TU WIEN Informatics

SMT-Based Automated Reasoning for Åqvist's Deontic Logics

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Christian Köll, BSc
Matrikelnummer 11727636

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof.in Dr.in Agata Ciabattoni
Zweitbetreuung: Univ.Ass. Mag. Dmitry Rozplokhas

Wien, 3. September 2025

Christian Köll	Agata Ciabattoni

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

SMT-Based Automated Reasoning for Åqvist's Deontic Logics

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Christian Köll, BSc
Registration Number 11727636

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof.in Dr.in Agata Ciabattoni
Second advisor: Univ.Ass. Mag. Dmitry Rozplokhas

Vienna, September 3, 2025

Christian Köll	Agata Ciabattoni

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Christian Köll, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, habe ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT-Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 3. September 2025

Christian Köll

Danksagung

Ich möchte Agata Ciabattoni und Dmitry Rozplokhas für ihre hervorragende Betreuung, wertvolle Beratung und hilfreichen Anregungen während der Anfertigung dieser Arbeit danken. Ihre Expertise und Unterstützung haben wesentlich zur Entwicklung und Präzisierung dieser Arbeit beigetragen. Insbesondere möchte ich Dmitry Rozplokhas für seine grundlegende Arbeit danken, auf der diese Arbeit basiert.

Mein besonderer Dank gilt meinen Eltern, die mir das Studium ermöglicht und mich auf meinem akademischen Weg unterstützt haben.

Auch meiner Freundin danke ich ausdrücklich für ihre verlässliche Unterstützung.

Acknowledgements

I would like to thank Agata Ciabattoni and Dmitry Rozplokhas for their excellent supervision, valuable guidance, and insightful feedback throughout the course of this thesis. Their expertise and support greatly contributed to the development and refinement of this work. In particular, I am thankful to Dmitry Rozplokhas for his foundational work on which this thesis builds.

My special gratitude goes to my parents, who made my studies possible and supported me throughout my academic journey.

I am also deeply thankful to my girlfriend for her reliable support.

Kurzfassung

Die Logiken von Åqvist bilden eine wichtige und vielfach untersuchte Familie innerhalb der deontischen Logik. Rozplokhas [Roz24] entwickelte Small-Model-Konstruktionen, die beliebige Gegenmodelle in äquivalente Modelle mit polynomial beschränkter Größe transformieren. Diese Konstruktionen zeigen, dass das Theoremerkennungsproblem für alle vier Åqvist-Logiken **E**, **F**, **F+(CM)** und **G** ko-NP-vollständig ist, und ermöglichen aussagenlogische Kodierungen für effizientes automatisiertes Schließen.

In meiner Arbeit kodiere ich Rozplokhas' Konstruktionen mithilfe des SMT Solvers Z3, um die Gültigkeit von Formeln zu überprüfen und minimale Gegenmodelle für nicht gültige Formeln zu finden. Wir präsentieren den gesamten Ablauf, vom Einlesen der Eingabeformeln, über die Z3-Kodierungen bis hin zur Darstellung der Gegenmodelle.

Das entwickelte Tool bietet drei mögliche Darstellungsarten für Gegenmodelle: in reinen Text, als Matrix oder als gerichteten Graphen. Alle Informationen können in einem gerichteten Graphen mit geraden, kreuzungsfreien Kanten für Modelle mit bis zu drei Welten dargestellt werden. Bei größeren Modellen wird die Präferenzrelation als Matrix und in reinem Textformat dargestellt.

Als zusätzliche Optimierung führen wir Vereinfachungsregeln und -prozeduren ein, die Eingabeformeln so weit wie möglich reduzieren, was zu kleineren Kodierungen (um 55%) und einer Reduzierung der Formelgröße um 77% führt und die Lösungszeiten um das 1,71-Fache beschleunigt. Als Fallstudie haben wir das Tool auf bekannte deontische Paradoxien angewendet, um zu analysieren, ob diese in Åqvists Logiken blockiert werden. Abschließend demonstrieren wir die Effizienz des Tools durch Laufzeitmessungen an einer generierten Menge von Testformeln und vergleichen die Ergebnisse mit dem bestehenden automatisierten Ansatz von Parent und Benzmüller, der auf Isabelle/HOL basiert. Im Vergleich zu deren Ansatz erzielt unser Tool bei ungültigen Formeln eine bis zu 25-fache Beschleunigung und unterstützt zudem alle Åqvist-Logiken, während Isabelle/HOL lediglich **E** abdeckt. Durch die intuitive Syntax und die leicht verständliche Darstellung der Gegenmodelle ist es auch für Nicht-Experten zugänglich und stellt somit eine leichte und effiziente Alternative für zukünftige Anwendungen im Bereich des automatisierten deontischen Schließens dar.

Abstract

Åqvist's logics are an important and widely studied family within the field of deontic logics. Rozplokhas [Roz24] developed small model constructions for them, which transform arbitrary countermodels into equivalent models of bounded polynomial size. These constructions establish the co-NP-completeness of the theoremhood problem for all four Åqvist's logics: **E**, **F**, **F+(CM)**, and **G**, and provide propositional encodings that enable efficient automated reasoning.

In my thesis, I encoded Rozplokhas' constructions with the SMT solver Z3 to check the validity of formulas and to find minimal countermodels for invalid formulas. We present the entire workflow in detail: from parsing the input formulas, through the Z3-based encodings, to the structured representation of the countermodels.

The tool provides three alternative ways of presenting countermodels: as plain text, as a matrix, or as a directed graph. All information can be displayed in a directed graph with straight, crossing-free edges for models with up to three worlds. For larger models, the preference relation is represented as a matrix and in plain-text form.

As an additional optimisation, we introduce simplification rules and procedures that reduce input formulas as much as possible, resulting in smaller encodings (by 55%) and a 77% reduction in formula size, which together lead to a speedup of 1.71 in solving times.

As a case study, we applied the tool to analyse some well-known deontic paradoxes and check whether they are blocked in Åqvist's logics. Finally, we demonstrate the efficiency of the tool by performing runtime measurements on a generated test set of formulas and compare the results with an existing automated approach by Parent and Benzmüller based on Isabelle/HOL. Compared to their approach, our tool achieves up to a 25-fold speedup on invalid formulas, and supports all Åqvist's logics rather than just the logic **E**. Its intuitive syntax and human-readable countermodel representations make our tool accessible to non-experts, positioning it as a lightweight and efficient alternative for future applications in automated deontic reasoning.

Contents

CHAPTER 1

Introduction

Reasoning about norms (normative reasoning) plays a central role in various fields. Deontic logic provides a formal framework to represent and reason about normative concepts such as obligation, permission, and prohibition. It has been applied for many years across disciplines [WM93] and remains highly relevant today, particularly in artificial intelligence [CHS+23] (especially for autonomous agents and systems [NBCG22, Sin22, NBCG21, SBA20]), law [GP25, CPS21], and philosophy [CDF25, vBCF+23, FP21].

Over the past decades, a wide variety of deontic logics have been proposed to capture the nuances of normative reasoning [PvDT13, GHP+21]. The first deontic logic introduced in the literature is Standard Deontic Logic (SDL) [VW51]. Although influential, expressive limitations of the original framework showed in the so-called deontic paradoxes led to the development of more refined frameworks. These include dyadic deontic logics, which condition obligations on contextual information (e.g., "ψ ought to hold given ϕ") and allow for greater flexibility in representing normative scenarios.

These logics can be broadly categorised into norm-based systems, which represent obligations through an explicit set of normative statements or rules, and preference-based systems, which interpret deontic modalities over possible worlds ordered by a preference relation.

Preference-based systems are particularly well suited for modelling two prominent types of conditional obligations:
Conditionals contrary-to-duty, such as those discussed by Chisholm [Chi63], which are triggered when a primary obligation is violated, and defeasible deontic conditionals, which capture conditional obligations that can be overridden by more specific or contextually stronger norms. These conditionals reflect the non-monotonic character of normative reasoning, allowing for exceptions and priority among rules.

One of the earliest formalisations of preference-based dyadic deontic logic was proposed by Hansson [Han71], who introduced the logics DSDL1, DSDL2, and DSDL3. These

systems interpret conditional obligations in terms of preference structures over possible worlds, laying important groundwork for later developments in this tradition.

A fundamental basis for preference-based semantics is the work of Lewis [Lew73a], as presented in [Par21]. Lewis introduced the limit assumption, which states that if a formula holds in some world, then there is a best world satisfying it. This assumption plays a central role in subsequent developments of preference-based semantics.

An important family of preference-based deontic logics are the logics formulated by Lennart Åqvist [Åqv84], which extend the modal logic **S5** [Lew18] with suitable deontic operators. While **S5** features a unary necessity operator $\Box\phi$, which is true if ϕ holds in all possible worlds, Åqvist's system augments this framework with a conditional obligation $\bigcirc(\psi \mid \phi)$ (to be read as "ψ is obligated, given ϕ"). This expression is true if ψ holds in all the most preferred ("best") worlds among the world where ϕ holds. The conditional obligation operator can be used to express prohibitions ($\bigcirc(\neg\psi \mid \phi)$) and permissions ($\neg\bigcirc(\neg\psi \mid \phi)$).

Åqvist [Åqv84] originally proposed three related logics **E**, **F**, and **G**. These logics are increasingly expressive and restrictive on the preference relations.

- logic **E** imposes no structural conditions on the preference relation.
- logic **F** adds constraints by assuming a limited preference relation.
- logic **G** further strengthens the structure by assuming that the preference relation is total, meaning that any two worlds are comparable, in addition to being limited.

Building on these foundations, Parent [Par14] introduced the logic **F+(CM)**, which extends logic **F** by incorporating the principle of Cautious Monotony (CM), well known from the field of non-monotonic reasoning [KLM90]. To validate this principle, Parent omits the totality assumption of **G** and instead assumes a smooth preference relation. Smoothness means that for every condition that holds in some world, there is at least one best world where it also holds.

In a related development, the use of preference structures was explored in the domain of conditional reasoning. Lewis's family of logics for counterfactuals, such as the system **VTA** [Lew73b], provide a semantics based on similarity or preference between possible worlds. Notably, **VTA** corresponds to the deontic logic **G**.

Burgess introduces an alternative but related framework with his preferential conditional logic (**PCL**) [Bur81, NO15] and its extensions. By adding an absoluteness axiom to **PCL**, one obtains the system **PCA**, which aligns with the deontic logic **F+(CM)**.

A central task in logic is to derive formulas from premises and construct counterexamples when this is not the case. However, traditional Hilbert-style systems are inadequate for these tasks. To address these limitations, sequent calculi have been developed, offering a more structured approach to deduction. Originating in Gentzen's work [Gen35], these systems enable the elimination of the cut rule, guaranteeing that every derivable sequent

has a cut-free proof. This is a property desirable both for proof analysis and for proving the decidability of the logic.

For logics with conditional obligations cut-free calculi based on hypersequents have proven especially effective. Hypersequents extend traditional sequents with a structural mechanism to reason about sequents in parallel. Recent work by Ciabattoni et al. [COP22, COP$^+$23, CT24] presents analytic calculi for such logics, tailored to ensure syntactic properties such as cut-elimination and completeness with respect to preference-based semantics.

An alternative method is to embed these conditional systems into higher-order logic, as proposed by Parent and Benzmüller [PB24]. This embedding leverages the expressive power of higher-order frameworks to represent preference relations and conditional modalities within an uniform semantic setting.

A further approach, based on the logic's semantics, involves small model constructions, as introduced by Friedman and Halpern [FH94]. Their method provides a transformation of any model satisfying a formula into a model of bounded size that also satisfies it. This construction applies to Burgess's logic **PCL** and its extensions, in particular to **PCA** (corresponding to **F+(CM)**) and **VTA** (corresponding to **G**). However, since the construction relies on smoothness and transitivity of the preference relation, it is not applicable to weaker logics such as **E** and **F**.

Recently, Rozblokhas developed alternative small model constructions [Roz24], combining blocks of worlds. The constructions are applicable across all Åqvist's logics and yield models of polynomial size. Beyond that, he presented an alternative semantical characterisations using the natural frame properties acyclicity and transitivity. From this, he derived the co-NP-completeness of the theoremhood problem, resolving a previously open question for logic **F**. Building on this formalisation, Rozblokhas also provided an encoding of formulas into classical propositional logic (see Section 2.4).

Using these encodings, we developed an efficient tool for automated reasoning by leveraging Satisfiability Modulo Theories (SMT) solving technology. The tool takes a string as an input and uses the SMT-solver Z3 of Microsoft [DMB08] to determine the validity of the formula and obtain a countermodel if it is not valid.

The stepwise procedure is depicted in Figure 1.1. The application parses the input text to a formula. If an error occurs during parsing, it gives feedback stating what is wrong with the input and indicates the position of the error. Details of the parsing process are provided in Chapter 3. After parsing, the formula is simplified as much as possible (see Chapter 4) before being encoded into classic propositional logic for the SMT-solver Z3 (see Chapter 5). The encoding depends on the selected one of Åqvist's logics. If Z3 fails to find a model, the formula is considered valid. Otherwise, a countermodel of the original formula is extracted from the output of Z3. The tool will represent these models in a readable way, where options for the representation are compared in Chapter 6.

To evaluate the reasoning capabilities of the tool, we implement three well-known

paradoxes from deontic logic: Chisholm's paradox, the Gentle Murder paradox, and the principle of Deontic Explosion. We test these cases across the four Åqvist logics. The results demonstrate how the preference-based semantics affect the validation or blocking of each paradox.

We further compare our tool with the Isabelle/HOL-based framework by Parent and Benzmüller. While their approach offers high assurance, it requires familiarity with Isabelle syntax and lacks human-readable countermodel representations. Our tool provides user-friendly input, multiple visualization formats for models, and significantly faster runtimes in several cases. Especially for invalid formulas, our tool achieves speedups of over 25× compared to the Isabelle-based approach.

1.1 Thesis Structure

This thesis is structured as follows: We begin by introducing of the theoretical foundations of Åqvist's logics and small model constructions in Chapter 2. Chapters 3 to 5 cover the implementation of parsing, simplification, and encoding for the Z3 solver. Various approaches to visualizing countermodels are presented in Chapter 6. The handling of classic deontic paradoxes is discussed in Chapter 7. In Chapter 8 the performance of different optimizations is analysed. We compare our tool with an existing approach by Parent and Benzmüller in Chapter 9. Finally, Chapter 10 summarizes the results and outlines directions for future work.

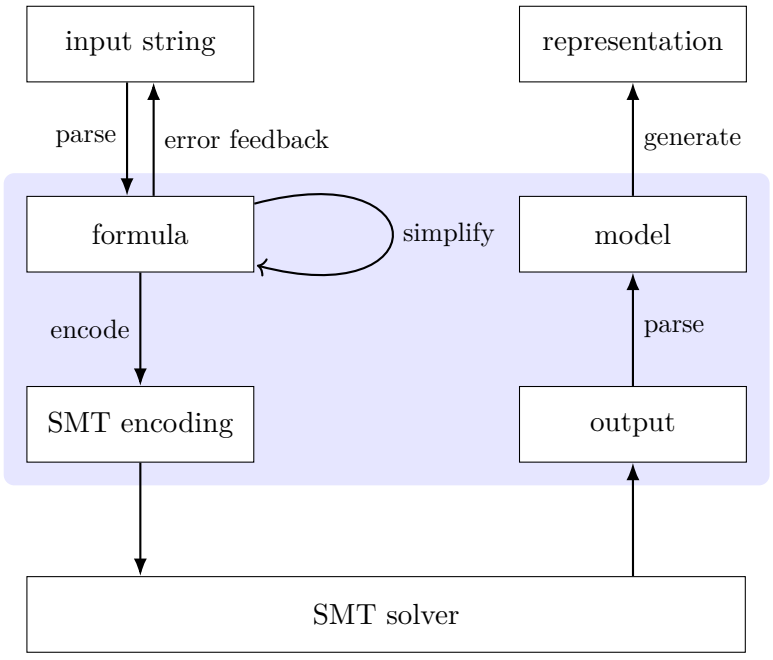


Figure 1.1: Tool Structure

CHAPTER 2

Preliminaries on Åqvist's Systems

We introduce the syntax and semantics of Åqvist's deontic logics, focusing on their modal operators and preference-based model theory. We present both the general semantic characterizations and their finite-model counterparts, followed by a review of axioms and principles for each logic, and a summary of Rozplokhas' small model constructions and their propositional encodings.

2.1 Syntax and preference-based Semantics

Åqvist's logical systems build upon classical propositional logic by introducing two modal operators: a unary modality $\Box$ representing necessity, and a binary modality $\bigcirc(\psi \mid \phi)$ expressing conditional obligation, meaning that ψ is obligatory given ϕ.

We define the set of well-formed formulas over the set of propositional variables Var and Boolean constants by the following BNF.

$$\mathcal{F} ::= v \in Var \mid \top \mid \bot \mid \neg\mathcal{F} \mid \mathcal{F} \wedge \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \Box\mathcal{F} \mid \bigcirc(\mathcal{F} \mid \mathcal{F})$$

Additionally, we have the following derived connectives:

- implication $\phi \to \psi$ short for $\neg\phi \vee \psi$
- equality $\phi \leftrightarrow \psi$ short for $(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$
- exclusive or $\phi \oplus \psi$ short for $(\neg\phi \vee \neg\psi) \wedge (\phi \vee \psi)$
- possibility $\Diamond\phi$ short for $\neg\Box\neg\phi$
- prohibition $F(\psi \mid \phi)$ short for $\bigcirc(\neg\psi \mid \phi)$
- permission $P(\psi \mid \phi)$ short for $\neg\bigcirc(\neg\psi \mid \phi)$

- $O(\phi)$, $F(\phi)$, and $P(\phi)$ short for $\bigcirc(\phi \mid \top)$, $F(\phi \mid \top)$, and $P(\phi \mid \top)$, respectively

We use the notations and definitions from the work of Rozblokhas [Roz24].

By $Sub\mathcal{F}(\phi)$ we denote the set of all subformulas of ϕ (including ϕ itself) and by $|\phi|$ the size (number of symbols) of a formula ϕ.

Definition 2.1. A *preference model* is a triple $(W, \succeq, \mathbb{V})$ where W is a (non-empty) set of worlds, $\succeq$ is a binary relation on W, and $\mathbb{V} : \mathrm{Var} \to 2^W$ is a valuation function. We denote by $W(M)$ the set of worlds of a given model.

We denote the strict preference relation by $\succ$, where $w_1 \succ w_2$ holds if and only if $w_1 \succeq w_2$ and $w_2 \not\succeq w_1$.

The semantics of the obligation operator depend on the notion of "best" worlds. Two ways of defining the best worlds are found in the literature: optimality and maximality [Par21]. Our choice of maximality is consistent with that of Rozblokhas, who uses the same definition [Roz24].

Definition 2.2. For a preference model $M = \langle W, \succeq, \mathbb{V}\rangle$ and $U \subseteq W$ we define

$$\max(U) = \{v \in U \mid \nexists\, u \in U : u \succ v\}.$$

Using the notion of maximality, an obligation $\bigcirc(\psi \mid \phi)$ is true in a model when ψ is true in all best worlds satisfying ϕ. A necessity $\Box\phi$ is true when ϕ is true in all worlds. The other Boolean operators are interpreted in the usual way, relative to each world in the model.

Note that the definition of satisfaction below follows Rozblokhas' presentation [Roz24], we only extended it by cases for $\top$ and $\bot$.

Definition 2.3. (Satisfaction) For a preference model $M = \langle W, \succeq, \mathbb{V}\rangle$ the truth set $\|\varphi\|^M$ of a formula φ is defined inductively:

- $\|\top\|^M = W$,
- $\|\bot\|^M = \emptyset$,
- $w \in \|x\|^M$ for $x \in Var$ when $w \in \mathbb{V}(x)$,
- $w \in \|\neg\psi\|^M$ when $w \notin \|\psi\|^M$,
- $w \in \|\psi_1 \wedge \psi_2\|^M$ when $w \in \|\psi_1\|^M$ and $w \in \|\psi_2\|^M$,
- $w \in \|\Box\beta\|^M$ when $\|\beta\|^M = W$,
- $w \in \|\bigcirc(\gamma \mid \alpha)\|^M$ when $\max(\|\alpha\|^M) \subseteq \|\gamma\|^M$.

Logic	Limit conditions		Properties of $\succeq$	
	limited	smooth	transitive	total
E				
F	✓			
F+(CM)		✓	✓	
G		✓	✓	✓

Table 2.1: Preference-semantical characterisations for Åqvist's logics (with maximality as the notion of bestness). This table is adapted from Figure 1 in the work of Rozblokhas [Roz24], originally derived from Parent [Par21, Table 1 and 2].

We say that w satisfies φ in model M (denoted $M, w \models \varphi$) when $w \in \|\varphi\|^M$. Furthermore, M validates φ (denoted $M \models \varphi$) when $\|\varphi\|^M = W$.

Åqvist's logics differ in their constraints on the preference relation, specifically transitivity and totality, as well as in their limit conditions, namely limitedness and smoothness.

- transitivity: for all $w_1, w_2, w_3 \in W$, if $w_1 \succeq w_2$ and $w_2 \succeq w_3$, then $w_1 \succeq w_2$
- totality: for all $w_1, w_2 \in W$, either $w_1 \succeq w_2$ or $w_2 \succeq w_1$

There are no constraints on the relation in the weakest logic **E**. **F**considers limited preference relations. For the logic **F+(CM)** it has to be transitive and for **G** additional total. Regarding the limit conditions, for the logic **F** limitedness is required, and for the logics **F+(CM)** and **G** smoothness. Limitedness ensures that for any non-empty set of worlds satisfying a formula, there is at least one maximal world with respect to the preference relation. Smoothness strengthens this by requiring that every world satisfying a formula is either itself maximal or is strictly dominated by some maximal world.

Definition 2.4. (Limit conditions) Let $M = \langle W, \succeq, \mathbb{V} \rangle$ be a model.

- M is *limited* if and only if for any formula ϕ if $\|\phi\|^M \neq \emptyset$ then $\max(\|\phi\|^M) \neq \emptyset$
- M is *smooth* if and only if for any formula ϕ and any world $w \in \|\phi\|^M$ there exists a world $u \in \max(\|\phi\|^M)$ such that either $u = w$ or $u \succ w$.

Definition 2.5. A Formula ϕ is a theorem of Åqvist's logic $\mathcal{L}$ if and only if $M \models \varphi$ for any preference model M that satisfies model conditions for logic $\mathcal{L}$ in Table 2.1.

A model M is called a countermodel if $M \not\models \phi$. It is called a $\mathcal{L}$-countermodel if it additionally fulfils the conditions for $\mathcal{L}$ in Table 2.1.

Regarding finite preference models, Rozblokhas showed that a model $M = \langle W, \succeq, \mathbb{V} \rangle$ with a finite set of worlds W is:

- limited if $\succeq$ is acyclic [Roz24, Lemma 3.12],

- smooth if $\succeq$ is transitive [Roz24, Lemma 3.18].

This yields alternative semantic descriptions, where for finite models the properties of limitedness and smoothness can be replaced by acyclicity and transitivity of the preference relation, respectively. See Table 2.2 for the overview of this alternative characterisations for all of Åqvist's logics.

Theorem 2.1. A Formula ϕ is a theorem of Åqvist's logic $\mathcal{L}$ if and only if $M \models \varphi$ for any preference model M that satisfies model conditions for logic $\mathcal{L}$ in Table 2.2.

Logic	Cardinality of W	Properties of $\succeq$		
		acyclic	transitive	total
E	finite			
F	finite	✓		
F+(CM)	finite		✓	
G	finite		✓	✓

Table 2.2: Finite-model characterisations for Åqvist's logics (with maximality as the notion of bestness), presented by Rozblokhas [Roz24, Fig. 3].

2.2 Axioms and Derived Principles

Since logic **F** extends **S5**, we present the axioms and inference rules of this system as formalised by Chellas in [Che80, Section 1.2]. (T) is called the necessity axiom and (K) the distribution axiom.

$$\Box\phi \to \phi \qquad \textbf{(T)}$$

$$\Diamond\phi \to \Box\Diamond\phi \qquad \textbf{(5)}$$

$$\Box(\phi \to \psi) \to (\Box\phi \to \Box\psi) \qquad \textbf{(K)}$$

$$\Diamond\phi \leftrightarrow \neg\Box\neg\phi \qquad \textbf{(Df}\Diamond\textbf{)}$$

$$\phi \quad \text{where } \phi \text{ is a tautology} \qquad \textbf{(PL)}$$

The inference rules of the system include the Rule of Necessitation and Modus Ponens.

$$\frac{\phi}{\Box\phi} \qquad \textbf{(RN)}$$

$$\frac{\phi \to \psi \quad \phi}{\psi} \qquad \textbf{(MP)}$$

E extends **S5** by the following additional axioms, drawing from Parent's presentation [Par21, Section 3.1].

$$\bigcirc(\psi \to \chi \mid \phi) \to (\bigcirc(\psi \mid \phi) \to \bigcirc(\chi \mid \phi)) \quad \textbf{(COK)}$$

$$\bigcirc(\psi \mid \phi) \to \Box\bigcirc(\psi \mid \phi) \quad \textbf{(Abs)}$$

$$\Box\phi \to \bigcirc(\phi \mid \psi) \quad \textbf{(Nec)}$$

$$\Box(\phi \leftrightarrow \psi) \to (\bigcirc(\chi \mid \phi) \leftrightarrow \bigcirc(\chi \mid \psi)) \quad \textbf{(Ext)}$$

$$\bigcirc(\phi \mid \phi) \quad \textbf{(Id)}$$

$$\bigcirc(\chi \mid \phi \wedge \psi) \to \bigcirc(\psi \to \chi \mid \phi) \quad \textbf{(Sh)}$$

F extends **E** by one extra axiom. Note that we use the equivalent formula $\neg\Box\neg A$ for possibility $\Diamond A$, meaning that A must be true in at least one world.

$$\neg\Box\neg A \to (\bigcirc(B \mid A) \to \neg\bigcirc(\neg B \mid A)) \quad \textbf{(D*)}$$

F+(CM) and **G** extend logic **F** by adding the rule of Cautious Monotony and Specificity Preservation, respectively.

$$(\bigcirc(B \mid A) \wedge \bigcirc(C \mid A)) \to \bigcirc(C \mid A \wedge B) \quad \textbf{(CM)}$$

$$(\neg\bigcirc(\neg B \mid A) \wedge \bigcirc(B \to C \mid A)) \to \bigcirc(C \mid A \wedge B) \quad \textbf{(Sp)}$$

We also introduce two of the principles derived from the axioms of **E** by Parent [Par21].

$$\text{If } \vdash B \to C \text{ then } \vdash \bigcirc(B \mid A) \to \bigcirc(C \mid A) \quad \textbf{(RW)}$$

$$\bigcirc(B \mid A) \wedge \bigcirc(C \mid A) \to \bigcirc(B \wedge C \mid A) \quad \textbf{(AND)}$$

2.3 Small Model Construction

A small model construction for every Åqvist's logic $\mathcal{L}$ was presented in [Roz24]. He shows, for an arbitrary $\mathcal{L}$-countermodel M of some formula φ, how to construct an $\mathcal{L}$-countermodel with a polynomially bounded number of worlds with respect to $|\varphi|$. Rozblokhas calls this process a rearrangement of a model.

A rearranged model consists of:

- a finite number of worlds from M
- copies of some of them
- a new preference relation on the selected worlds

The goal is to construct a rearranged model satisfying the same subformulas of φ as the original model. For the modalities $\Box$ and $\bigcirc$, the evaluation in a world requires consideration of other worlds. To distinguish between modalities that are validated and those that are not by a model M, the following sets are defined:

- Box^+: validated $\Box\phi$ formulas,
- Box^-: non-validated $\Box\phi$ formulas,
- Ob^+: validated $\bigcirc(\psi \mid \phi)$ formulas,
- Ob^-: non-validated $\bigcirc(\psi \mid \phi)$ formulas.

Falsification in the rearranged model can be achieved by selecting an arbitrary world from the corresponding set for the following cases:

- For the formula φ:
$$W \setminus ||\varphi||^M$$
- For each formula $\Box\phi \in \text{Box}^-$:
$$W \setminus ||\phi||^M$$
- For each formula $\bigcirc(\psi \mid \phi) \in \text{Ob}^-$:
$$\max(||\phi||^M) \setminus ||\psi||^M$$

To select an arbitrary world from a set, the representation function $rep\colon (2^W \setminus \{\emptyset\}) \to W$ is used. The set of falsifying worlds is defined as follows.

Definition 2.6. (Falsifying worlds) For a model $M = \langle W, \succeq, V\rangle$ such that $M \not\models \varphi$, $\text{Fal}(\varphi, M) = \text{rep}(W \setminus ||\varphi||^M) \cup \text{Fal}^{\Box}(\varphi, M) \cup \text{Fal}^{\bigcirc}(\varphi, M)$, where $\text{Fal}^{\Box}(\varphi, M) = \{\text{rep}(W \setminus ||\beta||^M) \mid \Box\beta \in \text{Box}^-(\varphi, M)\}$, $\text{Fal}^{\bigcirc}(\varphi, M) = \{\text{rep}(\max(||\alpha||^M) \setminus ||\gamma||^M) \mid \bigcirc(\gamma \mid \alpha) \in \text{Ob}^-(\varphi, M)\}$.

Rozblokhas' construction is based on blocks consisting of a set of worlds and a preference relation on them.

Definition 2.7. A *block* on M is a tuple $\langle U, \succeq_U\rangle$ where $U \subseteq W(M)$ and $\succeq_U$ is a binary relation on U. We will use $W(B)$ to refer to the set of worlds in B. For a given M and $U \subseteq W(M)$, we will consider the blocks of the following forms:

- $antichain(U) = \langle U, \succeq^a\rangle$, where $\succeq^a$ is an *empty relation*;
- $chain(S) = \langle \{w_i\}_{i=1}^n, \succeq^{ch}\rangle$ if $S = [w_1, \ldots, w_n]$ is a finite ordered sequence of worlds and $w_i \succeq^{ch} w_j$ iff $i \leq j$;
- $clique(U) = \langle U, \succeq^{cl}\rangle$ where $u_1 \succeq^{cl} u_2$ for all $u_1, u_2 \in U$.

Composite constructions combine blocks with an additional preference relation on them. Their purpose is to generate a rearranged model for a given model.

Definition 2.8. (Composite construction) A composite construction on M is a tuple $\langle L, \preceq_L, \mathcal{B}\rangle$ where L is a set of labels, $\preceq_L$ is a binary relation on L, and $\mathcal{B}$ is a labelling function that maps every label from L into a block on M. Each composite construction $\mathbf{c} = \langle L, \preceq_L, \mathcal{B}\rangle$ on $M = \langle W, \preceq, \mathcal{V}\rangle$ generates a model $gen(\mathbf{c}) = \langle W^{gen}, \preceq^{gen}, \mathcal{V}^{gen}\rangle$, where

- $W^{gen} = \{(l, w) \mid l \in L, w \in W(\mathcal{B}(l))\}$;
- $(l_1, w_1) \preceq^{gen} (l_2, w_2)$ iff either $l_1 \preceq_L l_2$ or both $l_1 = l_2$ and $w_1 \preceq_U w_2$ for $\mathcal{B}(l_1) = \langle U, \preceq_U\rangle$;
- $(l, w) \in \mathcal{V}^{gen}(x)$ iff $w \in \mathcal{V}(x)$.

Figure 2.1 shows composite constructions for all of Åqvist's logics. We will not go into detail on how Rozplokhas developed the constructions and which conditions they fulfil.

2.4 Propositional encoding for Åqvist's logics

[Roz24] provided an encoding of his small model constructions into classical propositional logic (see Appendix C). Consistent with his method, these encodings cover all of Åqvist's logics.

For any given formula φ and logic $\mathcal{L}$ he defines a propositional formula $F_{\mathcal{L}}(\varphi)$, which is valid if and only if φ is valid in $\mathcal{L}$. $F_{\mathcal{L}}(\varphi)$ encodes a countermodel M for φ with worlds $\{w_1 \dots w_{N(\varphi)}\}$, where $N(\varphi)$ is the size bound which can be derived from the small model constructions. Rozplokhas defines the following variables used in the encoding.

- $p_{i,j}$ for $1 \le i, j \le N(\varphi)$ encoding $w_i \succeq w_j$
- v_i^{ψ} for $1 \le i \le N(\varphi)$ and $\phi \in Sub\mathcal{F}(\varphi)$ encoding $M, w_i \models \psi$

If ψ is not a propositional variable, v_i^{ψ} depends on the v-variables for the immediate subformulas of ψ. The dependences are encoded by the following set of propositional equivalences.

$$
\begin{aligned}
\mathcal{C}^{ev}(\varphi) = \quad & \{ \quad v_i^{\neg\psi} \iff (\neg v_i^{\psi}) \quad \mid \quad (\neg\psi) \in Sub\mathcal{F}(\varphi) \quad \}_{1\le i\le N(\varphi)} \\
\cup \{ \; & v_i^{\psi_1\wedge\psi_2} \iff (v_i^{\psi_1} \wedge v_i^{\psi_2}) \mid (\psi_1 \wedge \psi_2) \in Sub\mathcal{F}(\varphi) \; \}_{1\le i\le N(\varphi)} \\
\cup \{ \; & v_i^{\Box\beta} \iff (\textstyle\bigwedge_{j=1}^{N(\varphi)} v_j^{\beta}) \mid (\Box\beta) \in Sub\mathcal{F}(\varphi) \; \}_{1\le i\le N(\varphi)} \\
\cup \{ \; & v_i^{\bigcirc(\gamma \mid \alpha)} \iff (\textstyle\bigwedge_{j=1}^{N(\varphi)}(v_j^{\gamma} \vee \neg v_j^{\alpha} \vee (\bigvee_{t=1}^{N(\varphi)}(p_{t,j} \wedge \neg p_{j,t} \wedge v_t^{\alpha})))) \\
& \qquad\qquad \mid \bigcirc(\gamma \mid \alpha) \in Sub\mathcal{F}(\varphi) \; \}_{1\le i\le N(\varphi)}
\end{aligned}
$$

To encode transitivity and totality of $\succeq$, based on their definitions, the following sets of formulas are used.

$$
\begin{aligned}
\mathcal{C}^{trans}(\varphi) &= \{ \, (p_{i,j} \wedge p_{j,k}) \implies p_{i,k} \, \}_{1\le i,j,k\le N(\varphi)} \\
\mathcal{C}^{total}(\varphi) &= \{ \, p_{i,j} \vee p_{j,i} \, \}_{1\le i,j\le N(\varphi)}
\end{aligned}
$$

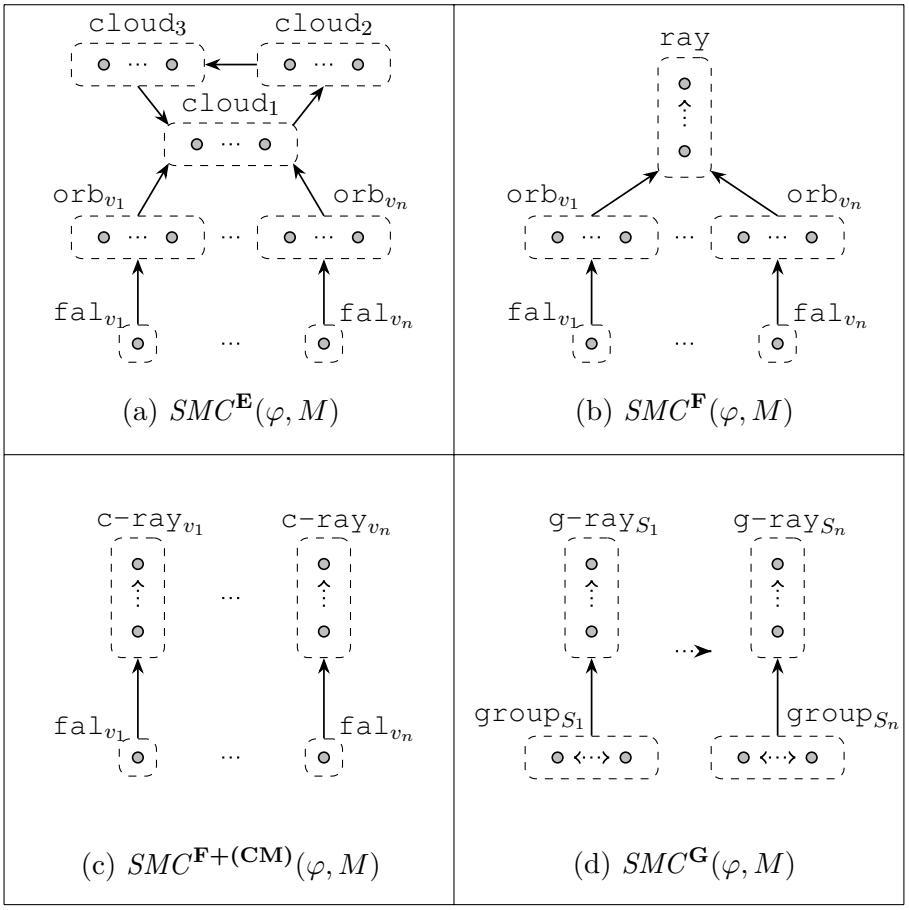


Figure 2.1: Small model constructions for Åqvist's logics developed in [Roz24, Fig. 2]. Gray circles represent worlds, dashed rectangles represent blocks. Symbol $\cdots$ inside a block indicates an antichain, $\hat{\vdots}$ indicates a chain, and $\leftrightarrow$ indicates a clique. Solid arrows represent the preference relation $\succeq_L$ between blocks: an arrow from a block l_1 to a block l_2 means $l_2 \succeq_L l_1$. The arrow $\dashrightarrow$ between blocks in construction $SMC^{\mathbf{G}}(\varphi, M)$ means that there is a linear order on blocks. Note that the preference relation in constructions $SMC^{\mathbf{E}}(\varphi, M)$ and $SMC^{\mathbf{F}}(\varphi, M)$ is not transitive.

The relation $\succeq$ is acyclic, if there exists a relation $\succeq^t$ that is transitive, irreflexive and contains $\succeq$. Using additional variables $t_{i,j}$ for $1 \leq i, j \leq N(\varphi)$ encoding the fact $w_i \succeq^t w_j$, we encode acyclicity with the following set of formulas.

$$\begin{aligned} \mathcal{C}^{acyclic}(\varphi) = \quad & \{\, (t_{i,j} \wedge t_{j,k}) \implies t_{i,k} \,\}_{1 \leq i,j,k \leq N(\varphi)} \\ \cup\ & \{\, \neg t_{i,i} \,\}_{1 \leq i \leq N(\varphi)} \\ \cup\ & \{\, p_{i,j} \implies t_{i,j} \,\}_{1 \leq i,j \leq N(\varphi)} \end{aligned}$$

To encode the countermodels for all of Åqvist's logics, we combine the sets of formulas in

the following ways. The encoded models falsify φ in world w_1.

$$
\begin{aligned}
F_{\mathbf{E}}(\varphi) &= \neg v_1^{\varphi} \wedge \bigwedge \mathcal{C}^{ev}(\varphi) \\
F_{\mathbf{F}}(\varphi) &= \neg v_1^{\varphi} \wedge \bigwedge \mathcal{C}^{ev}(\varphi) \wedge \bigwedge \mathcal{C}^{acyclic}(\varphi) \\
F_{\mathbf{F+(CM)}}(\varphi) &= \neg v_1^{\varphi} \wedge \bigwedge \mathcal{C}^{ev}(\varphi) \wedge \bigwedge \mathcal{C}^{trans}(\varphi) \\
F_{\mathbf{G}}(\varphi) &= \neg v_1^{\varphi} \wedge \bigwedge \mathcal{C}^{ev}(\varphi) \wedge \bigwedge \mathcal{C}^{trans}(\varphi) \wedge \bigwedge \mathcal{C}^{total}(\varphi)
\end{aligned}
$$

CHAPTER 3

Formula Parsing

This chapter describes how formulas are parsed within the tool, from input syntax to internal object representations. We introduce the supported operators and their categorization, explain how formulas are tokenized and parsed based on operator precedence, and describe the handling of associative operations and error feedback.

We allow eight operators, which can be categorised hierarchically as seen in Figure 3.1. There are two unary operators, negation and necessity, and six binary operators. The binary operators can further be grouped into associative and non-associative ones, which are implication and the conditional obligation. This distinction will be important for the encoding of formulas.

For the associative operators we introduce the new categories *dominant-value* operations and *parity-dependent* operations. The result of a dominant-value operation can be determined by the presence or absence of a single dominant value. A disjunction is true if one of its arguments is true and a conjunction is true if none of its arguments is false. For parity-dependent operations the result depends on the parity of true or false values. Exclusive or yields true if and only if the number of true arguments is odd. Analogously the result of an equality is true exactly if the number of false arguments is even. More detailed explanations of these categories can be found in sections 4.3.1 and 4.3.2, where we will use their properties for the simplification of formulas.

Formulas are internally represented by objects for top, bottom, variables, and composed formulas using operators on variables and constants. For each operation there is one object type representing composed formulas using this operation on formula objects. Objects for unary and binary operations save one and two formulas as arguments respectively. Binary operations are stored in the exact way they are input by the user, preserving the order of the arguments.

When checking if two binary operation objects are equal the method depends on the properties of the operation.

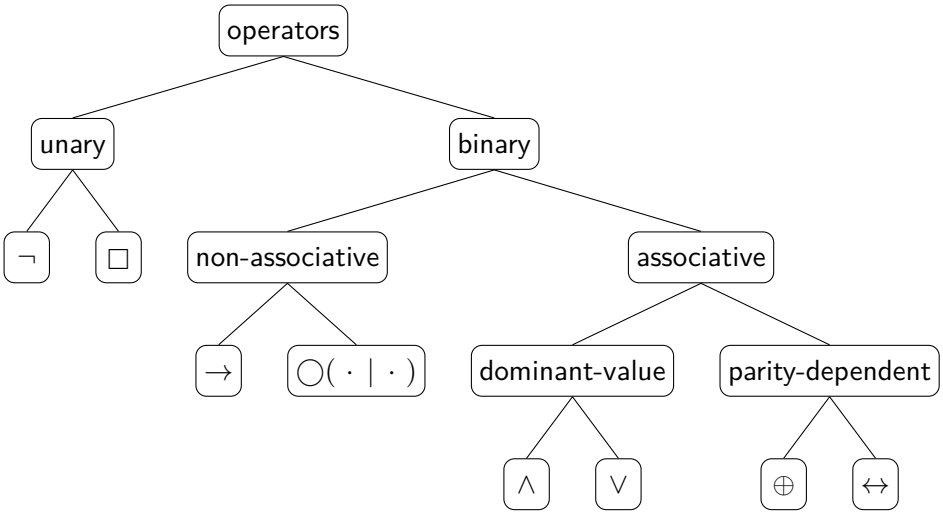


Figure 3.1: Hierarchical Categorisation of Operators

For non-associative operations we simply check if the first and second arguments are equal respectively.

But if the operation is associative we need to consider all arguments connected with the same operators. As an example, we consider the formula $\neg b \oplus a \oplus \Box z$. Figure 3.2 shows two different object representations of the formula, which correspond to the formulas $(\neg b \oplus a) \oplus \Box z$ and $\neg b \oplus (a \oplus \Box z)$. Although the formulas are equivalent, the objects would not be considered equivalent only comparing their two arguments.

(a) ExclusiveOr(ExclusiveOr(Negation(b), a), Necessity(z))

(b) ExclusiveOr(Negation(b), ExclusiveOr(a, Necessity(z)))

Figure 3.2: Two different object representations of the formula $\neg b \oplus a \oplus \Box z$.

To properly group arguments of an associative operation across various object structures, we go recursively through the operation objects on two arguments to gather arguments sharing the same operator. One can imagine the recursive argument extraction as a depth first search over the syntax tree of the formula.

For instance by traversing the syntax tree of the formula $r \wedge r' \wedge \Box s \wedge (\neg t \oplus (u \wedge v))$

shown in Figure 3.3 we obtain the following list of arguments.

$$[\, r, r', \Box s, (\neg t \oplus (u \wedge v)\,]$$

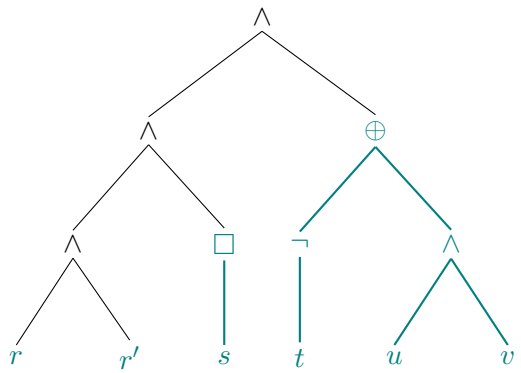


Figure 3.3: Syntax Tree of $r \wedge r' \wedge \Box s \wedge (\neg t \oplus (u \wedge v))$

Comparing the list of arguments the object representations in Figure 3.2 are considered equal.

To deal with different orders of the arguments such as in the formulas $a \vee b \vee c$ and $b \vee a \vee c$, we sort the arguments first.

Therefore we need an order on formulas.

Let's start by defining a function giving the kind of a formula.

Definition 3.1. (Kind of a formula)

$$Kind(\phi) = \begin{cases} \text{Bottom} & \text{if } \phi = \bot \\ \text{Top} & \text{if } \phi = \top \\ \text{Variable} & \text{if } \phi \in \text{Var} \\ \text{Negation} & \text{if } \phi \text{ is of the form } \neg\phi_1 \\ \text{Necessity} & \text{if } \phi \text{ is of the form } \Box\phi_1 \\ \text{Conjunction} & \text{if } \phi \text{ is of the form } (\phi_1 \wedge \phi_2) \\ \text{Disjunction} & \text{if } \phi \text{ is of the form } (\phi_1 \vee \phi_2) \\ \text{ExclusiveOr} & \text{if } \phi \text{ is of the form } (\phi_1 \oplus \phi_2) \\ \text{Implication} & \text{if } \phi \text{ is of the form } (\phi_1 \rightarrow \phi_2) \\ \text{Equality} & \text{if } \phi \text{ is of the form } (\phi_1 \leftrightarrow \phi_2) \\ \text{Obligation} & \text{if } \phi \text{ is of the form } \bigcirc(\phi_1 \mid \phi_2) \end{cases} \quad \text{where } \phi_1, \phi_2 \in Sub\mathcal{F}(\phi)$$

We define an order on all kinds of the formulas.

Definition 3.2. (Order on kinds)

$$\begin{aligned} &\mathit{Bottom} <_k \mathit{Top} <_k \mathit{Variable} <_k \mathit{Negation} <_k \mathit{Necessity} <_k \mathit{Implication} \\ &\quad <_k \mathit{Obligation} <_k \mathit{Conjunction} <_k \mathit{Disjunction} <_k \mathit{ExclusiveOr} <_k \mathit{Equality} \end{aligned}$$

This order is used in the order on formulas.

Definition 3.3. (Order on formulas)

$$\begin{aligned} \phi <_f \psi \Leftrightarrow\ & \mathit{Kind}(\phi) <_k \mathit{Kind}(\psi) \\ & \vee (\mathit{Kind}(\phi) = \mathit{Kind}(\psi) \wedge \mathrm{c}(\phi, \psi)) \end{aligned}$$

$$\mathrm{c}(\phi,\psi) \Leftrightarrow \begin{cases} \text{false} & \text{if } k \in \{\text{Bottom, Top}\} \\ \phi <_{lex} \psi & \text{if } k = \text{Variable} \\ \mathit{Arg}(\phi) = \mathit{Arg}(\psi) & \text{if } k \in \{\text{Negation, Necessity}\} \\ l(\phi) < l(\psi) \vee (l(\phi) = l(\psi) \wedge r(\phi) < r(\psi)) & \text{if } k \in \{\mathit{Implication}, \mathit{Obligation}\} \\ \mathit{Args}(\phi) <_l \mathit{Args}(\psi) & \text{otherwise} \end{cases}$$

where $k = \mathit{Kind}(\phi) = \mathit{Kind}(\psi)$.

The following auxiliary functions are used:

- $l(\phi) = \phi_1$ for $\phi = (\phi_1 \to \phi_2)$ or $\phi = \bigcirc(\phi_1 \mid \phi_2)$
- $r(\phi) = \phi_2$ for $\phi = (\phi_1 \to \phi_2)$ or $\phi = \bigcirc(\phi_1 \mid \phi_2)$
- $\mathrm{Arg}(\phi) = \psi$ for $\phi = \neg\psi$ or $\phi = \Box\psi$
- $\mathrm{Args}(\phi) = [\phi_1, \ldots, \phi_n]$ for $\phi = f(\phi_1, \ldots, \phi_n)$ and $f \in \{\wedge, \vee, \oplus, \leftrightarrow\}$

As a final step, we define an order on lists of formulas.

Definition 3.4. (Order on lists of formulas)

$$\begin{aligned} [\phi_1, \ldots \phi_n] <_l [\psi_1, \ldots \psi_n] \Leftrightarrow\ & n < m \\ & \vee (n = m \\ & \quad \wedge\ \exists\, k \leq n \text{ s.t. } \forall\, 1 \leq i < k.\ \alpha_i = \beta_i \wedge \alpha_k <_f \beta_k) \end{aligned}$$

where $[\alpha_1, \ldots \alpha_n]$ and $[\beta_1, \ldots \beta_m]$ are the recursive orderings of $[\phi_1, \ldots \phi_n]$ and $[\psi_1, \ldots \psi_m]$ respectively.

Note that this definition is an instance of structural recursion over the inductively defined set of formulas. It is well-founded because each recursive step is applied to the structurally simpler subformulas.

The arguments of associative operation objects are sorted by the defined formula order to ensure that equivalent formulas map to identical argument lists.

3.1 Input Format

The input format uses infix notation with single characters or combinations of characters representing the allowed operators, as shown in Table 3.1. The characters '∼', '&', '/' and '#' can be used for negation, conjunction, disjunction and exclusive or, respectively. For implication the strings "− >" and "< −" can be used and "< − >" for equality. Obligation is represented by a combination of 'O', parenthesis and '|'.

To represent true or false the worlds themselves can simply be written, as the parser is case-insensitive only in this case.

Table 3.1: Allowed Operations and Constants

Operation Name	**Input Symbols/Strings**	**Display Symbol**
Negation	∼	¬
Necessity	[ ]	□
Conjunction	&	∧
Disjunction	/	∨
Exclusive Or	#	⊕
Obligation	$O(\cdot \mid \cdot)$	$\bigcirc(\cdot \mid \cdot)$
Implication	− >, < −	→, ←
Equality	< − >	↔
Top	true	⊤
Bottom	false	⊥

3.1.1 Identifier syntax

The variable nomenclature is based on Haskell's conventions, which are well suited for mathematical variables and include the helpful prime to denote a modified version of a variable (e.g. x').

Haskell only allows variable names that start with a lowercase letter [M$^+$10], whereas this tool also accepts names starting with an uppercase letter. Variable names must begin with a letter, which can be lowercase or uppercase. After the initial letter, identifiers can include:

- lowercase and uppercase letters
- digits
- underscore _
- prime '

More formally, the syntax for identifiers can be described in a Backus-Naur Form (BNF) style.

$$
\begin{aligned}
\langle \text{identifier} \rangle &::= \langle \text{letter} \rangle\ \langle \text{id} \rangle^* \\
\langle \text{id} \rangle &::= \langle \text{letter} \rangle \mid \langle \text{digit} \rangle \mid \text{'} \mid \_ \\
\langle \text{letter} \rangle &::= \text{a} \mid \cdots \mid \text{z} \mid \text{A} \mid \cdots \mid \text{Z} \\
\langle \text{digit} \rangle &::= 0 \mid \cdots \mid 9
\end{aligned}
$$

3.2 Operator Precedence

The unary operators, negation and necessity, have the highest precedence. Conjunction has the highest precedence of the binary operators, followed by disjunction and exclusive or with the same precedence. Implication and equality have a lower precedence and obligation has the weakest.

Table 3.2 shows the operators ordered from highest to lowest precedence.

Table 3.2: Operator Precedence

Operator	**Symbol**	**Precedence Value**
Negation, Necessity	$\neg, \Box$	5
Conjunction	$\wedge$	4
Disjunction, Exclusive Or	$\vee$, $\oplus$	3
Implication, Equality	$\rightarrow$, $\leftarrow$, $\leftrightarrow$	2
Obligation	$\bigcirc(\cdot \mid \cdot)$	1

3.3 Tokenizing

The parser starts by tokenizing the input string, as shown in the first line of Algorithm 3.1. Operator symbols are stored as characters in the token list, where operators entered as multiple characters are replaced by a certain representing symbol, which are shown in Table 3.3. Other characters are concatenated to a string and if they form the string ”true”, ”false” or a valid variable name the corresponding object is created.

Table 3.3: Symbol Equivalents

Symbols	**Single Symbol**
[ ]	*
->	}
<-	{
<->	=

Consider the following input string.

```
a & ((b & c) / ~d ) & ~[]e
```

The resulting token list will appear as follows, where bold, coloured parts represent formula objects.

[**a**, &, (, (, **b**, &, **c**,), /, ∼, **d**,), &, ∼, ⋆, **e**]

In the tokenizing process we check if only allowed symbols are used in the intended way and if the variable names are valid. Any remaining errors in the input format are discovered during the next parsing step.

3.4 Parsing Tokens

The parsing of the token list is done recursively, where in the first step we handle parenthesis, as described in Algorithm 3.1, lines 2 to 4. We go through the parsing process of the following input string, starting with the call of the function described in Algorithm 3.2. We find the leftmost (opening) parenthesis and the associated closing parenthesis (lines 1 and 2).

a & **(** **b** & **c** / ∼ **d** **)** & ∼ ⋆ **e**

The sublist of tokens within parenthesis are parsed first, i.e. the parsing method is called recursively. When the token list contains no parenthesis, we handle the operators in the list until we obtain a singleton list, see lines 5 to 7 of Algorithm 3.1. As described in lines 1 and 2 of Algorithm 3.3, we start by finding the leftmost operator with the highest precedence. Because of the highest precedence values, the unary operators are selected first.

b & **c** / ∼ **d**
4 3 **5**

We replace the unary operator symbol and the token to the right of it as an argument by an accordingly new formula object (see lines 3 to 8 of Algorithm 3.3), in our example a negation of the variable d.

b & **c** / ¬**d**
4 3

When a binary operator is selected we combine the arguments to the left and right to a new object which replaces the three tokens (see lines 9 to 13 of Algorithm 3.3).

b ∧ **c** / ¬**d**
3

After the last operator got handled, the algorithm returns the following single formula object in the token list (see line 8 of Algorithm 3.1).

(**b** ∧ **c**) ∨ ¬**d**

The whole parenthesis term gets replaced by the result of the recursive call of handleParenthesis (see line 5 of Algorithm 3.2). We can now handle the remaining operators.

$$\underset{4}{\mathbf{a}\ \&\ (\mathbf{b}\ \wedge\ \mathbf{c})}\ \vee\ \underset{4}{\neg\mathbf{d}}\ \&\ \underset{5}{\sim}\ \underset{5}{*}\ \mathbf{e}$$

If multiple unary operators are next to each other, the algorithm selects the most left of them. We iteratively form an object from all these consecutive operators and the argument on the right. Therefore go through the unary operators from right to left and stepwise build a single object.

$$\sim\ *\ \mathbf{e}$$

$$\sim\ \Box\mathbf{e}$$

$$\neg\Box\mathbf{e}$$

The row of unary operators and the token after them get replaced by the single object.

$$\underset{4}{\mathbf{a}\ \&\ (\mathbf{b}\ \wedge\ \mathbf{c})}\ \vee\ \underset{4}{\neg\mathbf{d}}\ \&\ \neg\Box\mathbf{e}$$

For multiple occurrences of an operator on the same level, the algorithm first handles the leftmost of them, because of left associativity. To implement right associative operators, we would need to check if multiple occurrences of the operator with the highest precedence are in the token list and select the most right one.

$$\mathbf{a}\ \wedge\ (\underset{4}{(\mathbf{b}\ \wedge\ \mathbf{c})}\ \vee\ \neg\mathbf{d})\ \&\ \neg\Box\mathbf{e}$$

The parsing algorithm returns the following formula object for the original input string.

$$\mathbf{a}\ \wedge\ ((\mathbf{b}\ \wedge\ \mathbf{c})\ \vee\ \neg\mathbf{d})\ \wedge\ \neg\Box\mathbf{e}$$

Algorithm 3.1: parseFormula

```
Data: A formula string inputString
Result: The parsed formula
1 tokenList ← tokenise(inputString);
2 while tokenList.contains('(') do
3 |   handleParenthesis(tokenList);
4 end
5 while size(tokens) > 1 do
6 |   handleOperator(tokenList);
7 end
8 return tokenList[0];
```

Algorithm 3.2: handleParenthesis

Data: A list of tokens *tokenList*

1 *open* ← index of first '(' in *tokenList*;
2 *close* ← index of matching ')' in *tokenList* starting from *openIndex*;
3 *subTokens* ← *tokenList*[*openIndex* + 1 .. *closeIndex* − 1];
4 *result* ← parseFormula(*subTokens*);
5 *tokenList*[*open* .. *close*] ← [*result*]

Algorithm 3.3: handleOperator

Data: A list of tokens *tokenList*

1 *opIndex* ← index of leftmost operator in *tokenList* with highest precedence;
2 *operator* ← *tokenList*[*opIndex*]
3 **if** *operator is unary* **then**
4 | *start* ← *opIndex*;
5 | *end* ← index of first formula object from *opIndex*;
6 | *operandTokens* ← *tokenList*[*opIndex* + 1 .. *end*];
7 | *result* ← createUnaryFormula(*operator*, *operandTokens*);
8 **end**
9 **else**
10 | *start* ← *opIndex* − 1;
11 | *end* ← *opIndex* + 1;
12 | *result* ← createBinaryFormula(*operator*, *start*, *end*);
13 **end**
14 *tokenList*[*start* .. *end*] ← [*result*]

3.4.1 Obligation check

The two operators 'O' and '|' for obligation are only allowed together and combined with parenthesis. A variable cannot be named 'O' such as in the following input string.

```
O & P
```

We interpret '|' as obligation and check if it is only used in combination with 'O'. It is crucial that the obligation has the lowest precedence value, as stated in Table 3.2, such that the obligation operator '|' is always selected last within one recursion level.

For instance parsing the following string, we handle the parenthesis and select the obligation after the implication and obtain an obligation object.

```
O(x -> y | z)
```

When parenthesis get handled and there is an 'O' in front of the opening parenthesis, we check if the object obtained from handling the parenthesis term is an obligation formula. For instance these inputs would lead to a parser error.

```
O(n # m)
```

```
O(x)
```

Also the other way around, if there is no 'O' in front of parenthesis, the obtained object must not be an obligation object. Take a look at this invalid input string.

```
(x | x') / y
```

We also have to take care of recursion level 0. Before returning the final parsed formula, when the result is an obligation object we check if the input string begins with an 'O'. So for instance the following input string leads to a parser error.

```
[]a / b | ~c & d
```

3.5 Parser Error Feedback

To give precise feedback about a possible parser error, we also want to know where it occurred. For calculating the position we create and maintain a list of the token lengths additional to the token list itself.

Example

We go through an example with the following input string.

```
x' & true / <-> [] x
```

After tokenizing we obtain the following tokens with their lengths noted below. Note that the lengths refer to the input string, hence we store a length of 4 for '⊤' and a length of 3 for '='.

$$\begin{array}{ccccccccc} \mathbf{x'} & \& & \top & / & = & * & \mathbf{x} \\ 2 & 1 & 4 & 1 & 3 & 2 & 1 \end{array}$$

Each time we replace tokens by an expression, because we handled an operator or parenthesis recursively, we sum up the lengths of the processed tokens.

$$\begin{array}{ccccccc} \mathbf{x'} & \& & \top & / & = & \Box\mathbf{x} \\ 2 & 1 & 4 & 1 & 3 & 3 \end{array}$$

$$\begin{array}{ccccc} \mathbf{x'} \wedge & \top & / & = & \Box\mathbf{x} \\ 7 & 1 & 3 & 3 \end{array}$$

In the next step the algorithm selects the operator symbol /, where we find an operator symbol as the second argument. To calculate the position of '=' we sum up the lengths of the tokens $x' \wedge \top$ and '/' to the left of it, which results in 8.

To show the position of the error to the user, we display the input string without spaces in a monospaced font with an indicator below, as shown in Figure 3.4. The number of blanks before the indicator is the position we calculated before.

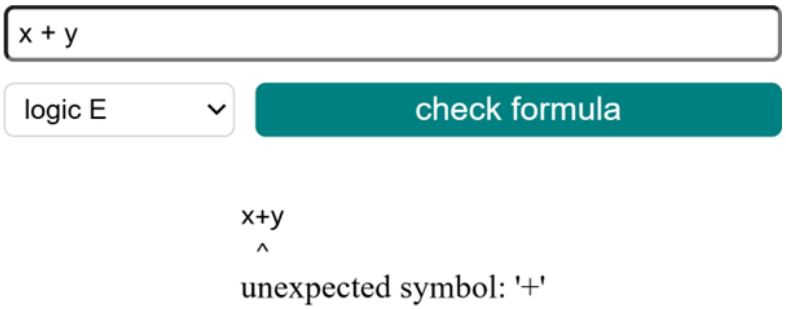


Figure 3.4: Error Feedback Example

Figure 3.5 shows several examples for error feedback.

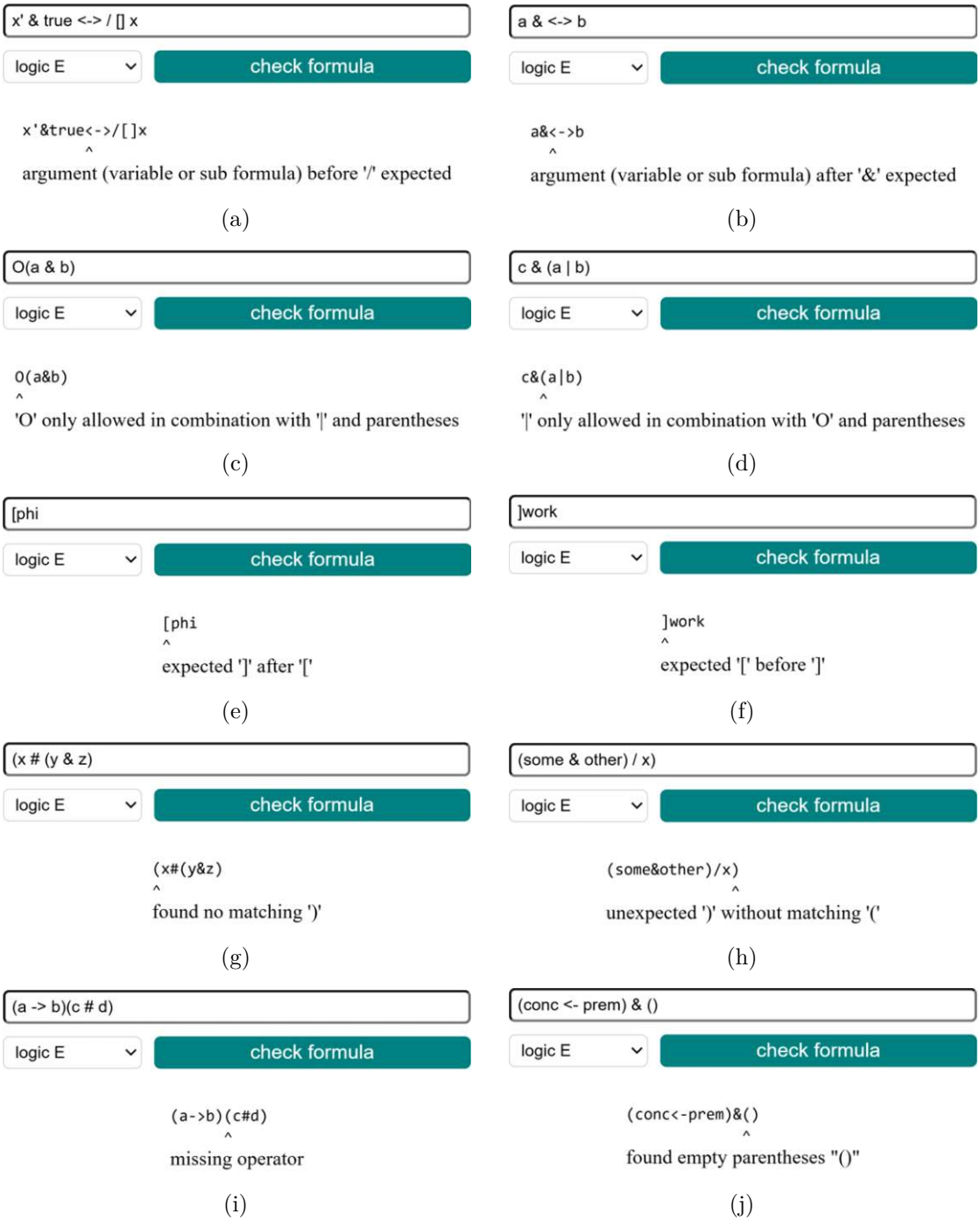


Figure 3.5: Examples for parser error feedback

CHAPTER 4

Simplification of Formulas

Before encoding a formula into propositional logic, we aim to simplify it as much as possible. Simplifications that reduce the number of necessity and conditional obligation operators also reduce the upper bound on the number of worlds required for a countermodel, leading to the largest reduction in encoding size.

Simplification serves two main purposes, first, reduction of the size of the resulting propositional encoding, and second, it can eliminate semantic redundancies, which not only improves performance but also helps express the logical structure of the formula more clearly.

In this chapter we present simplification rules. They are designed to preserve logical equivalence, ensuring that a simplified formula is satisfied by a model if and only if it is satisfied by the original formula. Therefore, they naturally also preserve satisfiability and validity of the formula. While some simplifications are purely syntactic, others rely on semantic properties of the underlying logic, such as modal axioms or the interpretation of deontic operators.

The simplifications are applied bottom-up, ensuring that subformulas are simplified before their parent expressions. This strategy prevents missed opportunities for simplification in nested contexts.

In the following sections, we define simplification rules for each type of operator.

4.1 Simplification of unary Operations

4.1.1 Negation

For negation we can simplify basic Boolean negations and double negation.

(Neg1) $\neg\top \longrightarrow \bot$

(Neg2) $\neg\bot \longrightarrow \top$

(Neg3) $\neg\neg\phi \longrightarrow \phi$

4.1.2 Necessity

Analogously to double negation, we have idempotents of the necessity operation.

(Nec1) $\Box\Box\phi \longrightarrow \Box\phi$

Proof. To prove the correctness of the rule, we show that both directions of the equivalence hold.

($\Rightarrow$) Assume $\Box\Box\phi$. By the T-axiom of system, we obtain $\Box\phi$.

($\Leftarrow$) From $\Box\phi$, we derive $\Box\Box\phi$ by the Rule of Necessitation.

Since both directions hold, we conclude that

$$\Box\Box\phi \equiv \Box\phi.$$

□

The necessity operator applied to the constants $\top$ and $\bot$ can be eliminated, since they have the same value for all worlds by definition.

(Nec2) $\Box\top \longrightarrow \top$

Proof. We prove the correctness of the rule by showing that $\Box\top$ is a tautology.

By definition, in every model, $\top$ holds in all worlds. Therefore, the formula $\Box\top$ also holds in all models. Hence, $\Box\top$ is a tautology. □

(Nec3) $\Box\bot \longrightarrow \bot$

Proof. We validate this rule by proving that $\Box\bot$ cannot be satisfied in any model.

Assume $\Box\bot$ holds. By the semantics of the necessity operator, this means that $\bot$ must hold in all worlds. However, $\bot$ is false in every world by definition. This results in a contradiction. Therefore, $\Box\bot$ is unsatisfiable. □

A necessity operator in front of an obligation formula can be dropped.

(Nec4) $\Box\bigcirc(\phi \mid \psi) \longrightarrow \bigcirc(\phi \mid \psi)$

Proof. The correctness of the simplification rule is shown by proving the logical equivalence between both sides. This is done by proving the implication in both directions.

($\Rightarrow$) Assume $\Box \bigcirc (\phi \mid \psi)$. By the absoluteness axiom (Abs) we obtain $\bigcirc(\phi \mid \psi)$.

($\Leftarrow$) Assume $\bigcirc(\phi \mid \psi)$. By the reflexivity axiom (T) of necessity, we obtain $\Box \bigcirc (\phi \mid \psi)$.

Since both directions hold, we conclude that

$$\bigcirc(\phi \mid \psi) \equiv \Box \bigcirc (\phi \mid \psi).$$

□

4.2 Simplification of non-associative Operations

4.2.1 Obligation

If an obligation has the same argument in both positions, the formula is always true.

(O1) $\quad \bigcirc(\phi \mid \phi) \quad \longrightarrow \quad \top$

Proof. The formula $\bigcirc(\phi \mid \phi)$ holds in all models by axiom (Id). □

We can simplify obligation statements with a Boolean constant as one of the arguments.

(O2) $\quad \bigcirc(\top \mid \phi) \quad \longrightarrow \quad \top$

Proof. Since $\top$ holds in all worlds by definition, it also holds in all best worlds satisfying any formula ϕ. Therefore $\bigcirc(\top \mid \phi)$ holds in all models. □

(O3) $\quad \bigcirc(\phi \mid \bot) \quad \longrightarrow \quad \top$

Proof. Assume $\bigcirc(\phi \mid \bot)$. Then ϕ would hold in all best worlds satisfying $\bot$. There exists no world satisfying $\bot$. Hence $\bigcirc(\phi \mid \bot)$ is true in every model. □

To derive a rule for simplifying the formula $\bigcirc(\bot \mid \top)$, we introduce several theorems that provide the necessary foundation.

Theorem 4.1. The formula $\bigcirc(\bot \mid \top)$ holds in a model, if there exists no best world.

Proof. Assume $\bigcirc(\bot \mid \top)$, i.e. $\bot$ holds in all best worlds satisfying $\top$. Since $\top$ is valid (i.e., it holds in all worlds), the best worlds relative to $\top$ are simply the best worlds overall. Therefore $\bot$ holds in all best worlds.

Let us assume there is a best world in a model satisfying the formula. Because of $\bigcirc(\bot \mid \top)$ this world satisfies $\bot$. This contradicts the definition of $\bot$. We conclude that there are no best worlds in any model satisfying $\bigcirc(\bot \mid \top)$. □

Theorem 4.2. There exists a model in logic **E** which has no best world.

Proof. Let $M = \langle W, \succeq, \mathbb{V} \rangle$ with $W = \{w_1, w_2, w_3\}$ and $\succeq = \{(w_1, w_2), (w_2, w_3), (w_3, w_1)\}$ be a model in logic **E**. In M none of the worlds w_1, w_2, w_3 is a best world because of $w_3 \succeq w_1$, $w_1 \succeq w_2$, and $w_2 \succeq w_3$, respectively. Therefore, no best world exists in the model M. □

Theorem 4.3. In every model in logic **F**, **F+(CM)**, or **G** there exists a best world.

Proof. **F**:
Let $M = \langle W, \succeq, \mathbb{V} \rangle$ be a model in logic **F**, i.e. $\succeq$ is acyclic.
We assume for contradiction that no best world exists.
Then for every $w \in W$, there exists $v \in W$ such that $v \prec w$.
There are two possible ways in which this can occur:
Case 1:
We have an infinite descending chain.

$$\cdots \prec w_3 \prec w_2 \prec w_1$$

Since W is finite, such a chain cannot exist.
Case 2:
The preference relation contains a cycle.

$$w_1 \prec w_l \prec \cdots \prec w_2 \prec w_1$$

This contradicts the acyclicity of $\succeq$.
Since we have a contradiction in every case, there exists a best world in each model with a finite set of worlds and an acyclic preference relation, i.e. in each model for logic **F**.

F+(CM), **G**:
Let $M = \langle W, \succeq, \mathbb{V} \rangle$ be a model, where $\succeq$ is transitive.
Suppose, for contradiction, that the model contains no best world.
Given any world $w \in W$, there is always a world $v \in W$ such that $v \prec w$ holds.
This yields an infinite descending chain.

$$\cdots \prec w_3 \prec w_2 \prec w_1$$

Since W is finite, so such a chain cannot exist.
Hence, there must be at least one $w \in W$ such that no $v \in W$ satisfies $v \prec w$.
Such a w is maximal under $\preceq$, i.e., a best world.
Therefore, every finite model in **F+(CM)** and **G** has at least one best world. □

From Theorems 4.1, 4.2, and 4.3, we derive the following result.

Theorem 4.4. The formula $\bigcirc(\bot \mid \top)$ is valid in every model of the logics **F**, **F+(CM)**, and **G**. But, it is not valid in the logic **E**.

$$\not\models_{\mathbf{E}} \bigcirc(\bot \mid \top) \quad \text{and} \quad \models_{\mathcal{L}} \bigcirc(\bot \mid \top) \quad \text{for all } \mathcal{L} \in \{\mathbf{F},\ \mathbf{F+(CM)},\ \mathbf{G}\}.$$

Proof. We proceed by case analysis over the logics.

F, **F+(CM)**, **G**:
By Theorem 4.1, the formula $\bigcirc(\bot \mid \top)$ holds in a model only if there are no best worlds.
By Theorem 4.3, every model in logic **F**, **F+(CM)**, and **G** *does* have at least one best world.
Hence, $\bigcirc(\bot \mid \top)$ is false in all models of these logics.

E:
By Theorem 4.2, there exists a model in logic **E** in which no best world exists.
By Theorem 4.1, it follows that $\bigcirc(\bot \mid \top)$ is true in this model.
Therefore, $\bigcirc(\bot \mid \top)$ is *not* equivalent to $\bot$ in logic **E**, since it can be true. □

The following simplification rule, supported by Theorem 4.4, holds in all of Åqvist's logics except **E**.

(O4) $\quad \bigcirc(\bot \mid \top) \quad \longrightarrow \quad \bot$

We can simplify obligation in some cases when it gets combined with other operations.

$$\bigcirc(\phi \mid \phi \wedge \psi) \quad \equiv \quad \top$$

Proof. By the identity principle (Id), we have:

$$\vdash \bigcirc(\phi \wedge \psi \mid \phi \wedge \psi).$$

Since $\vdash \phi \wedge \psi \to \phi$, we can apply the rule of right weakening (RW), which gives:

$$\vdash \bigcirc(\phi \wedge \psi \mid \phi \wedge \psi) \to \bigcirc(\phi \mid \phi \wedge \psi).$$

Therefore, $\bigcirc(\phi \mid \phi \wedge \psi)$ holds in all models, i.e., is a tautology. □

We use this result to derive the following generalised simplification rule for obligations with conjunctive conditions.

(O5) $\quad \bigcirc(\phi_i \mid \bigwedge_{j=1}^{n} \phi_j) \quad \longrightarrow \quad \top \qquad \text{for } 1 \leq i \leq n$

We can apply similar reasoning when the consequent of an obligation is a disjunction implied by the condition.

$$\bigcirc(\phi \vee \psi \mid \phi) \quad \equiv \quad \top$$

Proof. We know that $\vdash \phi \to \phi \vee \psi$, which is a tautology of propositional logic.

By the rule of right weakening (RW), this yields:

$$\vdash \bigcirc(\phi \mid \phi) \to \bigcirc(\phi \vee \psi \mid \phi).$$

By the (Id), we also have:

$$\vdash \bigcirc(\phi \mid \phi).$$

Applying modus ponens, we conclude:

$$\vdash \bigcirc(\phi \vee \psi \mid \phi).$$

Thus, the formula $\bigcirc(\phi \vee \psi \mid \phi)$ holds in all models, and is therefore a tautology. □

This gives rise to the following simplification rule for obligations with disjunctive consequents.

(O6) $\bigcirc(\bigvee_{j=1}^{n} \phi_j \mid \phi_i) \quad \longrightarrow \quad \top$

We can simplify obligations whose condition is the necessity of the obligated formula.

$$\bigcirc(\phi \mid \Box\phi) \quad \equiv \quad \top$$

Proof. We begin with the tautology $\vdash \Box\phi \to \phi$, that we obtain from the (T)-axiom.

By the right weakening rule (RW), this gives:

$$\vdash \bigcirc(\Box\phi \mid \Box\phi) \to \bigcirc(\phi \mid \Box\phi).$$

By the deontic identity principle (Id), we have:

$$\vdash \bigcirc(\Box\phi \mid \Box\phi).$$

Applying modus ponens, we conclude:

$$\vdash \bigcirc(\phi \mid \Box\phi).$$

Thus, $\bigcirc(\phi \mid \Box\phi)$ holds in all models, i.e. it is a tautology. □

This leads to the following simplification rule for obligations with a necessary condition.

(O7) $\bigcirc(\phi \mid \Box\phi) \quad \longrightarrow \quad \top$

4.2.2 Implication

An implication from an expression to the same one is true, so it can be replaced by $\top$ in any formula for simplification.

(I1) $\phi \to \phi \quad \longrightarrow \quad \top$

If at least one of the arguments of implication is top or bottom, we can simplify it with one of the following rules.

(I2) $\top \to \phi \quad \longrightarrow \quad \phi$

(I3) $\bot \to \phi \quad \longrightarrow \quad \top$

(I4) $\phi \to \top \quad \longrightarrow \quad \top$

(I5) $\phi \to \bot \quad \longrightarrow \quad \neg\phi$

We can simplify Boolean expressions when they are implied by their complements in the following ways.

(I6) $\neg\phi \to \phi \quad \longrightarrow \quad \phi$

(I7) $\phi \to \neg\phi \quad \longrightarrow \quad \neg\phi$

Regarding a combination of implication with necessity the following simplification rules can be applied.

(I8) $\Box\phi \to \phi \quad \longrightarrow \quad \top$

Proof. This formula holds in all models by the T-axiom of the system **S5**. □

(I9) $\neg\phi \to \neg\Box\phi \quad \longrightarrow \quad \top$

Proof. By the T-axiom of the system **S5** [Che80], $\Box\phi \to \phi$ holds in all models. Taking the contraposition gives $\neg\phi \to \neg\Box\phi$, which is logically equivalent. Therefore, the implication holds in all models. □

4.3 Simplification of associative Operations

For associative operations, simplification is best performed over multiple arguments. If done only pairwise, simplifications such as the following could be missed, since the duplicated arguments are not next to each other.

$$\bigcirc(b \mid a) \wedge b \wedge \bigcirc(b \mid a) \quad \longrightarrow \quad \bigcirc(b \mid a) \wedge b$$

We use the list of extracted arguments described in the introduction of Chapter 3, where here the sorting of the arguments is not necessary.

The simplification procedures use the function *make* which constructs a formula from a given list of arguments. For instance the *make*-function for conjunction forms the following conjunction out of the list $[\Box s,\ t,\ \top,\ \neg r]$.

$$\Box s \wedge t \wedge \top \wedge \neg r$$

Given a list with a single argument the function simply returns that argument.

4.3.1 Simplification of Dominant-value Operations

As discussed in Chapter 3 conjunction and disjunction can be grouped under the term dominant-value operations. These operations have elements, shown in Table 4.1, with certain properties and which we use in their simplification. These characteristic elements, as well as duplicated and complementary arguments are handled in the same way, such that we can come up with a general simplification procedure for these operations. These properties are caused by their duality through *De Morgan's law*.

$$\phi \vee \psi \quad \equiv \quad \neg\left(\neg\phi \wedge \neg\psi\right)$$

The result of dominant-value operations can be determined by a certain truth constant in specific cases. We call this constant the absorbing element. A conjunction containing the false constant as an argument is always false, and analogously a disjunction with the argument true constant is always true.

$$\phi \wedge \bot \quad \equiv \quad \bot$$
$$\phi \vee \top \quad \equiv \quad \top$$

The respective other true constant, when combined with another element, leaves the argument unchanged. We call this constant the neutral element, since this behaviour is analogous to the role of 0 in addition $(x + 0 = 0)$ and 1 in multiplication $(x \cdot 1 = x)$.

$$\phi \wedge \top \quad \equiv \quad \phi$$
$$\phi \vee \bot \quad \equiv \quad \phi$$

Table 4.1: Characteristic Elements for dominant-value operations

Operation	Neutral Element	Absorbing Element
Conjunction	$\top$	$\bot$
Disjunction	$\bot$	$\top$

Using the properties of the absorbing element we define the following simplification rules for conjunction and disjunction of multiple arguments.

(C1) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bot$ if $\exists\ i$ such that $\phi_i = \bot$

(D1) $\bigvee_{i=1}^{n} \phi_i \longrightarrow \top$ if $\exists\ i$ such that $\phi_i = \top$

The following simplification rules handle the neutral element of both dominant-value operations.

(C2) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bigwedge_{\substack{1 \leq i \leq n \\ \phi_i \neq \top}} \phi_i$ if $\exists\ i$ such that $\phi_i \neq \top$

(D2) $\bigvee_{i=1}^{n} \phi_i \longrightarrow \bigvee_{\substack{1 \leq i \leq n \\ \phi_i \neq \bot}} \phi_i$ if $\exists\ i$ such that $\phi_i \neq \top$

These rules address cases in which every argument is the neutral element.

(C3) $\bigwedge_{i=1}^{n} \top \longrightarrow \top$

(D3) $\bigvee_{i=1}^{n} \bot \longrightarrow \bot$

For both operators the combination of complementary arguments result in their respective absorbing element.

$$\phi \wedge \neg\phi \equiv \bot$$

$$\phi \vee \neg\phi \equiv \bot$$

Therefore, dominant-value operations over multiple formulas can also be reduced to their absorbing element if they contain a pair of complementary arguments.

(C4) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bot$ if $\exists\ j, k$ such that $\phi_j = \neg\phi_k$

(D4) $\bigvee_{i=1}^{n} \phi_i \longrightarrow \top$ if $\exists\ j, k$ such that $\phi_j = \neg\phi_k$

In the case of conjunctions, we additionally check for contradictory Boolean expressions involving necessity.

$$\Box\phi \wedge \neg\phi \equiv \bot$$

Proof. Assume $\Box\phi \wedge \neg\phi$ holds.

- From $\Box\phi$, it follows that ϕ holds in all worlds.
- From $\neg\phi$, it follows that ϕ does not hold in the actual world.

This leads to a contradiction. Therefore, the conjunction is unsatisfiable. □

$$\phi \wedge \Box\neg\phi \quad \equiv \quad \bot$$

Proof. Assume $\phi \wedge \Box\neg\phi$ holds.

- From ϕ, it follows that ϕ holds in the actual world.
- From $\Box\neg\phi$, it follows that $\neg\phi$ holds in all worlds, including the actual world.

This results in a contradiction. Hence, the conjunction is unsatisfiable. □

$$\Box\phi \wedge \Box\neg\phi \quad \equiv \quad \bot$$

Proof. Assume $\Box\phi \wedge \Box\neg\phi$ holds.

- From $\Box\phi$, it follows that ϕ holds in all worlds.
- From $\Box\neg\phi$, it follows that $\neg\phi$ holds in all worlds.

This creates a contradiction, since ϕ and $\neg\phi$ cannot both hold in any world. Thus, the conjunction is unsatisfiable. □

The following additional rules simplify a conjunction to $\bot$ when it contains two arguments that form contradictions involving stronger modalities, which are not covered by the general complementary argument rules.

(C5) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bot$ if $\exists\, i, j$ such that $\phi_i = \Box\psi$ and $\phi_j = \neg\psi$

(C6) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bot$ if $\exists\, i, j$ such that $\phi_i = \psi$ and $\phi_j = \Box\neg\psi$

(C7) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bot$ if $\exists\, i, j$ such that $\phi_i = \Box\psi$ and $\phi_j = \Box\neg\psi$

Conjunction and disjunction of the same argument can be reduced to the argument itself.

(C8) $\bigwedge_{i=1}^{n} \phi_i \longrightarrow \bigwedge_{\psi \in \{\phi_i \mid 1 \leq i \leq n\}} \psi$ if $\exists\, i, j$ s.t. $i \neq j \wedge \phi_i = \phi_j$

$$\textbf{(D5)} \qquad \bigvee_{i=1}^{n} \phi_i \quad \longrightarrow \quad \bigvee_{\psi \in \{\phi_i \mid 1 \leq i \leq n\}} \psi \qquad \text{if } \exists\, i, j \text{ s.t. } i \neq j \wedge \phi_i = \phi_j$$

The combination of necessities and obligations yields simplification rules for dominant-value operations that provide the greatest reduction in formula size, since we can reduce the number of worlds required in the countermodel. Necessities can be combined using

$$\Box(\phi) \wedge \Box(\psi) \quad \equiv \quad \Box(\phi \wedge \psi)$$

Proof. We prove both directions of the equivalence.

(⇒) The direction from left to right is given by the distribution axiom K of necessity.

(⇐) We assume $\Box(\phi \wedge \psi)$. This means that in every world, the formula $\phi \wedge \psi$ holds. Since $\phi \wedge \psi$ implies both ϕ and ψ, it follows that in each world:

$$\phi \text{ holds} \quad \text{and} \quad \psi \text{ holds.}$$

Therefore, $\Box\phi$ and $\Box\psi$ both hold. Consequently, $\Box\phi \wedge \Box\psi$ is true.

Since both directions hold, we conclude:

$$\Box(\phi) \wedge \Box(\psi) \equiv \Box(\phi \wedge \psi)$$

□

We can now use this equivalence to simplify conjunctions that mix necessity formulas with non-modal components.

$$\textbf{(C9)} \qquad \bigwedge_{i=1}^{n} \phi_i \quad \longrightarrow \quad \left(\bigwedge_{\text{Kind}(\phi_i) \neq \textit{Necessity}} \phi_i \right) \wedge \Box \left(\bigwedge_{\phi_j = \Box\psi_j} \psi_j \right)$$

A similar equation holds for obligations with a common condition.

$$\bigcirc(\phi \mid \chi) \wedge \bigcirc(\psi \mid \chi) \quad \equiv \quad \bigcirc(\phi \wedge \psi \mid \chi)$$

Proof. We prove the equation by showing the implications in both directions.

(⇒) This direction is given by the conjunction principle (And) derived from logic **E**.

(⇐) We assume $\bigcirc(\phi \wedge \psi \mid \chi)$. This means that in all best worlds satisfying χ, the formula $\phi \wedge \psi$ holds.

From this, it follows that in those same best χ-worlds:

- ϕ holds, hence $\bigcirc(\phi \mid \chi)$.

- ψ holds, hence $\bigcirc(\psi \mid \chi)$.

Therefore, it follows that $\bigcirc(\phi \mid \chi) \wedge \bigcirc(\psi \mid \chi)$.

As both implications have been shown, the equivalence holds:

$$\Box(\phi) \wedge \Box(\psi) \equiv \Box(\phi \wedge \psi)$$

□

This allows us to simplify conjunctions containing obligations with a common condition. They can be merged into a single obligation.

(C10) $$\bigwedge_{i=1}^{n} \phi_i \quad \longrightarrow \quad \left(\bigwedge_{\phi_i \neq \neg\bigcirc(\_ \mid \chi)} \phi_i \right) \wedge \bigcirc\left(\bigwedge_{\phi_j = \bigcirc(\psi_j \mid \chi)} \psi_j \;\middle|\; \chi \right)$$

From the equivalence we derive one combining obligation arguments of disjunctions.

$$\neg\bigcirc(\phi \mid \chi) \vee \neg\bigcirc(\psi \mid \chi) \quad \equiv \quad \neg\bigcirc(\phi \wedge \psi \mid \chi)$$

Proof.

$$\begin{aligned}
\neg\bigcirc(\phi \mid \chi) \vee \neg\bigcirc(\psi \mid \chi) &\equiv \neg\neg(\neg\bigcirc(\phi \mid \chi) \vee \neg\bigcirc(\psi \mid \chi)) && \text{double negation} \\
&\equiv \neg(\neg\neg\bigcirc(\phi \mid \chi) \wedge \neg\neg\bigcirc(\psi \mid \chi)) && \text{de morgans law} \\
&\equiv \neg(\bigcirc(\phi \mid \chi) \wedge \bigcirc(\psi \mid \chi)) && \text{double negation } (\times 2) \\
&\equiv \neg\bigcirc(\phi \wedge \psi \mid \chi) && (C10)
\end{aligned}$$

□

Using this equivalence, we obtain the following rule potentially combining multiple negated obligation formulas.

(D6) $$\bigvee_{i=1}^{n} \varphi_i \quad \longrightarrow \quad \left(\bigvee_{\phi_i \neq \neg\bigcirc(\_ \mid \chi)} \varphi_i \right) \vee \neg\bigcirc\left(\bigwedge_{\phi_j = \neg\bigcirc(\psi_j \mid \chi)} \psi_j \;\middle|\; \chi \right)$$

Since an implication $\phi \to \psi$ is logically equivalent to the disjunction $\neg\phi \vee \psi$, we form a corresponding equivalence for implications of formulas with obligations.

$$\bigcirc(\phi \mid \chi) \to \neg\bigcirc(\psi \mid \chi) \quad \equiv \quad \neg\bigcirc(\phi \wedge \psi \mid \chi)$$

Proof.

$$\begin{aligned}
\bigcirc(\phi \mid \chi) \to \neg\bigcirc(\psi \mid \chi) &\equiv \neg\bigcirc(\phi \mid \chi) \vee \neg\bigcirc(\psi \mid \chi) && \text{materialise implication} \\
&\equiv \neg\bigcirc(\phi \wedge \psi \mid \chi) && (D6)
\end{aligned}$$

□

This lets us rewrite such implications as a single negated obligation.

(I10) $\bigcirc(\phi \mid \chi) \to \neg\bigcirc(\psi \mid \chi) \quad \longrightarrow \quad \neg\bigcirc(\phi \wedge \psi \mid \chi)$

We apply the simplification rules in a generalised way on the list of arguments of a dominant-value operation. List reductions are the first steps. Since it can be done in linear time complexity, we start with applying the simplifications rules (C2) and (D2) by removing all occurrences of the neutral element (see line 1 of Algorithm 4.1). If the list is empty after that step, i.e. it only contained neutral elements, we return the neutral element itself. After that we remove all duplicated arguments.

We continue with early exits of the simplification procedure, by checking for the absorbing element and for complementary arguments. If we find one of both, we return the absorbing element, resulting by the rules (C1) and (D1).

The last processing step is the combination of obligation arguments by the rules (C10) and (D6).

If it was possible to combine obligations, we simplify the newly constructed obligations. To make sure that the simplification rules are applied exhaustively, we recursively call the simplification procedure for the dominant-value operation on the new list of arguments.

For conjunction we additionally combine all necessity arguments to a single one. The arguments $\Box\neg y$, $\Box x$ and $\Box(v \vee w)$ for instance can be combined to $\Box(\neg y \wedge x \wedge (v \vee w))$.

Algorithm 4.1: Simplification on the arguments of a dominant-value operation

```
   Data: A list of arguments args
   Result: A simplified formula
 1 args' ← RemoveAllOccurrences(args, neutral_element)
 2 if args' is empty then
 3 |   return neutral_element                    //(C2), (D2)
 4 end
 5 args' ← RemoveDuplicates(args')              //(C8), (D5)
 6 if args' contains absorbing_element then
 7 |   return absorbing_element                 //(C1), (D1)
 8 end
 9 if args' contains complementary arguments then
10 |   return absorbing_element                 //(C4), (C5), (C6), (C7), (D4)
11 end
12 args'' ← CombineFormulas(args')              //(C9), (C10), (D6)
13 if args'' ≠ args' then
14 |   return simplify(args'')
15 end
16 return make(args'')
```

Let's simplify the following conjunction as an example.

$$\neg a \wedge \bigcirc(b \mid a) \wedge \top \wedge \neg\bigcirc(d \wedge b \mid a) \wedge c \wedge \Box b \wedge \top \wedge \neg a \wedge \bigcirc(d \mid a) \wedge \top \wedge \Box e$$

The procedure is applied on the list of arguments.

$$[\,\neg a,\ \bigcirc(b \mid a),\ \top,\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box b,\ \top,\ \neg a,\ \bigcirc(d \mid a),\ \top,\ \Box e\,]$$

We start with by removing all occurrences of the neutral element $\top$.

$$[\,\neg a,\ \bigcirc(b \mid a),\ \top,\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box b,\ \top,\ \neg a,\ \bigcirc(d \mid a),\ \top,\ \Box e\,]$$

After checking that the list is not empty, we remove all duplicates from the list.

$$[\,\neg a,\ \bigcirc(b \mid a),\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box b,\ \neg a,\ \bigcirc(d \mid a),\ \Box e\,]$$

Since the argument list contains neither the absorbing element nor complementary arguments, we do not return the absorbing element. We continue with combining obligation formulas as well as necessity formulas.

Let us start with the necessities.

$$[\,\neg a,\ \bigcirc(b \mid a),\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box(\mathbf{b} \wedge \mathbf{e}),\ \Box b,\ \bigcirc(d \mid a),\ \Box e\,]$$

Now we combine the obligations.

$$[\,\neg a,\ \bigcirc(\mathbf{b} \wedge \mathbf{d} \mid \mathbf{a}),\ \bigcirc(b \mid a),\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box(b \wedge e),\ \bigcirc(d \mid a)\,]$$

The next step is trying to simplify the newly composed necessities and obligations, which has no effect for this formula. Recalling the simplification procedure on this new list of arguments, we this time find complementary arguments, which results in the returned, absorbing element $\bot$.

$$[\,\neg a,\ \bigcirc(b \wedge d \mid a),\ \neg\bigcirc(d \wedge b \mid a),\ c,\ \Box(b \wedge e)\,]$$

4.3.2 Simplification of Parity-dependent Operations

Parity-dependent operations include equality and exclusive or (see Chapter 3). The two operators are also related by a duality, as expressed by the following equivalence.

$$\phi \oplus \psi \quad \equiv \quad \neg\,(\phi \leftrightarrow \psi)$$

Analogously to dominant-value operations each of these operations also have a neutral element.

$$\phi \oplus \bot \quad \equiv \quad \phi$$
$$\phi \leftrightarrow \top \quad \equiv \quad \phi$$

Since obviously multiple neutral elements as arguments also do not have any effect, the properties of this characteristic element result in these simplification rules.

(X1) $\left(\bigoplus_{i=1}^{n} \phi_i\right) \oplus \left(\bigoplus_{j=1}^{m} \bot\right) \longrightarrow \bigoplus_{i=1}^{n} \phi_i$

(E1) $\left(\overset{n}{\underset{i=1}{\longleftrightarrow}} \phi_i\right) \leftrightarrow \left(\overset{m}{\underset{j=1}{\longleftrightarrow}} \top\right) \longrightarrow \overset{n}{\underset{i=1}{\longleftrightarrow}} \phi_i$

The following rules cover the case that the operation is only over neutral elements.

(X2) $\left(\bigoplus_{j=1}^{m} \bot\right) \longrightarrow \bot$

(E2) $\left(\overset{m}{\underset{j=1}{\longleftrightarrow}} \top\right) \longrightarrow \top$

The parity-dependent are characterised by a significant constant we call the inverting element. When it gets combined with any argument, it negates (inverts) the argument.

$$\phi \oplus \top \equiv \neg\phi$$
$$\phi \leftrightarrow \bot \equiv \neg\phi$$

The effect of several inverting elements as arguments depends on their number, since double negation is equivalent to the original expression. We can also think of it as combining all occurrences of the inverting element to a single element. This dependence on the parity of the elements gives these operations their name.

Theorem 4.5. Let $\circledast$ be a parity-dependent binary operator, logical equivalence ($\iff$) or exclusive disjunction ($\oplus$). A chain of n identical converting elements combined by $\circledast$ evaluates to:

- the neutral element if n is even,
- the converting element if n is odd.

$$\bigoplus_{i=1}^{n} \top = \begin{cases} \bot & \text{if } n \text{ is even} \\ \top & \text{if } n \text{ is odd} \end{cases}$$

$$\overset{n}{\underset{i=1}{\longleftrightarrow}} \bot = \begin{cases} \top & \text{if } n \text{ is even} \\ \bot & \text{if } n \text{ is odd} \end{cases}$$

Since the neutral element has no effect, we can drop an even number of inverting elements from the list of arguments. However, if the number is odd, we negate the whole remaining term. The following simplification rules formalise this behaviour over several arguments.

(X3) $\left(\bigoplus_{i=1}^{n} \phi_i\right) \oplus \left(\bigoplus_{j=1}^{m} \top\right) \longrightarrow \bigoplus_{i=1}^{n} \phi_i$ if m is even

(X4) $\left(\bigoplus_{i=1}^{n} \phi_i\right) \oplus \left(\bigoplus_{j=1}^{m} \top\right) \longrightarrow \neg\left(\bigoplus_{i=1}^{n} \phi_i\right)$ if m is odd

(E3) $\left(\overleftrightarrow{}_{i=1}^{n} \phi_i\right) \leftrightarrow \left(\overleftrightarrow{}_{j=1}^{m} \bot\right) \longrightarrow \overleftrightarrow{}_{i=1}^{n} \phi_i$ if m is even

(E4) $\left(\overleftrightarrow{}_{i=1}^{n} \phi_i\right) \leftrightarrow \left(\overleftrightarrow{}_{j=1}^{m} \bot\right) \longrightarrow \neg\left(\overleftrightarrow{}_{i=1}^{n} \phi_i\right)$ if m is odd

The combination of an argument with itself using a parity-dependent operation yields their respective neutral element.

$$\phi \oplus \phi \equiv \bot$$

$$\phi \leftrightarrow \phi \equiv \top$$

Therefore pairs of duplicated arguments can be dropped from the arguments.

(X5) $\bigoplus_{i=1}^{n} \phi_i \longrightarrow \bigoplus_{\substack{1\le i\le n \\ i\neq j,\ i\neq k}} \phi_i$ if $j \neq k$ and $\phi_j = \phi_k$ and $n > 2$

(E5) $\overleftrightarrow{}_{i=1}^{n} \phi_i \longrightarrow \overleftrightarrow{\bigoplus}_{\substack{1\le i\le n \\ i\neq j,\ i\neq k}} \phi_i$ if $j \neq k$ and $\phi_j = \phi_k$ and $n > 2$

In case the operator is only applied on a duplicated pair the term results in the neutral element.

(X6) $\phi \oplus \phi \longrightarrow \bot$

(E6) $\phi \leftrightarrow \phi \longrightarrow \top$

Combining complements with each other results in the inverting element.

$$\phi \oplus \neg\phi \equiv \top$$

$$\phi \leftrightarrow \neg\phi \equiv \bot$$

Combined with the fact that the combing a term with the inverting element results in the negation of the term, we obtain the following rules.

(X7) $\bigoplus_{i=1}^{n} \phi_i \longrightarrow \neg\left(\bigoplus_{\substack{1\le i\le n \\ i\neq j,\ i\neq k}} \phi_i\right)$ if $j \neq k$ and $\phi_j = \neg\phi_k$ and $n > 2$

(E7) $\overset{n}{\underset{i=1}{\longleftrightarrow}} \phi_i \longrightarrow \neg \left(\underset{\substack{1 \leq i \leq n \\ i \neq j,\ i \neq k}}{\longleftrightarrow} \phi_i \right)$ if $j \neq k$ and $\phi_j = \neg\phi_k$ and $n > 2$

Here we again have rules for the trivial cases with two elements.

(X8) $\phi \oplus \neg\phi \longrightarrow \top$

(E8) $\phi \leftrightarrow \neg\phi \longrightarrow \bot$

The last simplification rules deal with negated arguments. For negations of the whole term and at least one argument the following equivalences hold.

$$\neg(\neg\phi \oplus \psi) \equiv \phi \oplus \psi$$
$$\neg(\neg\phi \leftrightarrow \psi) \equiv \phi \leftrightarrow \psi$$

Thereby we can remove the negation of a parity-dependent operation over multiple arguments together with the negation of one argument. We introduce the following two additional rules for simplifying negations.

(Neg4) $\neg \left(\bigoplus_{i=1}^{n} \phi_i \right) \longrightarrow \left(\underset{\substack{1 \leq i \leq n \\ i \neq j}}{\bigoplus} \phi_i \right) \oplus \psi_j$ if $\phi_j = \neg\psi_j$

(Neg5) $\neg \left(\overset{n}{\underset{i=1}{\longleftrightarrow}} \phi_i \right) \longrightarrow \left(\underset{\substack{1 \leq i \leq n \\ i \neq j}}{\longleftrightarrow} \phi_i \right) \leftrightarrow \psi_j$ if $\phi_j = \neg\psi_j$

For negations of multiple arguments we can use the following equations.

$$\neg\phi \oplus \neg\psi \equiv \phi \oplus \psi$$
$$\neg\phi \leftrightarrow \neg\psi \equiv \phi \leftrightarrow \psi$$

The following simplification rules remove negations from two arguments for parity-dependent operations over several arguments.

(X9) $\bigoplus_{i=1}^{n} \phi_i \longrightarrow \left(\underset{\substack{1 \leq i \leq n \\ i \neq j,\ i \neq k}}{\bigoplus} \phi_i \right) \oplus \psi_j \oplus \psi_k$ if $\phi_j = \neg\psi_j$, $\phi_k = \neg\psi_k$

(E9) $\overset{n}{\underset{i=1}{\longleftrightarrow}} \phi_i \longrightarrow \left(\underset{\substack{1 \leq i \leq n \\ i \neq j,\ i \neq k}}{\longleftrightarrow} \phi_i \right) \leftrightarrow \psi_j \leftrightarrow \psi_k$ if $\phi_j = \neg\psi_j$, $\phi_k = \neg\psi_k$

Table 4.2: Characteristic Elements for parity-dependent operations

Operation	Neutral Element	Inverting Element
Exclusive Or	$\bot$	$\top$
Equality	$\top$	$\bot$

Using this analogous simplification rules, we can describe a generalised procedure to simplify a list of arguments of a parity-dependent operation, shown in Algorithm 4.2. We start as with dominant-value operations, by reducing the number of arguments by removing all occurrences of the neutral element (see line 1). Based on the simplification rules (X5) and (E5) we can pairwise remove duplicated elements (see line 2).

In the case that the list only consisted of neutral elements and an even number of duplicated arguments, such that this results in an empty list, the result of the simplification is the neutral element (lines 3 to 5). Since these two list reductions are performed consecutively, we only need to check once.

We remove all occurrences of the inverting element and count the number of occurrences. This number is saved in the variable *negCount*. Since using the simplification rules (X7) and (E7) complementary arguments yield the inverting element, we pairwise remove them and add the count of pairs to the *negCount*.

In lines 8 to 19 of the procedure we check if the list is empty and return an element depending on the parity of the *negCount*. If the count is even we return the neutral element, otherwise the inverting element - see rules (X5) and (E5).

In the last steps we try to reduce the number of negations in the formula. We consider the following equivalences.

$$\neg\phi \oplus \psi \quad \equiv \quad \phi \oplus \neg\psi \quad \equiv \quad \neg(\phi \oplus \psi)$$

$$\neg\phi \leftrightarrow \psi \quad \equiv \quad \phi \leftrightarrow \neg\psi \quad \equiv \quad \neg(\phi \leftrightarrow \psi)$$

The resulting formula has to be negated if the negation count is an odd number. We try to avoid this negation of the whole term. Therefore, if any argument is negated, we try to remove it together with the negation from a single argument, using the rules (Neg4) and (Neg5). Only in the case that this fails, the final formula will be negated.

In case the removing of the negation was successful, we can pairwise remove negations from arguments using the rules (X9) and (E9).

After all the simplifications on the argument list, we compose them again to a formula.

Let's go through the procedure for the following formula.

$$\top \oplus \neg g \oplus b \oplus \neg a \oplus \bot \oplus \neg b \oplus f \oplus \top \oplus e \oplus \neg d \oplus \neg c \oplus b \oplus b \oplus \bot \oplus \top \oplus \neg f$$

Algorithm 4.2: Simplification on the arguments of a parity-dependent operation

Data: A list of arguments $args$
Result: A simplified formula

```
1  args' ← RemoveAllOccurrences(args, neutral_element)   //(X1), (E1)
2  args' ← CancelDuplicates(args')                       //(X5), (E5)
3  if args' is empty then
4  |  return neutral_element                             //(X2), (E2), (X6), (E6)
5  end
6  negCount ← 0
7  removed ← TryRemoveInvertingElement(args')            //(X3), (E3)
8  if removed then
9  |  negCount ← 1                                       //(X4), (E4)
10 end
11 negCount ← negCount + CancelComplements(args')       //(X7), (E7)
12 if args' is empty then
13 |  if negCount is even then
14 |  |  return neutral_element
15 |  end
16 |  else
17 |  |  return inverting_element                        //(X8), (E8)
18 |  end
19 end
20 if negCount is odd then
21 |  removedNegation ← RemoveSingleNegation(args')     //(Neg4), (Neg5)
22 |  if not removedNegation then
23 |  |  f ← make(args')
24 |  |  return Negation(f)
25 |  end
26 end
27 args' ← PairwiseRemoveNegations(args')               //(X9), (E9)
28 return make(args')
```

We extract the following list of arguments.

$$[\,\top,\ \neg g,\ b,\ \neg a,\ \bot,\ \neg b,\ f,\ \top,\ e,\ \neg d,\ \neg c,\ b,\ b,\ \bot,\ \top,\ \neg f\,]$$

In the first step we remove all occurrences of $\bot$, the neutral element of exclusive or.

$$[\,\top,\ \neg g,\ b,\ \neg a,\ \bot,\ \neg b,\ f,\ \top,\ e,\ \neg d,\ \neg c,\ b,\ b,\ \bot,\ \top,\ \neg f\,]$$

We also remove all pairs of duplicated arguments.

$$[\,\top,\ \neg g,\ \neg a,\ b,\ \neg b,\ f,\ \top,\ e,\ \neg d,\ \neg c,\ b,\ b,\ \top,\ \neg f\,]$$

If the list would be empty now, we would return the neutral element.

We the single inverting element $\top$, leading to a *negCount* of 1.

$$[\,\neg g,\ \neg a,\ \neg b,\ f,\ e,\ \neg d,\ \neg c,\ b,\ \top,\ \neg f\,]$$

Also pairs of complementary arguments get counted and removed from the list. After adding the count of pairs *negCount* is now 3.

$$[\,\neg g,\ \neg a,\ \neg b,\ f,\ e,\ \neg d,\ \neg c,\ b,\ \neg f\,]$$

The if-condition in line 12 is false since the list is not empty.

Since the negation count is an odd number, we try to remove the negation from a single argument.

$$[\,\neg g,\ \neg a,\ e,\ \neg d,\ \neg c\,]$$

In the next step we pairwise remove further negations.

$$[\,g,\ \neg a,\ e,\ \neg d,\ \neg c\,]$$

Finally, we compose the simplified argument list back into a new formula, which is equivalent to the original one.

$$g \oplus a \oplus e \oplus d \oplus \neg c$$

CHAPTER 5

Z3 encoding

This chapter presents the encoding of formulas from Åqvist's logics into formulas of classical propositional logic, making them suitable for processing by the SMT-solver Z3 [DMB08]. We focus on establishing size bounds for countermodels and provide detailed explanations of the naming conventions and the encoding of operations employed in the tool.

If a formula is parsed successfully, it is then encoded into classical propositional logic, following the method developed by Rozplokhas [Roz24], and passed to Z3. The solver determines whether a countermodel exists for the formula, thereby showing that it is invalid.

5.1 Size Bound Calculation

The required number of worlds in a countermodel forms the basis for encoding formulas into propositional logic. The calculations of the size bound for each of Åqvist's logics are based on the small model constructions by Rozplokhas [Roz24].

The number of falsifying worlds, used in the sizebound calculations, can be calculated by the following formula.

$$|\mathrm{Fal}(\varphi, M)| = 1 + |\mathrm{Fal}^{\Box}(\varphi, M)| + |\mathrm{Fal}^{\bigcirc}(\varphi, M)|$$

Without determining for each subformula $\Box\phi$ and $\bigcirc(\psi \mid \phi)$, whether it must be satisfied, we use the following upper bound for the number of falsifying worlds.

$$|\mathrm{Fal}(\varphi, M)| \leq 1 + |\mathrm{Nec}(\varphi)| + |\mathrm{Obl}(\varphi)| \quad \text{where}$$

$$\mathrm{Nec}(\varphi) = \{\, \psi \in \mathit{Sub}\mathcal{F}(\varphi) \mid \mathrm{Kind}(\psi) = \mathrm{Necessity} \,\}$$

$$\mathrm{Obl}(\varphi) = \{\, \psi \in \mathit{Sub}\mathcal{F}(\varphi) \mid \mathrm{Kind}(\psi) = \mathrm{Obligation} \,\}$$

Figure 2.1 illustrates the small model constructions for Åqvist's logics. These constructions allow us to determine the maximum number of worlds required for a countermodel in each logic.

5.1.1 E

The construction for logic **E** is done by one world block per falsifying world $v_i \in \text{Fal}(\varphi, M)$. We add one block orb_{v_i} for each world v_i. The three blocks $cloud_1$, $cloud_2$, and $cloud_3$ are copies of the same block, they contain the same number of worlds. We obtain the following formula for the required number of worlds in a countermodel in logic **E**.

$$numberWorlds_E = 3 \cdot |\mathcal{B}(cloud_1)| + \sum_{i=1}^{n} (|\mathcal{B}(orb_{v_i})| + |\mathcal{B}(fal_{v_i})|)$$

For the different types of blocks we obtain the following size bounds by their definitions.

Rozplokhas states that each block (in the constructions for each of Åqvist's logics) contains at most $|Cond(\varphi)|$ blocks. Therefore the following inequalities hold.

$$|\mathcal{B}(cloud_1)| \leq |Cond(\varphi)|$$
$$|\mathcal{B}(orb_{v_i})| \leq |Cond(\varphi)|$$

In the construction, n denotes the number of falsifying worlds.

$$n = |Fal(\varphi, M)|$$

We know that each block labelled fal_{v_i} contains a single world.

$$|\mathcal{B}(fal_{v_i})| = |\{v_i\}| = 1$$

From these size bounds for the different types of blocks, we can define the following formula for the size bound.

$$sizeBound_E = 3 \cdot |Cond(\varphi)| + |Fal(\varphi, M)| \cdot (|Cond(\varphi)| + 1)$$

5.1.2 F

The composite construction for logic **F** is similar to that for logic **E**. The only difference is that, instead of the three *cloud*-blocks, we have a single block labelled ray.

$$numberWorlds_F = |\mathcal{B}(ray)| + n \cdot (|\mathcal{B}(orb)| + |\mathcal{B}(fal)|)$$

For the single new block in the construction, we again use the upper bound of $|Cond(\varphi)|$ worlds for each block.

$$|\mathcal{B}(ray)| \leq |Cond(\varphi)|$$

Combined with the upper bounds for the different blocks we derived for logic **E**, we obtain following size bound formula for logic **F**.

$$sizeBound_F = |Cond(\varphi)| + |Fal(\varphi, M)| \cdot (|Cond(\varphi)| + 1)$$

5.1.3 F + (CM)

The small model construction for logic **F+(CM)** consists of two blocks for each falsifying world v_i, one labelled $c\text{-}ray_{v_i}$ and one labelled fal_{v_i}. The total number of worlds required for this logic can be computed by the following formula.

$$numberWorlds_{F+(CM)} = \sum_{i=1}^{n} (|\mathcal{B}(falv_i)|) + \sum_{i=1}^{n} (|\mathcal{B}(c\text{-}ray_{v_i})|)$$

Since each block contains at most $|Cond(\varphi)|$ worlds, we obtain the following upper bound.

$$|\mathcal{B}(c\text{-}ray_{v_i})| \leq |Cond(\varphi)|$$

The following equivalence follows.

$$\sum_{i=1}^{n} (|\mathcal{B}(c\text{-}ray_{v_i})|) = n \cdot |Cond(\varphi)|$$

We already know that each block labelled fal_{v_i} consists of exactly one world, which leads to the following equation.

$$\sum_{i=1}^{n} (|\mathcal{B}(falv_i)|) = \sum_{i=1}^{n} 1 = n$$

Using the derived size bounds, we can compute the size bound for logic **F+(CM)** with the following formula.

$$sizeBound_{F+(CM)} = |Fal(\varphi, M)| \cdot (|Cond(\varphi)| + 1)$$

5.1.4 G

The small model construction for the logic **G** is based on a stratification of worlds. We can calculate the number of worlds we need in a countermodel for logic **G** with the following formula, where $[S_1, \ldots S_n]$ is the stratification of $Fal(\varphi, M)$.

$$numberWorlds_G = \sum_{i=1}^{n} |\mathcal{B}(g\text{-}ray_{S_i})| + \sum_{i=1}^{n} |\mathcal{B}(group_{S_i})|$$

Following the definition of Rozplokhas, a stratification is defined as follows:

Definition 5.1. (Stratification) For $M = \langle W, \preceq, V \rangle$ and a finite set $U \subseteq W$, a sequence $[S_1, \ldots, S_n]$ of non-empty subsets of W is called a stratification of U when U is the disjoint union of subsets $\{S_i\}_{i=1}^n$ (i.e., $U = \bigcup_{i=1}^n S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$), and for every $u_i \in S_i$ and $u_j \in S_j$, we have $u_i \preceq u_j$ if and only if $i \geq j$.

Hence, $Fal(\phi, M)$ is the disjoint union of the sets $\{S_i\}_{i=1}^{n}$. Therefore the number of all worlds covered by the stratification is equal to the number of worlds in $Fal(\varphi, M)$.

$$\sum_{i=1}^{n} |S_i| = |Fal(\varphi, M)|$$

Each block labelled $groups_{S_i}$ contains a clique of the worlds in S_i, and therefore contains $|S_i|$ worlds. From the equation above we know that the total number of worlds in the stratification is $|Fal(\varphi, M)|$, and we can derive the following equality.

$$\sum_{i=1}^{n} |\mathcal{B}(groups_{S_i})| = |Fal(\varphi, M)|$$

Since each blocks contains a maximum of $|Cond(\phi)|$ worlds, the following equation holds.

$$\sum_{i=1}^{n} |\mathcal{B}(g\text{-}ray_{S_i})| \leq n \cdot |Cond(\phi)|$$

From this, the size bound for logic **G** can be calculated by the following formula.

$$sizeBound_G = |Fal(\varphi, M)| + n \cdot |Cond(\phi)|$$

For logic **VTA** (i.e. **G**), Friedman and Halpern [FH94] use a different approach for constructing a countermodel. They take one world from the original model for each modality. This results in a linear size bound of the model and the following formula, which we use in our tool. Here Nec(φ) and Obl(φ) denote the set of all subformulas of φ that are of the kind Necessity and Obligation, respectively.

$$sizeBound_G = 1 + |\mathrm{Nec}(\varphi)| + |\mathrm{Obl}(\varphi)| \quad \text{where}$$

$$\mathrm{Nec}(\varphi) = \{\, \psi \in Sub\mathcal{F}(\varphi) \mid \mathrm{Kind}(\psi) = \mathrm{Necessity} \,\}$$

$$\mathrm{Obl}(\varphi) = \{\, \psi \in Sub\mathcal{F}(\varphi) \mid \mathrm{Kind}(\psi) = \mathrm{Obligation} \,\}$$

5.2 Variable Names

In his encoding Rozplokhas uses propositional variables denoting the preference relation and the true formulas in the worlds of a model. Variables of the form $p_{i,j}$ encode that world w_i is preferred to world w_j. To encode that a formula ϕ holds in world w_i, we use the variable p_i^{ϕ}.

For the Z3 input, we use ':' as a separator between the the base variable and its parameters. The name for a p-variable $p_{i,j}$ is $p{:}i{:}j$. Formulas are represented in their input format without blank spaces, for instance $v{:}O(b|a){:}3$ for the v-variable $v_3^{\bigcirc(b\,|\,a)}$. See Table 5.1 for examples of variable names.

In the representation of associative operators the arguments are sorted by the order on formulas $<_f$ (see Definition 3.3).

By doing so equivalent formulas such as $a \wedge b$ and $b \wedge a$ are both encoded by the variables $v{:}a\&b{:}i$ for each world i. This ordering ensures that subformulas equivalent through associativity are encoded uniformly, which can lead to smaller and more efficient encodings.

Implications are always represented in the rightward direction, so that the formulas $a \rightarrow b$ and $b \leftarrow a$ are both encoded using formulas of the form $v{:}\ a- > b{:}i$.

For parity-dependent operations the negations are additionally moved to the first arguments. We first remove any negations from the arguments and sort them by the defined order on formulas. Then we reapply a negation to each of the first arguments, equal to the count of negations initially removed. The following formulas are all represented by variables of the form $v{:}\sim(a \wedge \neg c)\#b{:}i$.

- $\neg(a \wedge \neg c) \oplus b$
- $b \oplus \neg(a \wedge \neg c)$
- $\neg b \oplus (a \wedge \neg c)$
- $(a \wedge \neg c) \oplus \neg b$

Table 5.1: Z3 Variable Naming

Variable	Name in encoding	Examples
v_i^ϕ	$v{:}\phi{:}i$	$v{:}O(b\|a){:}3, v{:}y'{:}1$
$p_{i,j}$	$p{:}i{:}j$	$p{:}4{:}2, p{:}1{:}3$
$t_{i,j}$	$t{:}i{:}j$	$t{:}3{:}12, t{:}5{:}2$

We define p-variables representing the preference relation over all worlds. For logic **F** we additional have to analogously define t-variables to encode that the relation is acyclic.

```
(declare-const p:1:1 Bool)
(declare-const p:1:2 Bool)
(declare-const p:1:3 Bool)
...
(declare-const p:9:9 Bool)
```

There is a v-variable for each world and subformula of the formula to encode. For the formula $\Box x \vee x'$ we obtain the following variables.

```
(declare-const v:x:1 Bool)
(declare-const v:x':1 Bool)
(declare-const v:[]x:1 Bool)
```

```
(declare-const v:[]x/x':1 Bool)

(declare-const v:x:2 Bool)
(declare-const v:x':2 Bool)
(declare-const v:[]x:2 Bool)
(declare-const v:[]x/x':2 Bool)
```

5.3 Single v-variable for Necessity and Obligation

Since necessity and obligation is defined generally over all worlds, we can replace the corresponding v-variables by a single one. In the example above instead of `v:[]x:1` and `v:[]x:2` we define the single variable `v:[]x`.

For the formula $\bigcirc(b \mid a)$ the variable `v:O(b|a)` replaces the following variable declarations.

```
(declare-const v:O(b|a):1 Bool)
(declare-const v:O(b|a):2 Bool)
(declare-const v:O(b|a):3 Bool)
(declare-const v:O(b|a):4 Bool)
(declare-const v:O(b|a):5 Bool)
(declare-const v:O(b|a):6 Bool)
(declare-const v:O(b|a):7 Bool)
```

5.4 Encoding of Operations

We encode all composed formulas according to the definition of the propositional equivalences by Rozplokhas [Roz24] (see Section 2.4).

5.4.1 Negation

For the formula $\neg c$ and the world w_2 the encoding is as follows.

```
(assert (= v:∼c:2 (not v:c:2)))
```

5.4.2 Necessity

The encoding of the single v-variable (see Section 5.3) for the necessity formula $\Box d$ is given below.

```
(assert (=
    v:[]d
    (and v:d:1 v:d:2 v:d:3 v:d:4 v:d:5 v:d:6 v:d:7 v:d:8 v:d:9)
))
```

5.4.3 Conjunction

By the propositional equivalence of conjunction, we obtain the following encoding for the formula $\neg c \wedge \Box d$.

```
(assert (= v:(∼c&[]d):2 (and v:∼c:2 v:[]d:2)))
```

5.4.4 Disjunction

The encoding of disjunction can be derived from the encodings of negation and conjunction done by [Roz24] using De Morgans Law and double negation.

$$
\begin{aligned}
v_i^{\psi_1 \vee \psi_2} &\iff v_i^{\neg\neg(\psi_1 \vee \psi_2)} && \text{double negation} \\
&\iff v_i^{\neg(\neg\psi_1 \wedge \neg\psi_2)} && \text{de morgans law} \\
&\iff \neg v_i^{\neg\psi_1 \wedge \neg\psi_2} && \text{negation encoding} \\
&\iff \neg(v_i^{\neg\psi_1} \wedge v_i^{\neg\psi_2}) && \text{conjunction encoding} \\
&\iff \neg(\neg v_i^{\psi_1} \wedge \neg v_i^{\psi_2}) && \text{negation encoding} \\
&\iff \neg\neg v_i^{\psi_1} \vee \neg\neg v_i^{\psi_2} && \text{de morgans law} \\
&\iff v_i^{\psi_1} \vee v_i^{\psi_2} && \text{double negation } (\times 2)
\end{aligned}
$$

Therefore we obtain the following encoding for the formula $\bigcirc(b \mid a) \vee (\neg c \wedge \Box d)$ and world 2.

```
(assert (= v:(O(b|a)/(∼c&[]d)):2 (or v:O(b|a):2 v:(∼c&[]d):2)))
```

5.4.5 Obligation

The encoding of obligations have the greatest formula size. Like mentioned in Section 5.3 we encode a general v-variable over all worlds.

```
(assert (= v:O(b|a)
    (and
        (or
            v:b:1
            (not v:a:1)

            (and p:1:1 (not p:1:1) v:a:1)
            (and p:2:1 (not p:1:2) v:a:2)
            (and p:3:1 (not p:1:3) v:a:3)
            (and p:4:1 (not p:1:4) v:a:4)
            (and p:5:1 (not p:1:5) v:a:5)
            (and p:6:1 (not p:1:6) v:a:6)
            (and p:7:1 (not p:1:7) v:a:7)
```

```
            )

            (or
                v:b:2
                (not v:a:2)

                (and p:1:2 (not p:2:1) v:a:1)
                (and p:2:2 (not p:2:2) v:a:2)
                ...
                (and p:7:2 (not p:2:7) v:a:7)
            )

            ...

            (or
                v:b:7
                (not v:a:7)

                (and p:1:7 (not p:7:1) v:a:1)
                (and p:2:7 (not p:7:2) v:a:2)
                ...
                (and p:7:7 (not p:7:7) v:a:7)
            )
        )
    ))
```

5.4.6 Exclusive Or

Exclusive or is useful in our automated reasoning tool because it allows us to enforce that exactly one of two conditions holds, which frequently arises in encoding deontic constraints. The operation is encoded based on the following equivalence.

$$\phi \oplus \psi \iff (\neg\phi \vee \neg\psi) \wedge (\phi \vee \psi)$$

Therefore the formula $x \oplus x'$ is equivalent to the formula $(\neg x \vee \neg x') \wedge (x \vee x')$. Hence it can be encoded in the following way.

```
(assert (=
    v:x#x':2
    (and
        (or
            (not v:x:2)
            (not v:x':2)
        )
```

```
            (or
                v:x:2
                v:x':2
            )
        )
    ))
```

Note that we directly use the `not` operator of Z3, avoiding the need to introduce separate v-variables `v: x:2` and `v: x':2`.

5.4.7 Implication

For the encoding of implications we use this equivalence.

$$\phi \to \psi \iff \neg\phi \vee \psi$$

Hence we can encode $a \to b$ by encoding it as $\neg a \vee b$ in the following way.

```
(assert (=
    v:a->y:2
    (or
        (not v:a:2)
        v:y:2
    )
))
```

Implication can also be used in the other direction using the input string `<-`. We encode a formula $\phi \leftarrow \psi$ simply like the equivalent formula $\phi \to \psi$. See for instance the encoding of the formula $c \leftarrow p$ for world 2.

```
(assert (=
    v:c<-p:2
    (or
        (not v:p:2)
        v:c:2
    )
))
```

5.4.8 Equality

For equality we can directly use the corresponding operation of Z3, which looks as this for the formula $(r \wedge s) \leftrightarrow t$ and world 2.

```
(assert (= v:(r&s)<->t:2 (= v:(r&s):2 v:t:2)))
```

5.5 Encoding the Properties of the Preference Relation

Using Rozblokhas' encodings for the properties of the preference relation we can add, depending on the logic, some of the following assertions for n worlds and for $1 \leq i, j, k \leq n$.

From the straight forward encoding of transitivity and totality we obtain the following assertions:

- for transitivity

```
(assert (impl
    (and p:i:j p:j:k)
    p:i:k
))
```

- for totality

```
(assert (or
    p:i:j
    p:j:i
))
```

To encode acyclicity we add assertions over the Boolean t-variables.

```
(assert (impl
    (and t:i:j t:j:k)
    t:i:k
))

(assert
    (not t:i:i)
)

(assert (impl
    p:i:j
    t:i:j
))
```

5.6 Finding a Minimal Countermodel

When reasoning about a formula, we call Z3 multiple times, as presented in Algorithm 5.1. To establish validity, we first attempt to construct a countermodel using the maximum number of worlds allowed by the encoding. If no countermodel exists at this upper bound, then no countermodel exists at all, and the formula is valid (see lines 1–3). Since most formulas admit a model with significantly fewer worlds, we continue with encodings and SMT calls with an increasing number of worlds starting from 1 (lines 4 to 9).

Algorithm 5.1: Minimal countermodel search

Data: Formula ϕ, size bound *sizeBound*
Result: A countermodel of minimal size, or `isValid` if none exists

```
1 if not found getModel(sizeBound) then
2 |   return isValid
3 end
4 for i ← 1 to sizeBound do
5 |   m ← getModel(i)
6 |   if found m then
7 |   |   return m
8 |   end
9 end
```

CHAPTER 6

Countermodel Representation

We compare three different options to represent countermodels: using text only, a directed graph, or a matrix.

6.1 Text representation

The most straightforward way to present a countermodel is textually, listing the preference relations in the formal notation. For example,

$$w_2 \succeq w_1$$

$$w_3 \succeq w_1$$

$$w_3 \succeq w_2$$

The simple approach of representing each pair of worlds separately can be improved by combining multiple relations into a single line.

$$w_3 \succeq w_1, w_2$$

$$w_2 \succeq w_1$$

The computation of these compact relations is presented in Algorithm 6.1. We maintain two maps that store, for each world, the lists of better and inferior worlds, respectively. Additionally, for efficiency, we store the size of each list.

We find the world w with the maximum number better or inferior worlds. The list of worlds l forms the left (resp. right) side of the relation line if these are better (resp.inferior) worlds. In lines 24 and 25 of Algorithm 6.1 the list l and the corresponding count entry is removed from the maps. For each world in l we remove the list from the respective other map and decrement the count by 1 (see lines 25 to 29).

To check if the same relation holds for additional worlds to w, we go through the stored list of the remaining worlds. If l is a sublist of the stored list, we add the corresponding world to the same side of the relation as w. Since l has the maximum length of all lists, it can only be a sublist if it is the equivalent list. Hence, for efficiency, we first check if the lists have the same length.

The process of finding the world connected to most other worlds, updating the lists and counts, as well as potentially adding further worlds, is repeated until all relations are handled.

Example

Let us go through the process for the following preference relation.

$$w_1 \succeq w_2 \quad w_1 \succeq w_3 \quad w_1 \succeq w_4 \quad w_1 \succeq w_5$$
$$w_3 \succeq w_2 \quad w_3 \succeq w_5 \quad w_4 \succeq w_2 \quad w_4 \succeq w_5$$

We obtain the following mappings, assigning each world a list of better and inferior worlds, respectively, represented by their numerical identifiers.

inferior:

$1 \mapsto [2, 3, 4, 5]$

$3 \mapsto [2, 5]$

$4 \mapsto [2, 5]$

better:

$2 \mapsto [1, 3, 4]$

$3 \mapsto [1]$

$4 \mapsto [1]$

$5 \mapsto [1, 3, 4]$

World 1 has the most relations to inferior worlds, from which we obtain the relation $w_1 \succeq w_2, w_3, w_4, w_5$. Since there is no world with the mapping to the same list, we do not extend the relation. We remove the mapping to the list of better worlds and the entry of 1 in all lists of inferior worlds. Mappings for which the list is now empty can be removed.

inferior:

$3 \mapsto [2, 5]$

$4 \mapsto [2, 5]$

better:

$2 \mapsto [3, 4]$

Algorithm 6.1: Compact preference relation

Data: Map from world to better worlds *better*
Map from world to inferior worlds $inferior$
Map from world to the number of better worlds *betterCounts*
Map from world to the number of inferior worlds $inferiorCounts$
Result: A compact representation of the preference relation

```
1  while betterisnotempty do
2      wBet, maxBet ← max(betterCounts)
3      wInf, maxInf ← max(inferiorCounts)
4      worldSide ← [w]
5      listSide ← l
6      if maxBet > maxInf then
7          w ← wBet
8          l ← better[w]
9          sameRels ← better
10         sameCounts ← betterCount
11         otherRels ← inferior
12         otherCounts ← inferiorCounts
13         relation ← [worldSide, listSide]
14     end
15     else
16         w ← wInf
17         l ← inferior[w]
18         sameRels ← inferior
19         sameCounts ← inferiorCount
20         otherRels ← better
21         otherCounts ← betterCounts
22         relation ← [listSide, worldSide]
23     end
24     sameRels.remove(w)
25     sameCounts[w] = 0
26     for v ∈ l do
27         otherRels[v].remove(w)
28         otherCounts[v] = otherCounts[v] − 1
29     end
30     for (v, m) ∈ sameRels do
31         if length(l) = length(m) then
32             if l = m then
33                 worldSide.add(v)
34             end
35         end
36     end
37 end
38 return relation
```

$$3 \mapsto \emptyset$$

$$4 \mapsto \emptyset$$

$$5 \mapsto [3, 4]$$

We continue with the mapping from world 3 to its inferior worlds. Because world 4 has the same mapping here, we combine the both to the relation $w_3, w_4 \succeq w_2, w_5$. After removing the two worlds from the corresponding mappings to better worlds, only mappings to empty lists remain.

inferior:

$$3 \mapsto \emptyset$$

$$4 \mapsto \emptyset$$

better:

$$2 \mapsto \emptyset$$

$$5 \mapsto \emptyset$$

The given preference relation can be expressed in two lines.

$$w_1 \succeq w_2, w_3, w_4, w_5$$

$$w_3, w_4 \succeq w_2, w_5$$

6.2 Directed graph

A directed graph combines all the information of a model in one graphical representation. Worlds are represented by nodes with their number, i.e. node i representing world w_i. Each node can be denoted with the true variables of the corresponding world. There is an arrow from node i to node j when $w_j \succeq w_i$. A countermodel for the formula $a \wedge \bigcirc(c \mid b)$ with the size bound as a number of worlds, can for example be represented by the graph in Figure 6.1.

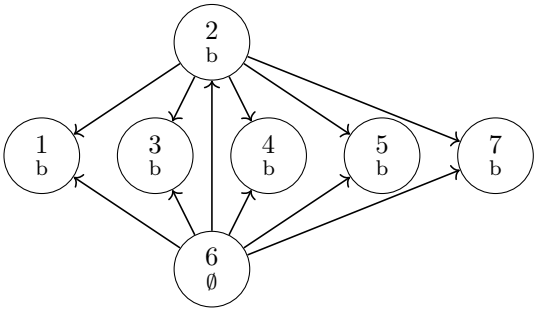
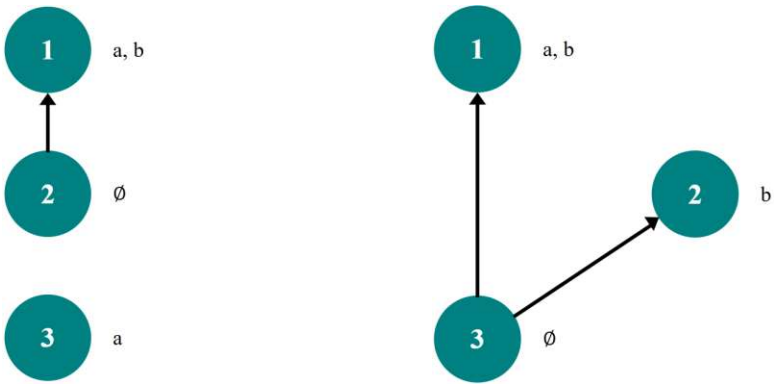


Figure 6.1: Example graph representation

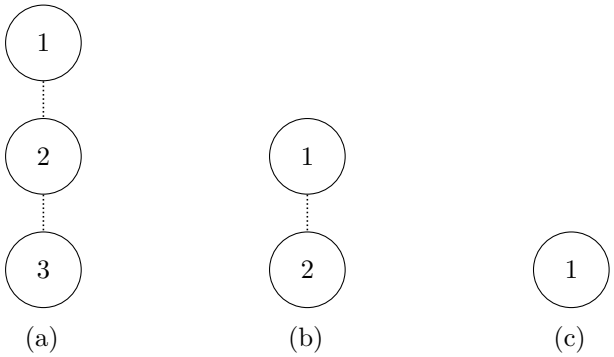
For larger models, and especially those with a dense preference relation, the graph becomes hard to read.. Up to three worlds in a model, we can represent the graphs with straight crossing-free edges. The layout allows us to position true variables beside their nodes, taking advantage of the space available for longer names. Depending on the preference relation we can display the graph in a clear drawing, where the nodes are arranged in a vertical column, or in a triangular orientation (see Figure 6.2).



(a) Linear graph (b) Triangular graph

Figure 6.2: Example graph UI with variables displayed next to node

The linear representation works for many such small models. We display nodes representing better worlds above the ones representing inferior ones. Apart from that, we try to keep the natural order (1, 2, 3) as much as possible.

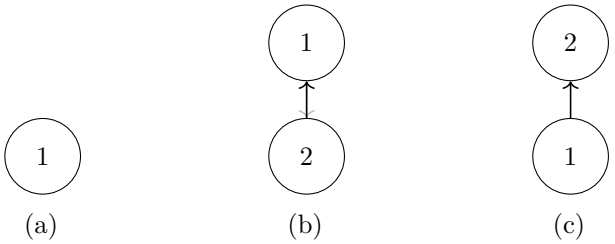


(a) (b) (c)

Figure 6.3: Line graphs in natural order

We go through the different cases depending on the number of worlds and the preference relation.
Models with a single worlds are the trivial case. For models with two worlds, we display node 2 on top, if $w_2 \succ w_1$, and node 1 on top otherwise.

The different cases for models with three worlds depend on the number of connections. There is a connection between two nodes i and j, if there is at least one arrow between



(a) (b) (c)

Figure 6.4: Graphs for models with one or two worlds

the two nodes, i.e. if $w_i \succ w_j$ or $w_j \succ w_i$. We talk about an undirected connection if there is a bidirectional arrow between two nodes (both relations hold) and about a directed connection from node i to node j if there is an arrow only from i to j (only one of both relations is true). A directed connection from node i to node j, is *in order* when $i < j$, and *against order* otherwise.

If the preference relation is empty, we display the nodes in the natural order (see Figure 6.5).

(a)

Figure 6.5: Graph for models with three worlds and no connections

Next we cover the cases with a single connection. For undirected and directed connections in order, we have the order $[1, 3, 2]$ if the connection is between node 1 and node 3. Otherwise, we can use the natural order, as displayed in Figure 6.6.

If the single connection is a directed order against order from node i to node j, we move j in front of i in the natural order. This orders result in the graphs shown in Figure 6.7.

The linear structure of nodes does not fit for some of the following described cases. For these cases we apply a triangular positioning of the nodes, with their natural order shown in Figure 6.8.

For models with three nodes and two connections, we start by determining the central node, i.e. the node with a connection with the two other nodes, which we indicate by bold lines. If both connections are undirected we can use the natural order. The line structure

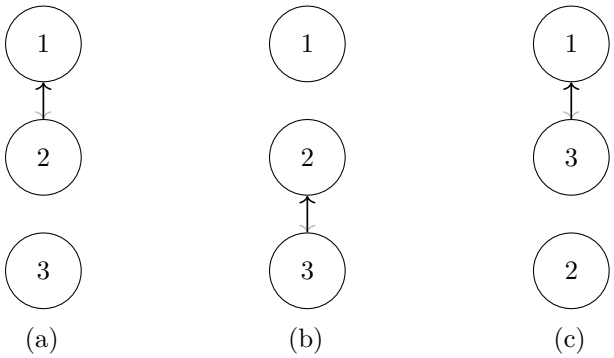


Figure 6.6: Graphs for models with three worlds and **one undirected connection or directed** connection **against order**

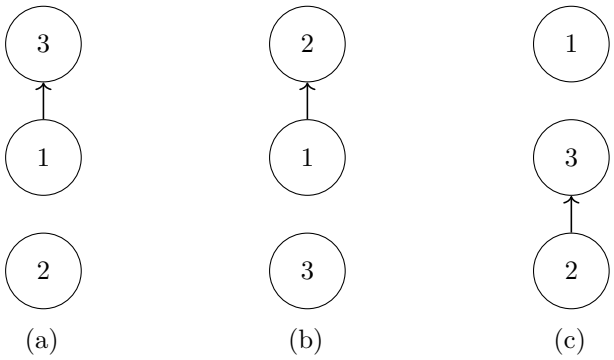


Figure 6.7: Graphs for models with three worlds and **one directed** connection **against order**

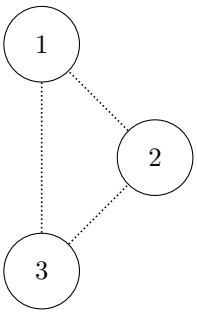


Figure 6.8: Triangular graph in natural order

can be used if node 2 is the central one, otherwise we use the triangular structure (see Figure 6.9).

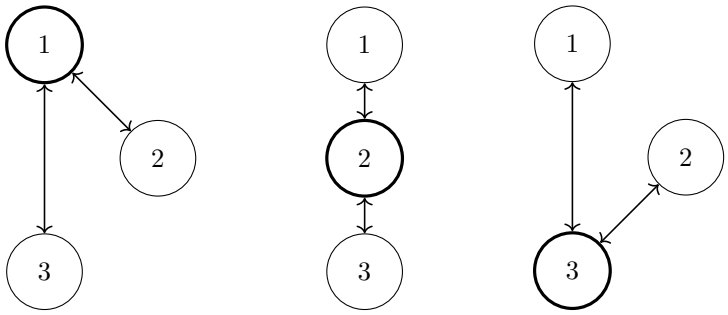


Figure 6.9: Graphs for models with three worlds and **two undirected** connections

We have three cases for models with two directed connections, depending on the number of in- and outgoing arrows in the central node. If one arrow is ingoing and one is outgoing, the order of the nodes is fully determined. The central node is the first in the order if it has two ingoing arrows, and on the bottom if it has two outgoing arrows. To complete the order we sort the remaining two nodes ascending by their numbers.

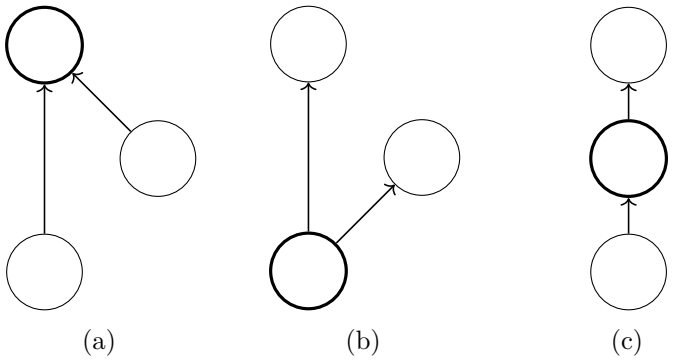


Figure 6.10: Graphs for models with three worlds and **two directed** connections

For the case of one undirected and one directed connection, we differentiate if the arrow of the directed connection is ingoing or outgoing with respect to the central node. When it is outgoing we obtain the structures displayed in Figure 6.11, depending on if node 1 is

(a) not part of the directed connection,

(b) the central node,

(c) the node the directed connection is ingoing.

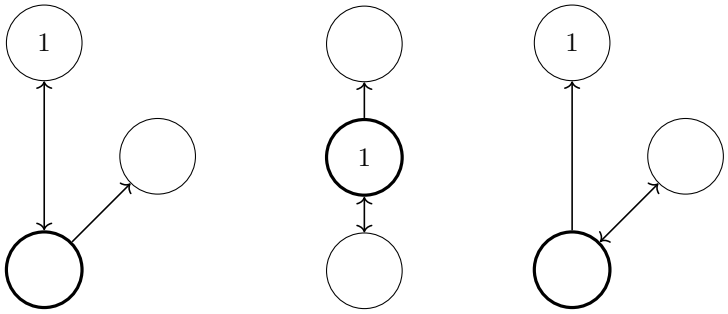


Figure 6.11: Graphs for models with three worlds and one undirected, one directed connection - arrow of directed connection **outgoing** from the central node

For to the cases where the arrow of the directed connection is ingoing to the central node, we use the same distinction and get the graph drawings of Figure 6.12, where in (b) exactly one of both arrows is bidirectional.

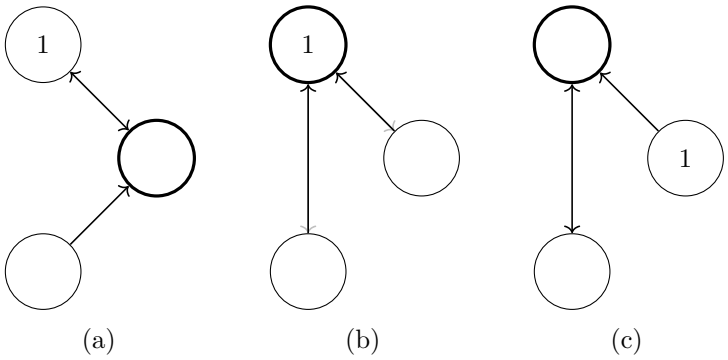


Figure 6.12: Graphs for models with three worlds and one undirected, one directed connection - arrow of directed connection **ingoing** in the central node

All graphs with three nodes and three connections are displayed in the triangle structure. We have cases depending on the number of directed connections.

If all connections are undirected we can use the natural order, as shown in Figure 6.13.

For models with a single direction connection, we change the natural order depending on this connection in the same way as described for models with three nodes and a single directed connection against the order. This results in the graphs presented in Figure 6.14, where the ones with directed connections in order trivially keep the natural order (a, b, c).

When two out of three connections are directed, we first determine the central node, as we did in for cases with two connections. As for models with two directed connections (without an additional undirected one), we have three analogues cases depending on the number of in- and outgoing arrows of the central node.

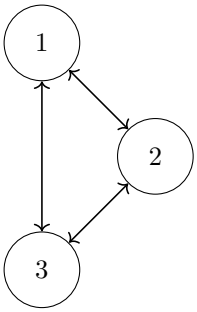


Figure 6.13: Graphs for models with three worlds and **three undirected** connections

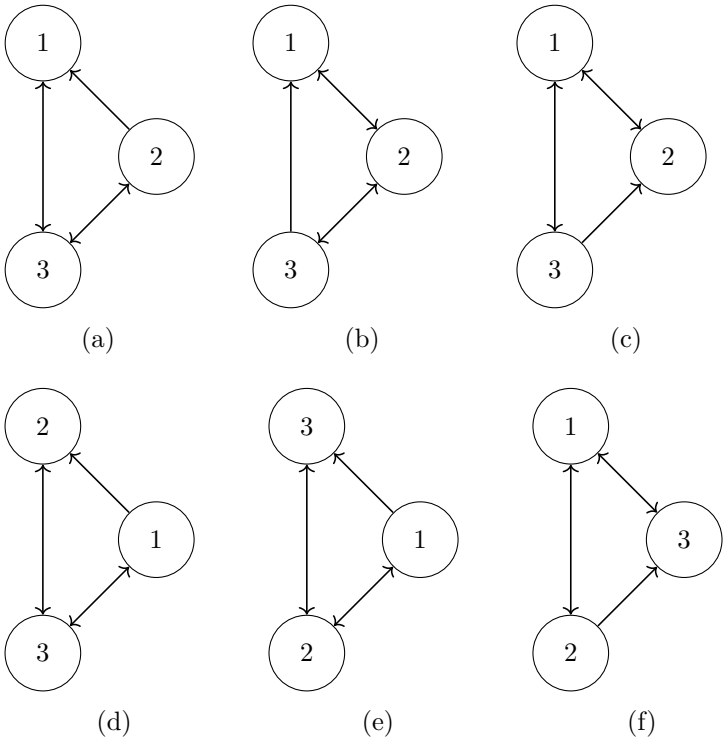
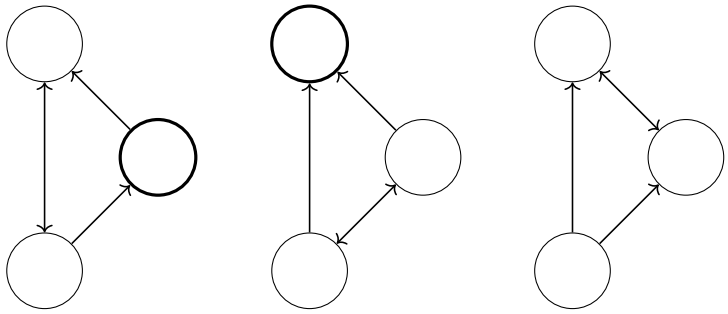


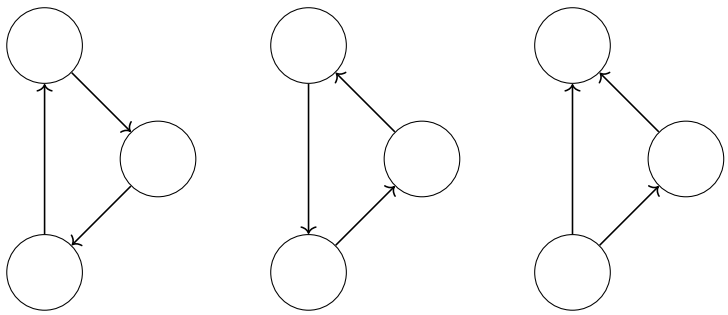
Figure 6.14: Graphs for models with three worlds and three connections - **one** of them **directed**



(a) (b) (c)

Figure 6.15: Graphs for models with three worlds and three connections - **two** of them **directed**

If a model has three directed connections, we either, if each node has one ingoing and one outgoing arrow, have a clockwise (see Figure 6.16a) or a counter-clockwise cycle (see Figure 6.16b) or a fully determined ordering of the nodes (see Figure 6.16c).



(a) (b) (c)

Figure 6.16: Graphs for models with three worlds and **three directed** connections

6.3 Matrix

A matrix can be used as a clear way to represent the preference relation over the set of worlds. We number the rows and columns by the numbers of the worlds. There is a mark in column i and row j, if and only if world w_i is preferred over world w_j, i.e. $w_i \succeq w_j$. The true variables of each world can directly be written next to the row of the world. Figure 6.17 shows the matrix for a countermodel of the formula $z \vee y \wedge \bigcirc(\ w \mid v) \vee \neg u \wedge \Box s$ with the size bound as a number of worlds.

To improve the readability of the matrix representation, especially for bigger models, by an interactive user interface. When moving the cursor on a mark, we highlight the corresponding world numbers, as shown in Figure 6.18. Therefore we use different colours to indicate better and inferior ones. The colours are also used to highlight the true

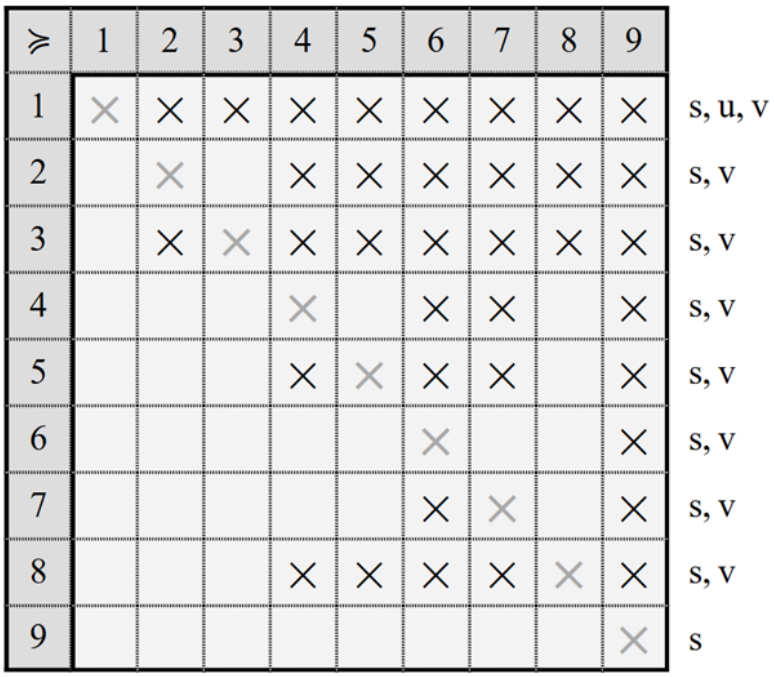


Figure 6.17: Example matrix interface

variables of the corresponding worlds. When a mark in the diagonal of the matrix is hovered the true variables of the world are marked grey (see Figure 6.19).

Similar we also highlight worlds for cursor movement on a world number. The numbers of all inferior worlds for a number of a row (see Figure 6.20), and of all better worlds for a number of a column (see Figure 6.21).

The most suitable representation depends on the size of the countermodel. For models with up to three worlds we display the directed graph giving the complete information. The matrix representation is used for the models bigger than those. Since the text representation of the preference relation is useful to quickly see the best or most inferior world of a model, we display it also for these bigger models.

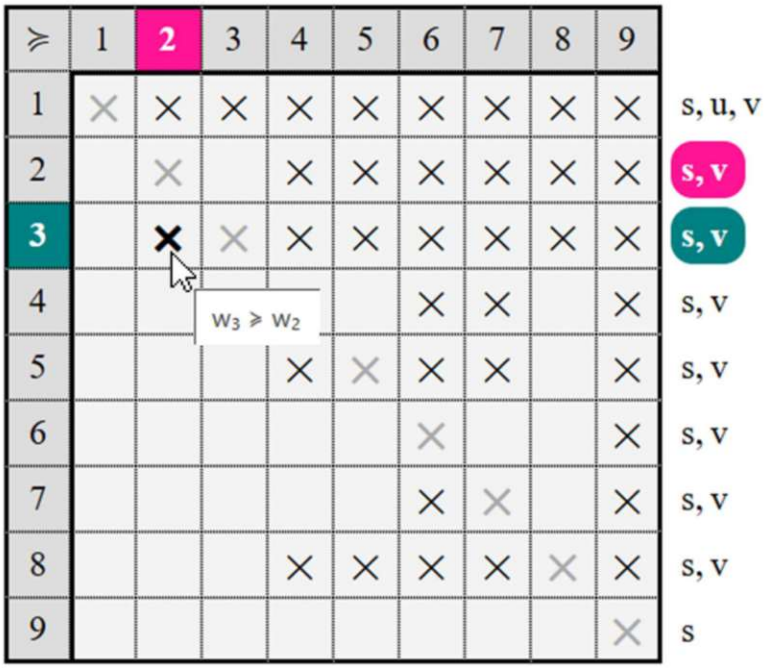


Figure 6.18: Example matrix interface with cursor on the marker

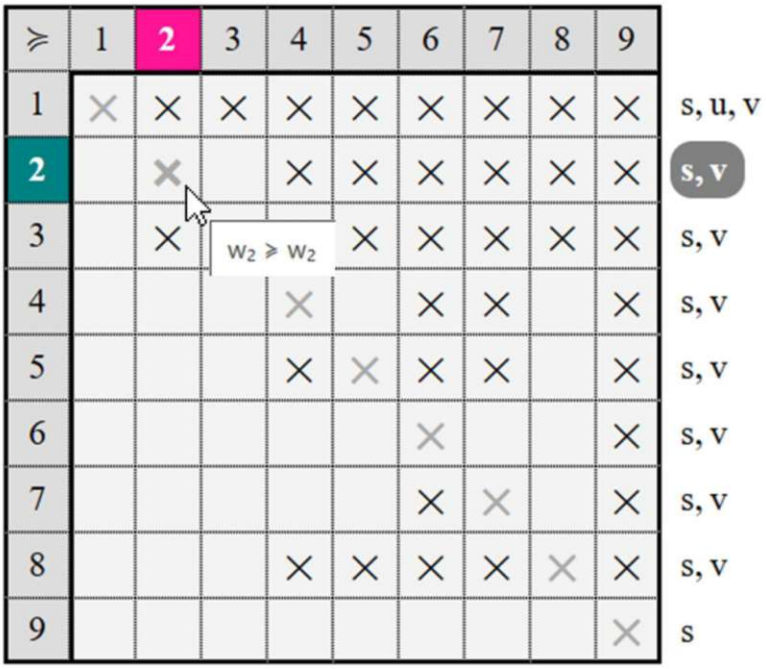


Figure 6.19: Example matrix interface with cursor on a diagonal marker

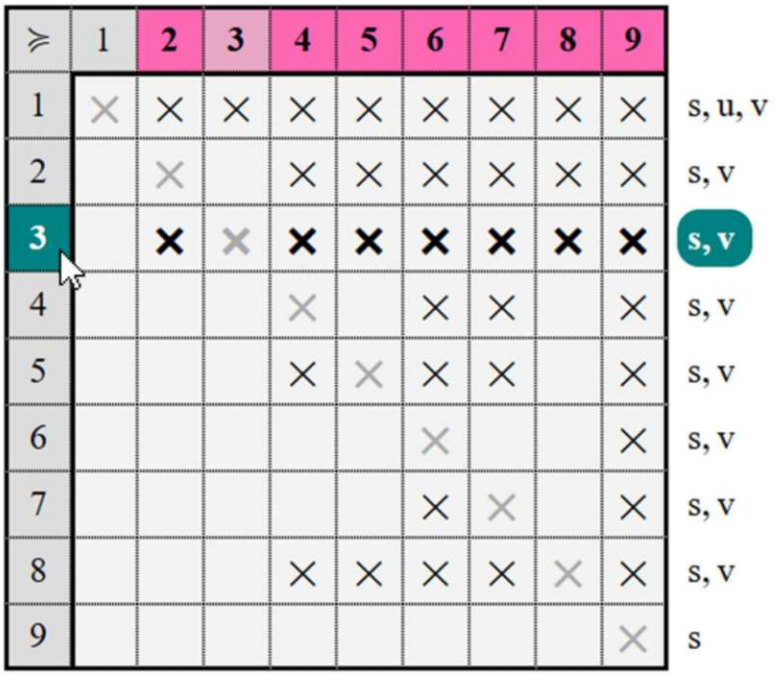


Figure 6.20: Example matrix interface with a highlighted row

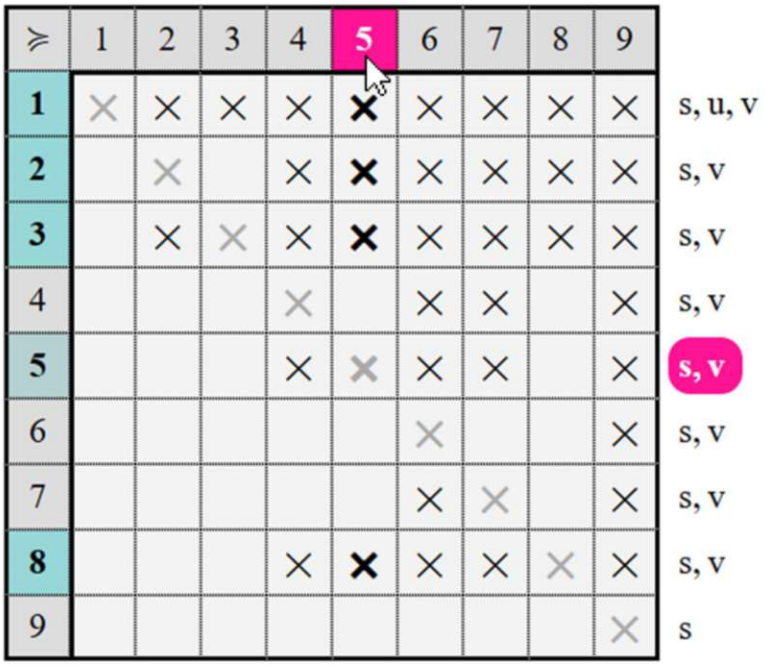


Figure 6.21: Example matrix interface with a highlighted column

CHAPTER 7

Implementation of Deontic Paradoxes

In this chapter, the automated reasoning tool is applied to a selection of well-known paradoxes in deontic logic. By paradoxes we mean problematic or counterintuitive patterns of reasoning that arise when formalizing normative concepts. For each paradox we give a formalization in SDL (with SDL formulas displayed in blue) and show how the paradox manifests in this logic. By formalizing the statements in Åqvist's systems, we can test their applicability across different logics, and illustrate how obligations behave when modeled in preference-based frameworks.

A classical contrary-to-duty paradox is given by Chisholm [Chi63].

Paradox 1 (Chisholm's Paradox)**.** We are given the following normative and factual statements:

1. The man ought to help his neighbours.
2. If he helps, he ought to tell them.
3. If he does not help, he ought not to tell them.
4. He does not help his neighbours.

We can formalize the paradox in SDL using the following formulas.

1. $\bigcirc(h)$
2. $\bigcirc(h \to t)$
3. $\neg h \to \bigcirc(\neg t)$

4. $\neg h$

In SDL, premises (2) and (3) must be encoded differently. Premise (2) is treated as an obligation of an implication, i.e., $\bigcirc(h \to t)$, whereas premise (3) is treated as an implication whose consequent is an obligation, i.e., $\neg h \to \bigcirc(\neg t)$. This asymmetry is standard in the literature (cf. [Chi63]) and causes the paradox to occur in SDL.

From (2), by axiom K of SDL, we get $\bigcirc(h \to t) \ \to \ (\bigcirc h \to \bigcirc t)$.

By (2) and modus ponens we get $(\bigcirc h \to \bigcirc t)$, and with (1) we infer $\bigcirc t$.

From (3) and (4), by modus ponens, we infer $\bigcirc \neg t$.

Hence, $\bigcirc t$ and $\bigcirc \neg t$ hold simultaneously, yielding a contradiction in SDL.

The formalization of the statements in Åqvist's logics is given by the following formulas.

1. $\bigcirc(h \mid \top)$
2. $\bigcirc(t \mid h)$
3. $\bigcirc(\neg t \mid \neg h)$
4. $\Box \neg h$

To analyse Chisholm's scenario with our tool, we check if the conjunction of the four statements is satisfiable.

$$\varphi = \bigcirc(h \mid \top) \wedge \bigcirc(t \mid h) \wedge \bigcirc(\neg t \mid \neg h) \wedge \Box \neg h$$

To test satisfiability, we provide the tool with $\neg\varphi$. The tool then checks whether this negation is valid. From this result, we can determine whether φ itself is satisfiable.

The tool outputs a model satisfying φ in logic **E**, consisting of two worlds, where h and t hold in the better world, and no propositional variable holds in the inferior one. Formally, the model is $M = \langle W, \succeq, \mathbb{V} \rangle$, with

- $W = \{w_1, \ w_2, \}$,
- $\succeq \ = \{(w_2, w_1)\}$,
- $\mathbb{V}(w_1) = \emptyset \ \mathbb{V}(w_2) = \{h, \ t\}$.

Also Parent and Benzmüller analysed this paradox in [BFP18] using their approach based on Isabelle/HOL. The only difference between the model they found and ours is the preference relation, which in their case is empty.

The well-known Gentle Murder paradox highlights that standard deontic logic allows deriving that killing is obligatory, despite the initial obligation not to kill [For84].

Paradox 2 (Gentle Murder)**.** Consider the following norms and facts:

1. It is obligatory not to kill.
2. If one kills, one ought to kill gently.
3. Killing gently entails killing.
4. Killing occurs.

In SDL the statements can be expressed by the following formulas.

1. $\bigcirc(\neg k)$
2. $k \to \bigcirc(g)$
3. $g \to k$
4. k

From (2) and (4), it follows: $\bigcirc(g)$

From (3) and $\bigcirc(g)$, we derive: $\bigcirc(k)$

This contradicts (1), yielding a paradox in standard deontic logic.

In Åqvist's logics the statements can be expressed by the following formulas, and the obligation to kill by $\bigcirc(k \mid \top)$.

1. $\bigcirc(\neg k \mid \top)$
2. $\bigcirc(g \mid k)$
3. $\Box(g \to k)$
4. k

To illustrate the Gentle Murder paradox, we test the validity of the following implication.

$$\Big(\bigcirc(\neg k \mid \top) \wedge \bigcirc(g \mid k) \wedge \Box(g \to k) \wedge k\Big) \to \bigcirc(k \mid \top)$$

We input the implication into the tool, which for logic **E** returns the countermodel $M = \langle W, \succeq, \mathbb{V}\rangle$ given by

- $W = \{w_1, w_2\}$,
- $w_2 \succcurlyeq w_1$,
- $\mathbb{V}(w_1) = \{g, k\}, \quad \mathbb{V}(w_2) = \emptyset$.

Here, k holds only in the less preferred world, so the best $\top$-worlds (i.e. the most preferred worlds overall) do not validate k. Therefore, $\bigcirc(k \mid \top)$ does not hold, and since the premises hold, the implication is shown to be invalid. This is the same countermodel for the paradox as the one found in [COP22] via hypersequents.

The so-called *repugnant conclusion* is a famous paradox in population ethics, originally formulated by Derek Parfit [Par84]. It illustrates counterintuitive implications of total utilitarian reasoning when comparing populations of different sizes and qualities of life. The presentation below closely follows the presentation by Parent and Benzmüller [PB24].

Paradox 3 (Parfit's Repugnant Conclusion)**.** Parfit characterizes the so-called *repugnant conclusion* as follows:

> "For any possible population of at least ten billion people, all with a very high quality of life, there must be some much larger imaginable population whose existence, if other things are equal, would be better even though its members have lives that are barely worth living" [Par84, Chapter 6]

In this paradox, total utilitarianism is applied by treating the overall value of a population as the sum of individual well-being. Consequently, a reduction in the quality of life for each individual can be outweighed by a sufficiently large increase in the number of people. As illustrated in Figure 7.1, this reasoning leads to what Parfit terms the *repugnant conclusion*: a very large population Z, whose members have lives barely worth living, would still be judged better than a much smaller population A, whose members enjoy excellent lives, simply because the total sum of well-being in Z exceeds that of A.

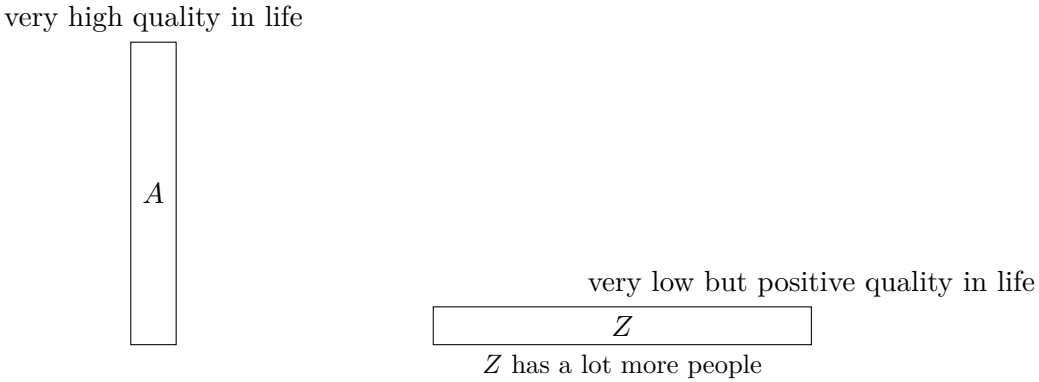


Figure 7.1: Repugnant conclusion, adapted from [PB24]

The repugnant conclusion can be understood as the outcome of repeatedly applying the reasoning behind the *mere addition* paradox. Figure 7.2 illustrates this paradox. In population A, every individual enjoys a very high quality of life. In population A^+, there exists a group of people identical in size and welfare to A, but in addition there are

further individuals whose lives are somewhat less good, though still worth living. This extension from A to A^+ is what Parfit calls a case of "mere addition". Population B contains the same number of individuals as A^+, and although their lives are all worth living at a welfare level slightly above the average of A^+, their quality of life is still below that of population A. By iterating this structure (adding B^+ and C, then C^+ and so on), we are eventually led to the repugnant conclusion: a very large population Z in which all individuals have lives of very low positive quality.

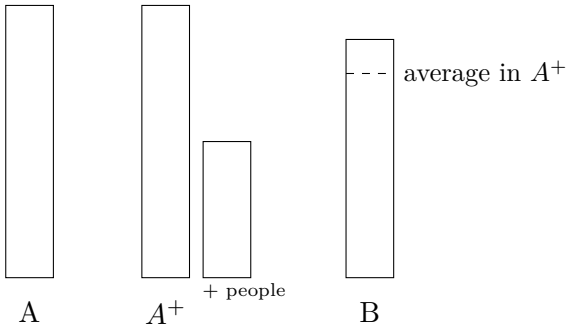


Figure 7.2: Mere addition paradox, adapted from [PB24]

The following assumptions are usually taken to be plausible:

1. $A > B$. Population A is strictly better than B. Otherwise, by parity of reasoning in the iterative construction (B^+, C, C^+, ...), one would have to conclude that A is not better than Z.

2. $A^+ \geq A$. Population A^+ is at least as good as A, since adding extra lives that are still worth living cannot make the situation worse.

3. $B > A^+$. Population B is strictly better than A^+, as both have the same size, but in B everyone has higher welfare, and welfare is distributed equally.

The relations $\geq$ and $>$ used in (1)–(3) are defined over propositional formulas. As shown in Table 7.1, the expression $\varphi \geq \psi$ is taken as shorthand for $\neg\bigcirc(\neg\varphi \mid \varphi \vee \psi)$. The strict preference $\varphi > \psi$ is then obtained by combining $\varphi \geq \psi$ with the condition that $\psi \not\geq \varphi$.

Definiendum	**Definiens**
$\varphi \geq \psi$	$\neg\bigcirc(\neg\varphi \mid \varphi \vee \psi)$
$\varphi > \psi$	$\neg\bigcirc(\neg\varphi \mid \varphi \vee \psi) \wedge \bigcirc(\neg\psi \mid \varphi \vee \psi)$

Table 7.1: Preference on formulas, adapted from [PB24].

Formalizing the above assumptions in Åqvist's preference-based logics and combining them, we obtain the following conjunction:

$$\begin{aligned}\psi = \neg\bigcirc(\neg A \mid A \vee B) \wedge \bigcirc(\neg B \mid A \vee B) \\ \wedge \neg\bigcirc(\neg A^{+} \mid A^{+} \vee A) \\ \wedge \neg\bigcirc(\neg B \mid B \vee A^{+}) \wedge \bigcirc(\neg A^{+} \mid B \vee A^{+}).\end{aligned}$$

We proceed as before by testing satisfiability via the negated formula $\neg\psi$. The tool checks whether this negation is valid, which allows us to determine the satisfiability of ψ.

For logic **E** we find the following model $M = \langle W, \succeq, \mathbb{V}\rangle$, satisfying ψ, with

- $W = \{w_1,\ w_2,\ w_3\}$,
- $\succeq = \{(w_1, w_3),\ (w_3, w_2),\ (w_2, w_1)\}$,
- $\mathbb{V}(w_1) = \{B\},\quad \mathbb{V}(w_2) = \{A\},\quad \mathbb{V}(w_3) = \{A^{+}\}$.

In Figure 7.3, we compare the model generated by our tool with the one from [PB24]. The latter contains one additional preference relation and one world satisfies an extra variable.

(a) Model in [PB24] (b) Model found by our tool

Figure 7.3: Models for repugnant conclusion

The repugnant conclusion can be avoided by rejecting the assumption of transitivity of the preference relation, as argued by, e.g. Temkin [Tem87]. Accordingly, for logics **F+(CM)** and **G**, our tool finds that $\neg\psi$ is valid. This implies that ψ is unsatisfiable in these logics.

Overall, the paradox of the repugnant conclusion is avoided in logics that do not require transitivity of the preference relation (**E**, **F**), while it still arises in the logics that enforce transitivity (**F+(CM)**, **G**).

CHAPTER 8

Optimisations and Benchmarking

This chapter explains the generation of a test set of formulas for runtime comparisons and discusses optimisations for the automated reasoning tool. Before introducing the optimisations, we briefly outline the software architecture of the tool.

8.1 System Architecture and Formula Class Hierarchy

The automated reasoning tool is implemented as a web application. It consists of a Vue frontend, which communicates with the Java backend, which in turn uses the Java Z3 API to interact with the Z3 solver (see Figure 8.1).

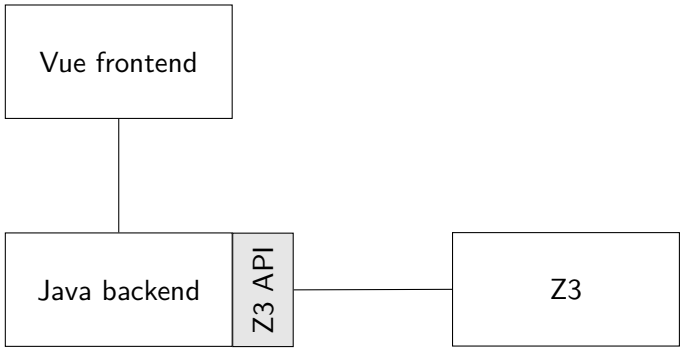


Figure 8.1: Architecture overview of the frontend–backend–solver interaction.

We examine the central class structure used to represent formulas internally, which is illustrated in the UML diagram in Figure 8.2. The methods *display* and *represent* are used to obtain a string representation of a formula using standard logical symbols and input-specific symbols, respectively. The latter is used to define variable names for the Z3 encoding.

The *equals*-method of an *AssociativeOperation* compares the complete list of arguments returned by the method *getArgsDeep*, without considering the order of them. These list is also important for the simplification process (implemented in the methods *simplify*) of these operation types, as discussed in Section 4.3.

The propositional encoding for Z3 of a formula object is returned by the method *encode*. Therefore the class *Z3Context* provides a central interface for encoding formulas into *BoolExpr* objects for the Z3 API and managing variables used in the encoding. It provides utility methods for constructing logical expressions, including *mkEq*, *mkNot*, *mkAnd*, *mkOr*, and *mkImpl*. Listing 8.1 shows an example of how these utility methods are applied in formula classes, specifically within the *encode* method of *Conjunction*.

Listing 8.1: Encoding method for `Conjunction`

```
public BoolExpr encode(int world, int sizeBound) {
    return mkEq(
        getVarV(this, world),
        mkAnd(
            getVarV(leftOperand, world),
            getVarV(rightOperand, world)
        )
    );
}
```

8.2 Test set generation

To compare the performance of different methods and implementations, we generate a test set of formulas.

The generation is done recursively based on the maximum depth of the formulas. The recursion base case consists of the following formulas with a depth of 0, the Boolean constants $\top$, $\bot$ and a single variable.

$$[\,\top,\ \bot,\ v_0\,]$$

Formulas of depth d with $d > 0$ are generated recursively using the formulas of a maximum depth of $d - 1$ as arguments. Note that this for instance when we generate formulas of depth 2 includes formulas of depth 0 as arguments. Therefore formulas of depth 2 include formulas like $v_0 \wedge \bigcirc(\top \mid v_0)$.

We take a look at the formulas with depth 1 we generate. For the unary operators we obtain the following formulas with atomic formulas of depth 0 as arguments.

$$[\,\neg\top,\ \neg\bot,\ \neg v_0,\ \Box\top,\ \Box\bot,\ \Box v_0\,]$$

Through combining atomic expressions using the binary operator obligation we obtain the following formulas.

$$[\,O(\top \mid \top),\ O(\top \mid \bot),\ O(\top \mid v_0)$$

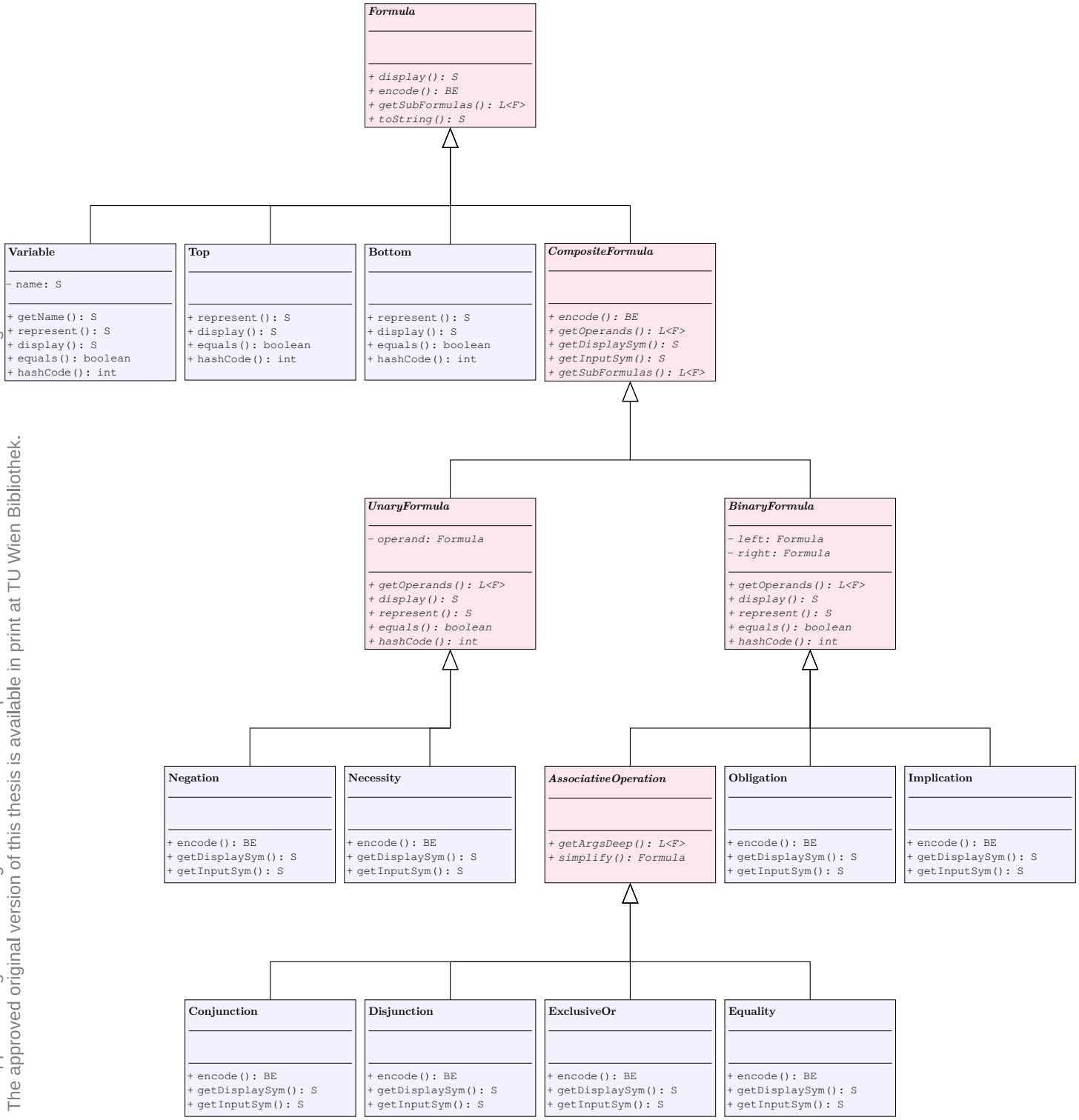


Figure 8.2: UML class diagram of the formula hierarchy.
Type abbreviations: `S` = `String`, `L<F>` = `List<Formula>`, `BE` = `BoolExpr`.

$$O(\bot \mid \top),\ O(\bot \mid \bot),\ O(\bot \mid v_0)$$
$$O(v_0 \mid \top),\ O(v_0 \mid \bot),\ O(v_0 \mid v_0)\,]$$

To combine different variables with each other we add additional variables to the test set of the one smaller depth. The number of variables we have to add depends on the depth. At depth d, up to 2^d different variables can be combined, see Table 8.1.

Table 8.1: Variables for generating formulas of a given depth

depth	**variables**	**example formula**
0	$[v_0]$	v_0
1	$[v_0, v_1]$	$v_0 \wedge v_1$
2	$[v_0, v_1, v_2, v_3]$	$(v_0 \wedge v_1) \vee O(v_2 \mid v_3)$

At depth 1, we can generate formulas combining two different variables, we obtain the following ones for obligation.

$$[\,\bigcirc(v_0 \mid v_0),\ \bigcirc(v_0 \mid v_1),\ \bigcirc(v_1 \mid v_0),\ \bigcirc(v_1 \mid v_1)\,]$$

Note that the formulas $\bigcirc(v_0 \mid v_1)$ and $\bigcirc(v_1 \mid v_0)$ are equivalent modulo variable renaming. We only add one of such formulas to the test set.

$$[\,\bigcirc(v_0 \mid v_0),\ \bigcirc(v_0 \mid v_1)\,]$$

We check whether two formulas are equivalent modulo variable renaming, by renaming variables to sequentially numbered names starting with v_0. The variables in a formula get renamed in the order of their first occurrence. For example renaming the formula $x \wedge y \wedge x$ we apply the following renamings, since x occurs before y.

$$x \mapsto v_0$$

$$y \mapsto v_1$$

These result in the following formula renaming.

$$x \wedge y \wedge x \rightsquigarrow v_0 \wedge v_1 \wedge v_0$$

For instance all obligations with two different variables, like $\bigcirc(b \mid a)$, $\bigcirc(v_0 \mid v_1)$ and $\bigcirc(v_1 \mid v_0)$, as arguments get renamed to $\bigcirc(v_0 \mid v_1)$.

We often have to check if a formula equivalent modulo variable renaming is already in the test set while generating formulas. To efficiently do that, we for each depth maintain a set of formula strings obtained from the formulas through renaming variables.

Not needed parenthesis are skipped in the strings, such that we only add one of formulas equivalent due to associativity. For example the formulas $(v_0 \wedge v_1) \wedge v_2$ and $v_0 \wedge (v_1 \wedge v_2)$

both are represented by the string "$v_0 \wedge v_1 \wedge v_2$" and we only add one of them to the test set.

We avoid generating commutatively equivalent formulas, such as $v_0 \wedge \top$ and $\top \wedge v_0$ for conjunction. The following conjunctions of depth 1 are generated.

$$[\, \top \wedge \top,\ \top \wedge \bot,\ \top \wedge v_0,\ \bot \wedge \bot,\ \bot \wedge v_0,\ v_0 \wedge v_0,\ v_0 \wedge v_1 \,]$$

For a maximum depth of 1 and excluding the implication operator, we obtain the following 47 formulas.

$$\begin{aligned}
&[\, \top,\ \bot,\ v_0,\\
&\ \neg\top,\ \neg\bot,\ \neg v_0,\\
&\ \Box\top,\ \Box\bot,\ \Box v_0,\\
&\ \bigcirc(\top \mid \top),\ \bigcirc(\top \mid \bot),\ \bigcirc(\top \mid v_0),\\
&\ \bigcirc(\bot \mid \top),\ \bigcirc(\bot \mid \bot),\ \bigcirc(\bot \mid v_0),\\
&\ \bigcirc(v_0 \mid \top),\ \bigcirc(v_0 \mid \bot),\ \bigcirc(v_0 \mid v_0),\ \bigcirc(v_0 \mid v_1),\\
&\ \top \wedge \top,\ \top \wedge \bot,\ \top \wedge v_0,\ \bot \wedge \bot,\ \bot \wedge v_0,\ v_0 \wedge v_0,\ v_0 \wedge v_1,\\
&\ \top \vee \top,\ \top \vee \bot,\ \top \vee v_0,\ \bot \vee \bot,\ \bot \vee v_0,\ v_0 \vee v_0,\ v_0 \vee v_1,\\
&\ \top \oplus \top,\ \top \oplus \bot,\ \top \oplus v_0,\ \bot \oplus \bot,\ \bot \oplus v_0,\ v_0 \oplus v_0,\ v_0 \oplus v_1,\\
&\ \top \leftrightarrow \top,\ \top \leftrightarrow \bot,\ \top \leftrightarrow v_0,\ \bot \leftrightarrow \bot,\ \bot \leftrightarrow v_0,\ v_0 \leftrightarrow v_0,\ v_0 \leftrightarrow v_1 \,]
\end{aligned}$$

The number of formulas grows rapidly as we increase the maximum depth. For a maximum depth of 2 we obtain a test set of 10 460 formulas, which we use in the following section for comparison of different optimisation levels.

8.3 Time Measurements for different Optimisation Levels

We compare the runtime of the tool on the generated test set. Time measurements of the base implementation serve as a reference point. Table 8.2 shows the average runtime for simplifying the formula, computing the size bound $N(\varphi)$, Z3 encoding, checking validity, finding a minimum model (for all and only invalid formulas), and the total runtime for the test set described in the previous section.

In the class *Z3Context* we use memorisation, by caching already created *BoolExpr*-objects, to speed up the encoding, which achieves a speedup of 1.15.

Our main optimisation strategy is the formula simplification discussed in Chapter 4. This reduces the formula size (i.e. the number of symbols) by 77% and the size bound for the required number of worlds in a countermodel by 52% (see Table 8.3). For the total runtime of the tool this yields a speedup of 1.71 compared to the version using memorisation.

	simplify	N(φ)	encode	validity	min.(all)	min.(inv.)	total
base	–	0.01	13.64	24.17	17.13	28.81	54.95
memoriz.	–	0.01	12.03	23.60	12.35	20.76	47.99
simplify	0.06	0.01	6.45	15.83	5.81	16.87	28.10

Table 8.2: Runtimes in milliseconds

	formula size	size bound
without simplification	98.36	8.35
with simplification	22.46	4.01

Table 8.3: Reduction of formula size and size bound

CHAPTER 9

Comparison to Existing Tool

We compare the introduced tool with the existing automated approach for Åqvist's logics by Parent and Benzmüller [PB22] using Isabelle/HOL. Their approach uses higher-order logic to formalize and verify deontic correspondences within Åqvist's framework. Isabelle is an interactive theorem prover that supports a variety of logical formalisms. In particular, Isabelle/HOL is based on classical higher-order logic, making it suitable for the formalization of complex logical systems. Their approach embeds Åqvist's logics into Isabelle's meta-logic, allowing properties and correspondences to be expressed, and automatically verified using higher-order reasoning. This method emphasizes formal proof construction and logical soundness.

9.1 Feature Overview and Comparison

We compare our tool to the Isabelle/HOL implementation in terms of:

- **Supported logics:** Our tool supports all of Åqvist's logics (**E**, **F**, **F+(CM)**, **G**), while theirs is restricted to **E**.
- **Performance:** Our tool uses finite model construction via propositional encoding and SAT-based solving, resulting in significantly faster runtimes.
- **Simplification:** Additionally, our tool is capable of simplifying the input formula, which improves readability and also computational performance.
- **Input format:** Our tool accepts straightforward syntax requiring no prior knowledge. In contrast, Isabelle/HOL requires familiarity with formal proof environments and function syntax.

- **Model readability:** Our tool provides countermodels in three human-readable formats (graphical, matrix, and text), compared to Isabelle's internal Nitpick output.

- **Extensibility:** Our tool is designed to be modular, with the potential for extending support to new operators and optimizations (see Chapter 8 and Chapter 10).

Table 9.1: Feature comparison between our tool and Isabelle/HOL

Feature	**Our Tool**	**Parent & Benzmüller**
Supported logics	**E**, **F**, **F+(CM)**, **G**	**E**
Model generation	Yes (minimal models)	Yes (via Nitpick)
Model format	Graph, Matrix, Text	Isabelle output
Input simplicity	Intuitive syntax	Isabelle/HOL code
Formula simplification	Yes	No
Runtime performance	Fast (SAT-based)	Slower (HOL-based)
Extensibility	Easy to extend	Harder (formal proofs)
Usability for non-experts	High	Low

9.2 Example Model Comparison

To illustrate differences in model representation, we consider the following model $M = \langle W,\ \succeq,\ \mathbb{V}\rangle$, with

- $W = \{w_1,\ w_2,\ w_3\}$,

- $w_2 \succcurlyeq w_1,\ w_3 \succcurlyeq w_1,\ w_3 \succcurlyeq w_2$,

- $\mathbb{V}(w_1) = \{c\},\ \ \mathbb{V}(w_2) = \{a,\ b\},\ \mathbb{V}(w_3) = \{b\}$.

Isabelle provides countermodels in the following Nitpick format.

Free variables:

$$a = (\lambda x.\ \_)(i_1 := \text{False},\ i_2 := \text{True},\ i_3 := \text{False})$$
$$b = (\lambda x.\ \_)(i_1 := \text{False},\ i_2 := \text{True},\ i_3 := \text{True})$$
$$c = (\lambda x.\ \_)(i_1 := \text{True},\ i_2 := \text{False},\ i_3 := \text{False})$$

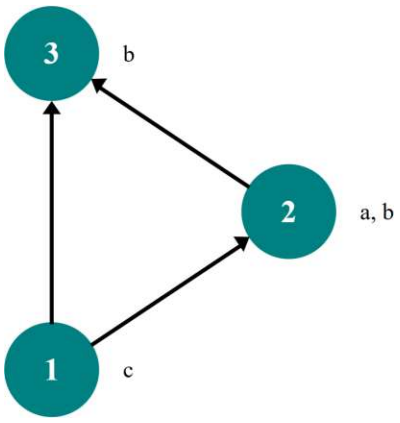
Skolem constants:

$$v = i_3$$
$$v = i_3$$
$$v = i_2$$
$$x = i_2$$

Constants:

$$aw = i_1$$
$$(r) = (\lambda x.\ \_)$$
$$((i_1, i_1) := \text{True},\ (i_1, i_2) := \text{False},\ (i_1, i_3) := \text{False},$$
$$(i_2, i_1) := \text{True},\ (i_2, i_2) := \text{True},\ (i_2, i_3) := \text{False},$$
$$(i_3, i_1) := \text{True},\ (i_3, i_2) := \text{True},\ (i_3, i_3) := \text{True})$$

Our tool displays the same model in three different representations discussed in Chapter 6, which can be seen in Figure 9.1.



(a) Graph

⩾	1	2	3	
1	×			c
2	×	×		a, b
3	×	×	×	b

(b) Matrix

$3 \geqslant 1, 2$
$2 \geqslant 1$

(c) Text

Figure 9.1: Model representations of our tool

9.3 Performance Evaluation

We combine the size bound method with the Isabelle approach to compare the runtimes. Our routine first checks validity, then finds a minimal countermodel. For the invalid formula $a \vee b \vee \bigcirc(a \mid b) \vee \Box\neg a \vee \Box\neg c \vee \Box\neg b$, we obtained a runtime of 8559 milliseconds using Isabelle and 338 milliseconds with our tool. This corresponds to a speedup of 25.29×, even without applicable simplifications.

We also tested the valid formula $(\bigcirc(b \mid a) \wedge \bigcirc(c \mid a)) \rightarrow \bigcirc(d \mid a)$, for which no simplification rule is applicable.

Here, we obtained a runtime of 2783 milliseconds with Isabelle/HOL and 1057 milliseconds with our tool, resulting in a speedup of 2.63.

The formula $\bigcirc(b \mid a) \wedge \bigcirc(c \mid a) \wedge \bigcirc(d \mid a)$ can be simplified, and we achieved a speedup of 2.97 with a runtime of 668 milliseconds compared to 1986 milliseconds using Isabelle.

In particular, for invalid formulas, our automated reasoning tool shows the greatest advantage.

9.4 Summary

Parent and Benzmüller's work has laid important groundwork for mechanizing Åqvist's logic **E** in a proof assistant. Our tool complements this effort with a lightweight, efficient, and user-friendly approach applicable to a broader range of deontic systems. Its speed, minimal countermodels, and input simplicity make it a strong candidate for further academic and practical applications in automated deontic reasoning.

CHAPTER 10

Conclusion

Åqvist's logics form a central framework in the field of deontic logic. To use them for reasoning (providing either proof or countermodel) [Roz24] introduced a semantic-based method, by developing small model constructions for these logics and encoding them into classical propositional formulas. In this thesis we use these encodings to develop an efficient automated reasoning tool that leverages SMT technologies through the usage of the SMT-solver Z3 of Microsoft to check validity and find a minimal countermodel for non-valid formulas.

After introducing preliminaries, including the propositional encoding, we explained the parsing of formulas, where we give error feedback for invalid inputs. We presented how formulas are simplified by applying transformation rules in a bottom-up manner. For the encoding for Z3 we gave formulas for the upper bound of the required number of worlds, explained the encoding for the supported operators, and the process of finding a minimal countermodel with respect to the number of worlds. To present the countermodels in a readable form we proposed three different representations using text only, as a directed graph, and as a matrix. We use these variants depending of the number of worlds in the countermodel.

As a case study, we used the tool to check whether deontic paradoxes are blocked by attempting to find countermodels. Finally, the runtimes were compared for different optimisations and the developed automated reasoning tool was compared to the approach by Parent and Benzmüller [PB22] using Isabelle/HOL. While their approach provides high assurance within a proof assistant, it is restricted to logic **E**, requires familiarity with Isabelle's formal syntax, and produces countermodels in a less accessible format. In contrast, our tool supports all of Åqvist's logics, offers intuitive input and multiple human-readable model representations, and achieved substantial performance improvements. In particular, for invalid formulas our method outperformed the Isabelle/HOL framework by more than a factor of 25, demonstrating its potential as a reliable and efficient tool for future academic and practical applications in automated deontic reasoning.

10.1 Future Work

While this work provides a solid foundation, future work can extend its scope in various directions.

Future work could focus on extending our framework to handle more complex paradoxes, including those found in legal reasoning, helping to gain a better understanding of conflicts and improve computational methods in law. Our tool could also be used to study examples from legal texts, capturing key aspects of obligations, permissions, and rules in practice.

Another direction is to explore deontic explanations—simple justifications of why certain obligations or permissions hold, as discussed in [CUP]. These explanations could help make normative reasoning in legal contexts more transparent and easier to understand, providing insights into why specific rules apply or why one action is preferred over another.

Although we currently support standard propositional connectives (such as conjunction, disjunction, negation, implication, equivalence, and exclusive or), as well as both deontic modalities used by Åqvist, the tool could be extended to accept additional deontic operators for prohibition (e.g., $F(\psi \mid \phi)$) and permission (e.g., $P(\psi \mid \phi)$). Such additions would allow for more natural and expressive representations of normative statements, particularly when negative obligations or explicitly permitted actions need to be formalized. It would also be possible to add fixed versions of the conditional operations in the form of $\bigcirc(\psi)$, $F(\psi)$ and $P(\psi)$, which are equivalent to $\bigcirc(\psi \mid \top)$, $F(\psi \mid \top)$, and $P(\psi \mid \top)$, respectively. The unary possibility operator $\Diamond$ (input e.g. as "<>") could also be directly encoded into a propositional formula using the following equivalence.

$$\Diamond\psi \;\Leftrightarrow\; \bigvee_{i=1}^{N(\varphi)} v_i^{\psi}$$

The tool currently returns a countermodel with the minimal possible number of worlds. It could also be extended to find countermodels minimal with respect to the density of the preference relation. That is, models where the number of preference pairs (w_i, w_j) is as small as possible. This could be done by multiple Z3 calls, after each one adding an assertion to exclude the last found model to find all different models, and identify one of the least complex one of them. To exclude a model $M = \langle W, \succeq, \mathbb{V}\rangle$ with $W = \{w_1, \ldots, w_n\}$ one could add the following assertion.

```
(assert (or
```

$\forall\, w_i, w_j \in W$ with $w \succeq v$ `(not p:i:j)`

```
))
```

Another promising direction for extension would be the extension of the formula simplification phase. Future enhancements could include context-aware simplifications

that consider surrounding formula structure. For instance one could detect redundant subformulas across larger scopes.

The computation of the size bound for the required number of worlds in a countermodel could be improved. One could determine for each subformula of the form $\Box\phi$ and of the form $\bigcirc(\psi \mid \phi)$, if it needs to get satisfied. That is deciding if it is in Ob^+ (resp. Box^+) or in Ob^- (resp. Box^-). The cardinalities of this sets could be used to compute a more precise size bound, by enabling the use on the exact formula for the number of falsifying worlds.

$$|\mathrm{Fal}(\varphi)| = 1 + |\mathrm{Fal}^{\Box}(\varphi, M)| + |\mathrm{Fal}^{\bigcirc}(\varphi, M)|$$

The graph representation for bigger countermodels could be examined.
Also, the text representation of transitive preference relations could be extended. The relation given by $w_1, \ldots, w_n \succeq v_1, \ldots, v_m$ could be extended in the following way, if for each x_i it holds that $\{w_1, \ldots, w_n\} \subseteq \mathit{sameRel}[x_i]$ and for each y_i we have $\{v_1, \ldots, v_m\} \subseteq \mathit{otherRel}[y_i]$ (compare Algorithm 6.1).

$$x_1, \ldots, x_k \;\succeq\; w_1, \ldots, w_n \;\succeq\; v_1, \ldots, v_m \;\succeq\; y_1, \ldots, y_l$$

The formula test sets introduced in Section 8.2 can serve as a foundation for creating more advanced benchmark sets, enabling better tool comparisons and systematic testing of optimisations.

Overview of Generative AI Tools Used

While working on this thesis, I used the AI tools *ChatGPT* and *Gemini* to find synonyms and alternative formulations, to check grammar and spelling, as well as to help generating LaTeX code for formulas and figures.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Åqv84] L. Åqvist. Deontic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic: Volume II*, pages 605–714. Springer, Dordrecht, 1984.

[BFP18] Christoph Benzmüller, Ali Farjami, and Xavier Parent. A dyadic deontic logic in hol. In *14th International Conference on Deontic Logic and Normative Systems (DEON 2018)*. College Publications, 2018.

[Bur81] John P Burgess. Quick completeness proofs for some logics of conditionals. *Notre Dame Journal of Formal Logic*, 22(1):76–84, 1981.

[CDF25] Agata Ciabattoni, Josephine Dik, and Elisa Freschi. Mīmāṃsā on 'better-not'permissions. *Journal of Philosophical Logic*, pages 1–39, 2025.

[Che80] Brian F Chellas. *Modal logic: an introduction*. Cambridge university press, 1980.

[Chi63] Roderick M Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24(2):33–36, 1963.

[CHS+23] Agata Ciabattoni, John F Horty, Marija Slavkovik, Leendert van der Torre, and Aleks Knoks. Normative reasoning for ai. 2023.

[COP22] Agata Ciabattoni, Nicola Olivetti, and Xavier Parent. Dyadic obligations: proofs and countermodels via hypersequents. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 54–71. Springer, 2022.

[COP+23] Agata Ciabattoni, Nicola Olivetti, Xavier Parent, Revantha Ramanayake, and Dmitry Rozplokhas. Analytic proof theory for aqvist's system f. In *Deontic Logic and Normative Systems: 16th International Conference, DEON 2023*, pages 79–98. College Publications, 2023.

[CPS21] Agata Ciabattoni, Xavier Parent, and Giovanni Sartor. A kelsenian deontic logic. In *Legal Knowledge and Information Systems*, pages 141–150. IOS Press, 2021.

[CT24] Agata Ciabattoni and Matteo Tesi. Sequents vs hypersequents for åqvist systems. In *International Joint Conference on Automated Reasoning*, pages 176–195. Springer, 2024.

[CUP] Agata Ciabattoni, Blaz Istenic Urh, and Xavier Parent. Deontic explanations in åqvist's systems. *17 DEON*, page 79.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[FH94] Nir Friedman and Joseph Y Halpern. On the complexity of conditional logics. In *Principles of Knowledge Representation and Reasoning*, pages 202–213. Elsevier, 1994.

[For84] James William Forrester. Gentle murder, or the adverbial samaritan. *The Journal of Philosophy*, 81(4):193–197, 1984.

[FP21] Elisa Freschi and Matteo Pascucci. Deontic concepts and their clash in mīmāṃsā: Towards an interpretation. *Theoria*, 87(3):659–703, 2021.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[GHP+21] Dov M. Gabbay, John F. Horty, Xavier Parent, Ron van der Meyden, and Leendert van der Torre, editors. *Handbook of Deontic Logic and Normative Systems, Volume 2*. College Publications, London, 2021.

[GP25] Guido Governatori and Monica Palmirani. Legal explanation in defeasible deontic logic via legalruleml. 2025.

[Han71] Bengt Hansson. An analysis of some deontic logics. In *Deontic logic: Introductory and systematic readings*, pages 121–147. Springer, 1971.

[KLM90] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial intelligence*, 44(1-2):167–207, 1990.

[Lew18] Clarence Irving Lewis. *A survey of symbolic logic*, volume 5. University of California press, 1918.

[Lew73a] David Lewis. Counterfactuals, blackwells, 1973.

[Lew73b] David K. Lewis. *Counterfactuals*. Harvard University Press, Cambridge, MA, 1973.

[M+10] Simon Marlow et al. Haskell 2010 language report. 2010.

[NBCG21] Emery A Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. A normative supervisor for reinforcement learning agents. In *CADE*, pages 565–576, 2021.

[NBCG22] Emery A Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. Enforcing ethical goals over reinforcement-learning policies. *Ethics and Information Technology*, 24(4):43, 2022.

[NO15] Sara Negri and Nicola Olivetti. A sequent calculus for preferential conditional logic based on neighbourhood semantics. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 115–134. Springer, 2015.

[Par84] D Parfit. Reasons and persons oxford, uk: Oxford univ. *Press [Google Scholar]*, 1984.

[Par14] Xavier Parent. Maximality vs. optimality in dyadic deontic logic. *Journal of Philosophical Logic*, 43:1101–1128, 2014.

[Par21] Xavier Parent. Preference semantics for hansson-type dyadic deontic logic: a survey of results. *Handbook of deontic logic and normative systems*, 2:7–70, 2021.

[PB22] Xavier Parent and Christoph Benzmüller. Automated verification of deontic correspondences in isabelle/hol-first results. In *ARQNL@ IJCAR*, pages 92–108, 2022.

[PB24] Xavier Parent and Christoph Benzmüller. Conditional normative reasoning as a fragment of hol. *Journal of Applied Non-Classical Logics*, 34(4):561–592, 2024.

[PvDT13] Xavier Parent and Leendert van Der Torre. Handbook of deontic logic and normative systems. *An intuitionistic basis for input/output logic.*, page p, 2013.

[Roz24] Dmitry Rozplokhas. Lego-like small model constructions for åqvist's logics. In Agata Ciabattoni, Dali Gabelaia, and Ivan Sedlár, editors, *Advances in Modal Logic*, pages 631–651. College Publications, 2024.

[SBA20] Colin Shea-Blymyer and Houssam Abbas. A deontic logic analysis of autonomous systems' safety. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2020.

[Sin22] Lavanya Singh. Automated kantian ethics: A faithful implementation. In *German Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 187–208. Springer, 2022.

[Tem87] Larry S Temkin. Intransitivity and the mere addition paradox. *Philosophy & Public Affairs*, pages 138–187, 1987.

[vBCF+23] Kees van Berkel, Agata Ciabattoni, Elisa Freschi, Francesca Gulisano, and Maya Olszewski. Deontic paradoxes in mīmāṃsā logics: There and back again. *Journal of Logic, Language and Information*, 32(1):19–62, 2023.

[VW51] Georg Henrik Von Wright. Deontic logic. *Mind*, 60(237):1–15, 1951.

[WM93] Roel J Wieringa and John-Jules Ch Meyer. Applications of deontic logic in computer science: A concise overview. *Deontic logic in computer science*, pages 17–40, 1993.