetic verified by Coq. In J. Hurd, & T. Melham (Eds.) *Proceedings of the 18th international conference on Theorem Proving in Higher Order Logics*, vol. 3603 of *Theoretical Computer Science and General Issues*, (pp. 245–260). Berlin, Heidelberg: Springer-Verlag.

[180] Pakhomov, F. (2014). On the complexity of the closed fragment of Japaridze's provability logic. *Archive for Mathematical Logic*, *53*(7–8), 949–967.

[181] Pédrot, P.-M. (2019). Ltac2: tactical warfare. In *The Fifth International Workshop on Coq for Programming Languages, CoqPL*, (pp. 13–19).

[182] Pnueli, A. (1977). The temporal logic of programs. In *18th IEEE Symposium on the Foundations of Computer Science*, (pp. 46–57).

[183] Pohlers, W. (2009). *Proof Theory, The First Step into Impredicativity*. Berlin Heidelberg: Springer-Verlag.

[184] Post, E. L. (1944). Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, *50*, 284–316.

[185] Pozsgay, L. J. (1968). Gödel's second theorem for Elementary Arithmetic. *Mathematical Logic Quarterly*, *14*, 67–80.

[186] Rabinovich, A. (2014). A proof of Kamp's theorem. *Logical Methods in Computer Science*, *10*.

[187] Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., & Tatlock, Z. (2019). QED at large: A survey of engineering of formally verified software. *Foundations and Trends in Programming Languages*, *5*(2–3), 102–281.

[188] Sakaguchi, K. (2020). Validating mathematical structures. In N. Peltier, & V. Sofronie-Stokkermans (Eds.) *Automated Reasoning*, (pp. 138–157). Cham: Springer International Publishing.

[189] Sakaguchi, K. (2021). Mczify.
URL `https://github.com/math-comp/mczify`

[190] Sernadas, A., & Sernadas, C. (2012). *Foundations of Logic and Theory of Computation*, vol. 10 of *Texts in Computing*. London: College Publications, second ed.

[191] Shapirovsky, I. (2008). PSPACE-decidability of Japaridze's polymodal logic. In C. Areces, & R. Goldblatt (Eds.) *Advances in Modal Logic 7*, (pp. 289–304). College Publications.

[192] Sistla, A. P., & Clarke, E. M. (1985). The complexity of propositional linear temporal logics. *Journal of the ACM (JACM)*, *32*(3), 733–749.

[193] Smoryński, C. (1985). *Self-Reference and Modal Logic*. Springer.

[194] Solovay, R. M. (1976). Provability interpretations of modal logic. *Israel Journal of Mathematics*, *28*, 33–71.

[195] Sozeau, M. (2007). Subset coercions in Coq. In *Types for Proofs and Programs: International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, (pp. 237–252). Springer.

[196] Sozeau, M., Forster, Y., Lennon-Bertrand, M., Nielsen, J. B., Tabareau, N., & Winterhalter, T. (2023). Correct and complete type checking and certified erasure for Coq, in Coq. Hal-04077552.
URL `https://inria.hal.science/hal-04077552`

[197] Sozeau, M., & Mangin, C. (2019). Equations reloaded: High-level dependently-typed functional programming and proving in Coq. *Proceedings of the ACM on Programming Languages*, *3*(ICFP), 1–29.

[198] Stark, K., Schäfer, S., & Kaiser, J. (2019). Autosubst 2: Reasoning with multi-sorted de Bruijn terms and vector substitutions. *8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*.

[199] Svyatlovskiy, M. V. (2018). Axiomatization and polynomial solvability of strictly positive fragments of certain modal logics. *Mathematical Notes*, *103*, 952–967.

[200] Tarski, A. (1983). *Logic, Semantics and Metamathematics*, chap. The concept of truth in formalized languages, (pp. 152–278). Hackett. English translation of Tarski's 1936 Der Wahrheitsbegriff in den Formalisierten Sprachen.

[201] Tassi, E. (2019). Deriving proved equality tests in Coq-elpi: Stronger induction principles for containers in Coq. In *ITP 2019 - 10th International Conference on Interactive Theorem Proving*. Portland, United States.
URL `https://inria.hal.science/hal-01897468`

[202] Troelstra, A. S., & van Dalen, D. (1988). *Constructivism in Mathematics: An Introduction*, vol. I. Elsevier.

[203] Vardanyan, V. A. (1986). Arithmetic complexity of predicate logics of provability and their fragments. *Doklady Akad. Nauk SSSR*, *288*(1), 11–14. In Russian. English translation in Soviet Mathematics Doklady 33, 569–572 (1986).

[204] Vardanyan, V. A. (1988). Bounds on the arithmetical complexity of predicate logics of provability. In S. N. Adyan (Ed.) *Questions of Cybernetics: Complexity of Computation and Applied Mathematical Logic*, vol. 134 of *Scientific Council of the USSR Academy of Sciences on the complex problem "Cybernetics"*, (pp. 46–72). Academy of Sciences of the USSR. In Russian.

[205] Vardi, M. Y. (1997). Why is modal logic so robustly decidable? Tech. rep.
URL `https://hdl.handle.net/1911/96465`

[206] Vardi, M. Y. (2011). The rise and fall of LTL. *GandALF*, *54*.
URL `https://www.cs.rice.edu/~vardi/papers/gandalf11-myv.pdf`

[207] Visser, A., & de Jonge, M. (2006). No escape from Vardanyan's theorem. *Archive for Mathematical Logic*, *45*(5), 539–554.

[208] Visser, A., & Zoethout, J. (2019). Provability logic and the completeness principle. *Annals of Pure and Applied Logic*, *170*(6), 718–753.

[209] Wiedijk, F. (2000). The de Bruijn Factor.
URL `https://www.cs.ru.nl/~freek/factor/factor.pdf`

[210] Wiedijk, F. (2006). *The Seventeen Provers of the World*. Springer Berlin, Heidelberg.

[211] Wiedijk, F. (2012). The "de Bruijn factor".
URL `https://www.cs.ru.nl/~freek/factor/`

[212] Yavorsky, R. E. (2002). On arithmetical completeness of first-order logics of provability. In F. Wolter, H. Wansing, M. de Rijke, & M. Zakharyaschev (Eds.) *Advances in Modal Logic 3*, (pp. 1–16). World Scientific Publishing Co. Pte. Ltd.

[213] Ziliani, B., Dreyer, D., Krishnaswami, N. R., Nanevski, A., & Vafeiadis, V. (2013). Mtac: a monad for typed tactic programming in Coq. *ACM SIGPLAN Notices*, *48*(9), 87–100.

List of figures

List of tables