11 July 2024

Utrecht University

Universal Automated Theorem Prover for Non-classical Logics

Master Computing Science, Utrecht University

Bastiaan Haaksema (8176914)

Project facilitator: D.R.S. Ramanayake
First supervisor: P.R. North
Second supervisor: W.S. Swierstra

Table of Contents

Chapter 1 **1**

Introduction 1

Chapter 2 **4**

Preliminaries 4

2.1. Language of Logics 4

2.2. Hilbert-style Calculus for Classical Logic 4

2.3. Sequent Calculus for Classical Logic 6

2.4. Proof Search via Sequent Calculus 8

2.5. Calculi for Intuitionistic Logic 10

2.6. Superintuitionistic Logics 12

Chapter 3 **14**

Methodology 14

3.1. Cut-restricted Sequent Calculi 14

3.2. Logical Embeddings of Intermediate Logics 16

3.3. Refined Calculi for Intuitionistic Logic 17

3.4. Proof Search Optimizations 20

Chapter 4 **21**

Results 21

4.1. Proof Search Implementation 21

4.2. Experimental Evaluation 23

Chapter 5 **25**

Conclusion 25

Appendix A **30**

Implementation 30

Abstract

Non-classical logics are an adaptation of classical (boolean) logic that support nuanced forms of reasoning. This additional expressivity comes at a cost: non-classical logics tend to be more complex. This additional complexity is especially evident in the decision problem for the logic, namely when determining whether a given formula is provable in the logic. Automated theorem provers are implementations of decision procedures that can solve this problem using proof systems.

This thesis presents a generalized decision procedure for non-classical intermediate logics through an embedding into intuitionistic logic, enabled by the theory of cut-restriction that refines the standard proof system. The resulting automated theorem prover, SuperJ, is capable of proving theorems of a wide range of intermediate logics. The implementation is evaluated on a set of benchmark formulas and compared to the current state-of-the-art theorem prover intuitRIL. The results show that SuperJ is reasonably competitive with intuitRIL and could be further developed or used as a stepping stone to obtain a universal prover for non-classical logics.

Chapter 1

Introduction

Automated theorem proving is a subfield of automated reasoning, concerned with the development of algorithms for logical reasoning. It is important in computer science, with applications in software and hardware verification, programming languages and artificial intelligence. Interest in this topic remains strong ever since its inception, as evidenced by regularly held conferences, e.g. collectively as the International Joint Conference on Automated Reasoning (IJCAR) [1]. The main task of automated theorem provers is to show if some statement is a logical consequence of a particular logic represented by a set of axioms and rules. Since this is called the decision problem of logics, automated theorem provers are said to be decision procedure implementations. Often, the target logic for these procedures is classical propositional or first-order logic, but possibly a non-classical logic. Ideally, when a formula is found to be valid for the logic, the theorem prover produces a proof or constructs a counterexample otherwise. These proof procedures are capable of solving incredibly large problems in a short amount of time because of readily available computational resources. The community surrounding the international conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX) [2] has developed fast and practical automated theorem provers for many logics. Tableaux provers are particularly competitive because of their numerous heuristics and optimizations, such as restricted backtracking and focusing. Internally, automated theorem provers use a proof system, a formalization of the target logic. In general, instead of generating all possible proofs from the axioms and rules until one leading to the desired formula is found, efficient theorem provers usually start at the goal formula, searching backwards though all possible constructions of the proof.

From the perspective of logics as sets of formulas closed under modus ponens and uniform substitution, intuitionistic logic is a proper subset of classical logic since it rejects the law of the excluded middle ($p \vee \neg p$). Logics that lie between intuitionistic and classical logic in terms of subset inclusion are collectively called intermediate logics. These include Jankov, Gödel-Dummett, Kreisel-Putnam, Scott logic and many more. While the development of automated theorem provers has traditionally focused on classical logic, many practical applications often also require non-classical reasoning. For example, dependently typed programming languages leverage the constructive nature of intuitionistic logic through the Curry-Howard isomorphism. Furthermore, intermediate logics have their applications in the field of answer set programming [3]. From the study of proof theory, it is known that most intermediate logics lack a suitable proof system for automated theorem proving. Ciabattoni et al. [4, 5, 6], proposed the novel idea of cut-restriction that allows usage of the standard sequent calculus as proof system for proving theorems of intermediate logics.

Recently, Fiorentini and Ferrari [7] showed that cut-restriction forms a good basis for the implementation of an automated theorem prover, by introducing a decision procedure for some intermediate logics which they named intuitRIL. The focus of their work has been on the optimization for certain well-known intermediate logics, currently implemented logics include Gödel-Dummett Logics, Jankov Logic and Kreisel-Putnam Logic. Their implementation is an extension of an existing automated theorem prover for intuitionistic propositional logic that reduces the problem of deciding theorems of intermediate logics to the satisfiability problem (SAT). The procedure constructs either a sequent calculus derivation or a Kripke countermodel. The authors also note that the procedure could be extended to support more intermediate logics in a modular fashion.

While, intermediate logics have been studied extensively from a proof-theoretic perspective, proof search procedures, and especially automated theorem prover implementations are rare. Fiorino [8, 9] presented duplication-free tableaux calculi for Gödel-Dummett logic, Jankov logic and two other propositional intermediate logics. While Kuznets and Lellmann [10] give semantically inspired constructions of nested sequent calculi for propositional intermediate logics including Gödel-Dummett logic, and a prototype proof search implementation was also presented.

Through literature review, it has become clear that there are currently no automated theorem provers that support more than a few intermediate logics. The closest being intuitRIL, which can be extended to support additional logics, raising the question whether it is possible to generalize the procedure further. This brings us to the main research question of this thesis.

Is it possible to implement an all-purpose automated theorem prover for intermediate logics?

Fiorentini and Ferrari note that the usage of an incremental SAT-solver for intuitRIL, prevents them from exploiting standard sequent and tableaux calculi optimizations. Therefore, the presented research also investigates if a procedure based on conventional proof systems could still compete with intuitRIL for the logics it supports. Hence, in addition to the main research question, the following sub-questions are taken into consideration.

– *Are there any performance-increasing optimizations or search space reducing heuristics possible for proof search in intermediate logics?*
– *If it is possible to implement the procedure using standard proof systems, how would its performance compare to the current state-of-the-art theorem prover for these logics?*

To answer these questions, the automated theorem prover SuperJ has been developed and implemented in the functional programming language Haskell. The prover supports many intermediate logics via an embedding-preprocessing step followed by intuitionistic proof search. The foundation underlying the prover is the theory of cut-restriction, which has been demonstrated to be applicable to large classes of logics including commutative substructural and modal logics. By focussing on the smaller but infinite class of intermediate logics, the prover can be seen as a first step towards a universal automated theorem prover for non-classical logics.

After this introduction, the thesis is structured as follows. Chapter 2 provides the necessary proof theoretical background, with a focus on sequent calculi and intermediate logics. Chapter 3 continues with a discussion of the theory of cut-restriction and the usage of sequent calculi for proof search procedures. Chapter 4 introduces the automated theorem prover SuperJ, presents implementation details and performance evaluations comparing it to the state-of-the-art theorem prover intuitRIL. Finally, Chapter 5 concludes with a discussion of the results and possible directions for future work.

Acknowledgements

Parts of this work have been incorporated into the short workshop paper by Bastiaan Haaksema, Jens Otten, and Revantha Ramanayake, *"Implementing Intermediate Logics"* [11] accepted at the 5th International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL) associated with the International Joint Conference on Automated Reasoning (IJCAR 2024).

Related Work

Besides the theory of cut-restriction by Ciabattoni et al. [5], there are several other proposals for that aim to generalize proof systems for non-classical logics. Though these alternatives are either less general or unsuitable for the construction of the envisioned universal automated theorem prover that supports intermediate logics, two approaches are discussed here as their ideas are closely related.

Lahav and Zohar [12] introduced a generalization of analytic sequent calculi that retains certain results, such as decidability. Their parameterized notion of the subformula property allowed them to reduce the decidability problem of these calculi to the satisfiability problem [13]. Thus, effectively replacing proof searching by SAT solving instead. This is beneficial, since there exist many industrial-grade SAT solvers that are considered very efficient despite their exponential worst-case time complexity. While this presentation includes interesting complexity results, the proposed reduction is limited to pure calculi, whose rules do not enforce any limitations on the context formulas while including all the usual structural rules. This restriction still allows to capture many interesting logics including paraconsistent logics and primal logic, though unavoidably also excludes many other logics, such as our desired intermediate logics.

Fuenmayor and Benzmüller [14] have presented semantical embeddings of non-classical logics into classical higher order logic (HOL). Logics that have been shown to be compatible include conditional logics [15] and every normal modal logic [16]. Classical higher order logic also benefits from existing automated theorem provers. While there is a cut-free sequent calculus for HOL, its predicate variables may be instantiated with terms that introduce arbitrary new formulas. This allows for cut-simulation, which breaks the analyticity. Therefore, automated theorem provers based on this theory require various other means to avoid cut-simulations [17]. Furthermore, some important metalogical properties (such as the decidability of many intermediate and substructural logics) are obscured or lost under the embeddings into HOL [18, 5].

Chapter 2

Preliminaries

The proof systems that form the basis of many automated theorem provers originated from the research area of structural proof theory, and arose as a means to formalize the logical steps in mathematical arguments. Proof calculi present logics as formal systems consisting of a language, axioms, and rules of inference. Well-known styles of proof calculi include Hilbert-style calculi, sequent calculi, tableaux systems and the natural deduction system. In this chapter, common proof systems for classical and intuitionistic logic are introduced in a similar fashion as the textbooks of Galatos et al. [19] and Ono [20], starting with Hilbert-style calculi, followed by Gentzen's sequent calculi for classical and intuitionistic logic and concluding with a brief discussion on intermediate logics.

2.1. Language of Logics

For our basic formal language of logic, we have the logical connectives $\wedge$, $\vee$, $\to$ and $\neg$ for conjunction, disjunction, implication, and negation respectively. Additionally, $p, q, r, \dots$ are propositional variables belonging to the fixed countable set Φ. Arbitrary well-formed formulas are represented by the metavariables A, B, C, D, E, F. As usual, well-formed formulas are defined inductively, starting with propositional variables and built up using the logical connectives according to the following grammar in Backus-Naur form.

$$F \coloneqq p \in \Phi \mid F \wedge F \mid F \vee F \mid F \to F \mid \neg F$$

When writing formulas, $(\neg)$ binds more tightly than all other logical connectives. Otherwise, parentheses are used where required to avoid ambiguities. Any formula that appears in the inductive definition of some formula F, including F itself, is called a subformula of F. The set of all subformulas of F is denoted $\texttt{subf}(F)$ and the set of all propositional variables occurring in F is denoted $\texttt{var}(F)$. We will constrain our attention to the propositional logics compatible with the language defined so far. Though it can be extended to support other logics with e.g. quantifiers, modal operators, or substructural logical connectives.

2.2. Hilbert-style Calculus for Classical Logic

There are several ways of formalizing classical logic (**CL**), one of the standard ones being the Hilbert-style calculi. These systems typically consist of axiom schemes and only a few rules of inference, which determine the provability of formulas in the system. I.e. a given formula is provable in the system if it can be derived though rule applications, starting from the axiom schemes. Then, a proof of the formula is a finite tree-like structure showing the derivation. These syntactic concepts are conveniently symbolic and mechanical, as opposed to the semantical approaches to logic such as (boolean) algebra or model theory.

There are many alternative Hilbert-style calculi for classical logic, differing in their selection of axiom schemes. A standard one is the system **HK** [19] presented in Figure 2.1. This particular system is composed of axiom schemes and the single rule of *modus ponens*. Axiom schemes are families of formulas with the same syntactic shape, whereas the rule of modus ponens allows for the deduction of a formula B from the formulas $A \to B$ and A. Instances of the axiom schemes and the rule are obtained by replacing the metavariables in the axiom schemes with arbitrary formulas.

$$\frac{A \qquad A \to B}{B}\ \text{(mp)}$$

(A1) $A \to (B \to A)$

(A2) $(A \to (B \to C)) \to ((A \to B) \to (A \to C))$

(A3) $(A \wedge B) \to A$

(A4) $(A \wedge B) \to B$

(A5) $(A \to B) \to ((A \to C) \to (A \to (C \wedge B)))$

(A6) $A \to (A \vee B)$

(A7) $B \to (A \vee B)$

(A8) $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$

(A9) $(A \to B) \to ((A \to \neg B) \to \neg A)$

(A10) $A \to (\neg A \to B)$

(A11) $\neg\neg A \to A$

Figure 2.1: Hilbert-style calculus **HK** for classical logic.

Proofs in the Hilbert-style calculus **HK** are constructed inductively. Every axiom scheme instance is a proof of itself, and the modus ponens rule combines the two proofs of its premises to produce a proof of its conclusion.

Example 1 *The formula $p \to p$ has the following proof in **HK**, where the uppermost formulas in the proof are instances of axiom schemes (A1), (A1) and (A2) (from left to right), respectively.*

$$\dfrac{p \to (q \to p) \qquad \dfrac{p \to ((q \to p) \to p) \qquad (p \to ((q \to p) \to p) \to ((p \to (q \to p)) \to (p \to p)))}{(p \to (q \to p)) \to (p \to p)}}{p \to p}$$

From the example proof above, it should be clear that for any formula A, the formula $A \to A$ is provable in **HK**, by replacing every occurrence of P in the proof with A. This demonstrates that the logic formalized by the system is not only closed under modus ponens, but also under uniform substitution of propositional variables for arbitrary formulas.

It may be convenient to introduce a nullary logical connective (logical constant) $\top$ for the true proposition in our language. The Hilbert-style calculus **HK** can be adjusted accordingly by adding the constant as an axiom (A12), shown below. It is also possible to introduce a logical constant $\bot$ for the false proposition and redefining the negation $\neg A$ as an abbreviation for $A \to \bot$. In this case, the axiom schemes (A9) and (A10) can be replaced by (A13). The axiom scheme (A11) of **HK** is called the law of double negation. It can be replaced by the law of excluded middle (A14), under the assumption of the other axiom schemes.

(A12) $\top$ (A13) $\bot \to A$ (A14) $A \vee \neg A$

Even though the Hilbert-style calculus **HK** is an elegant formalization of classical logic, it lacks a *normal form* of proofs and a natural proof search algorithm. Specifically, this system is not well-suited for backward proof search, where rules are applied in reverse to find a proof of a given formula. Consider that the modus ponens rule requires the premise A to be known in order to apply it, even though it cannot be easily derived from the conclusion B. Next, we will see another proof system that can be adapted for proof searching.

2.3. Sequent Calculus for Classical Logic

In 1935, Gentzen [21] introduced another kind of proof system, the sequent calculi **LK** for classical logic and **LJ** for intuitionistic logic. Their main advantage over Hilbert-style systems is the existence of a normal form for proofs of every provable formula due to the famous cut-elimination theorem, showing that any proof can be normalized to one without any application of the cut rule (generalization of modus ponens). Importantly, it turns out that a decision procedure can be obtained from this consequence. In order to achieve this, the basic expressions in systems by Gentzen are *sequents* written using a meta-logical language, as opposed to formulas in Hilbert-style calculi and natural deduction systems. These expressions have the following form, where each A_i $(0 \leq i \leq n)$ and each B_j $(0 \leq j \leq m)$ are formulas.

$$\underbrace{A_1, \ldots, A_n}_{\text{antecedent}} \quad \Rightarrow \quad \underbrace{B_1, \ldots, B_m}_{\text{succedent}}$$

Here, commas and the sequent arrow '$\Rightarrow$' are metalogical symbols that allow reasoning inside of the formula (fixed depth deep inference). Thus, a sequent is a pair of finite sequences of formulas, where the left sequence is called the *antecedent* and the right sequence the *succedent*. The intended meaning of a sequent is that all the formulas in the antecedent imply any one of the formulas in the succedent, i.e. $\bigwedge \Gamma \to \bigvee \Delta$ where $\bigwedge \Gamma$ is the iterated conjunction $(\ldots(A_1 \wedge A_2) \wedge \ldots A_n)$, and $\bigvee \Delta$ the iterated disjunction $(\ldots(B_1 \vee B_2) \vee \ldots B_m)$. When the antecedent is empty, the sequent has the formula interpretation of $\bigvee \Delta$. Conversely, when the succedent is empty, the sequent represents $\neg \bigwedge \Gamma$. This formula interpretation of a sequent is also known as its *corresponding formula*. If the succedent contains at most one formula then the sequent is said to be single-succedent, otherwise it is multi-succedent.

The sequent calculus **LK** for classical logic, presented in Figure 2.2, consists of the identity axiom scheme and three kinds of inference rules: the cut rule, rules for logical connectives and structural rules. For the rules of inference, the metavariables appearing in the rules are called *active formulas* and the active formula of the cut rule is its *cut formula*. Greek uppercase letters are used to denote the rule *context*, i.e. the formulas in the sequent that are not active. Furthermore, the sequents above the line of a rule of inference are its upper sequents (the premises) and the sequent below the line is its lower sequent (the conclusion). For each inference rule in **LK**, its lower sequent is provable if all of its upper sequents are provable.

Axiom:

$$A \Rightarrow A$$

Cut rule:

$$\frac{\Gamma \Rightarrow \Delta, A \qquad A, \Sigma \Rightarrow \Pi}{\Gamma, \Sigma \Rightarrow \Delta, \Pi}\ (\mathsf{cut})$$

Rules for logical connectives:

$$\frac{A_i, \Gamma \Rightarrow \Delta}{A_1 \wedge A_2, \Gamma \Rightarrow \Delta}\ (\mathsf{L}\wedge) \quad i \in \{1,2\} \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B}\ (\mathsf{R}\wedge)$$

$$\frac{A, \Gamma \Rightarrow \Delta \qquad B, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta}\ (\mathsf{L}\vee) \qquad \frac{\Gamma \Rightarrow \Delta, A_i}{\Gamma \Rightarrow \Delta, A_1 \vee A_2}\ (\mathsf{R}\vee) \quad i \in \{1,2\}$$

$$\frac{\Gamma \Rightarrow \Delta, A \qquad B, \Sigma \Rightarrow \Pi}{A \to B, \Gamma, \Sigma \Rightarrow \Delta, \Pi}\ (\mathsf{L}{\to}) \qquad \frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \to B}\ (\mathsf{R}{\to})$$

$$\frac{\Gamma \Rightarrow \Delta, A}{\neg A, \Gamma \Rightarrow \Delta}\ (\mathsf{L}\neg) \qquad \frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}\ (\mathsf{R}\neg)$$

Structural rules:

$$\frac{\Gamma, A, B, \Sigma \Rightarrow \Delta}{\Gamma, B, A, \Sigma \Rightarrow \Delta}\ (\mathsf{LE}) \qquad \frac{\Gamma \Rightarrow \Delta, A, B, \Pi}{\Gamma \Rightarrow \Delta, B, A, \Pi}\ (\mathsf{RE})$$

$$\frac{A, A, \Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\ (\mathsf{LC}) \qquad \frac{\Gamma \Rightarrow \Delta, A, A}{\Gamma \Rightarrow \Delta, A}\ (\mathsf{RC})$$

$$\frac{\Gamma \Rightarrow \Delta}{A, \Gamma \Rightarrow \Delta}\ (\mathsf{LW}) \qquad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, A}\ (\mathsf{RW})$$

Figure 2.2: Sequent calculus **LK** for classical logic.

A formula A is said to be provable in the sequent calculus when there is a proof of the sequent $\Rightarrow A$ in it. Like in the Hilbert-style calculus, a proof in the sequent calculus is a tree-like structure showing the derivation, as in Example 2. The sequent at the root of the derivation is the *end sequent* to be proved. Furthermore, a sequent is provable in sequent calculus **LK** if and only if its corresponding formula is provable in the Hilbert-style calculus **HK**.

Similar to the Hilbert-style calculus **HK**, the sequent calculus **LK** can be extended with the logical constants $\top$ and $\bot$ by adding the axioms $\Rightarrow \top$ and $\bot \Rightarrow$ respectively. When the negation is redefined as an abbreviation for $A \to \bot$, the left (L$\neg$) and right (R$\neg$) negation rules can be deleted, as they are now derivable from the other rules. Note that while this simplifies the system, it does result in slightly longer proofs compared to the system with rules for negation. This is the preferred concise presentation of sequent calculi in the book of Troelstra and Schwichtenberg [22].

Example 2 *Proof of axiom* (A2) *in sequent calculus* ***LK****, where* (LE*) *denotes multiple applications of the left exchange rule* (LE).

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \Rightarrow A \qquad \dfrac{\dfrac{A \Rightarrow A \qquad \dfrac{B \Rightarrow B \qquad C \Rightarrow C}{B \to C, B \Rightarrow C}\,(\mathsf{L}{\to})}{A \to (B \to C), A, B \Rightarrow C}\,(\mathsf{L}{\to})}{B, A, A \to (B \to C) \Rightarrow C}\,(\mathsf{LE^*})}{A \to B, A, A, A \to (B \to C) \Rightarrow C}\,(\mathsf{L}{\to})}{A \to B, A, A \to (B \to C) \Rightarrow C}\,(\mathsf{LC})}{A, A \to B, A \to (B \to C) \Rightarrow C}\,(\mathsf{LE})}{\dfrac{\dfrac{A \to B, A \to (B \to C) \Rightarrow A \to C}{A \to (B \to C) \Rightarrow (A \to B) \to (A \to C)}\,(\mathsf{R}{\to})}{\Rightarrow (A \to (B \to C)) \to ((A \to B) \to (A \to C))}\,(\mathsf{R}{\to})}\,(\mathsf{R}{\to})
$$

2.4. Proof Search via Sequent Calculus

When using proof systems for proof searching, it is generally more convenient to start at the end sequent and work backwards, attempting to find a proof-like tree obtained by applying rules in reverse until all of its topmost sequents are axiom instances. More formally, for a sequent $\mathcal{S}$, proof searching is the process of constructing a proof by applying rules of inference in the converse direction, until all the upper sequents are axiom scheme instances. If successful, the sequent $\mathcal{S}$ is considered provable in the calculus. Only when exhaustive proof search has failed, it can be concluded that the sequent is not provable.

At first glance, the presence of the cut rule in **LK** seems to imply that there has not been any progress compared to their Hilbert-style counterparts with regard to the suitability for backward proof search, as the cut rule is a generalization of modus ponens. However, due the celebrated *cut-elimination* theorem of Gentzen [21], it is no longer necessary to take the cut rule into account when searching for a proof in the sequent calculus. Since, through detailed case analysis, it was shown that any proof in the sequent calculus can be transformed into a normal form without any application of the cut rule. When cut-elimination holds for a given sequent calculus, it is said to be cut-free. Additionally, the system gains the subformula property, often referred to as *analyticity*.

Definition 1 *(Subformula property) A sequent calculus has the subformula property when every provable sequent has a proof in it, where every formula appearing in the proof is a subformula of the end sequent.*

With the omission of the cut rule, we can modify the sequent calculus further to improve its suitability for proof searching. The structural rules of the sequent calculus **LK** are useful for studying the properties of logics, but also have the effect of exploding the proof search space. Consider the exchange rules (LE, RE) that allow the formulas in the antecedent and succedent of the sequent to be reordered. Exhaustive proof search requires all possible formula permutations to be considered. Furthermore, by alternating applications of the contraction (LC, RC) and weakening (LW, RW) rules, it is trivial to create an infinite proof search tree.

Axiom:

$$A, \Gamma \Rightarrow \Delta, A$$

Rules of inference:

$$\frac{A, B, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta}\,(L\wedge) \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B}\,(R\wedge)$$

$$\frac{A, \Gamma \Rightarrow \Delta \qquad B, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta}\,(L\vee) \qquad \frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B}\,(R\vee)$$

$$\frac{\Gamma \Rightarrow \Delta, A \qquad B, \Gamma \Rightarrow \Delta}{A \rightarrow B, \Gamma \Rightarrow \Delta}\,(L\rightarrow) \qquad \frac{A, \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \rightarrow B}\,(R\rightarrow)$$

$$\frac{\Gamma \Rightarrow \Delta, A}{\neg A, \Gamma \Rightarrow \Delta}\,(L\neg) \qquad \frac{A, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A}\,(R\neg)$$

Figure 2.3: Sequent calculus **LK*** for classical logic.

The sequent calculus **LK*** for classical logic is a refined variant of **LK** that has neither explicit structural rules nor the cut rule. This refined calculus enjoys several proof search advantages. First, by making the antecedent and succedent unordered collections of formulas, the exchange rules (LE, RE) can be omitted from the sequent calculus. This is achieved by changing the antecedent and succedent of the sequents to be finite multisets instead of sequences. Multisets are functions mapping each element from a set (its universe) to a natural number (its multiplicity). Next, consider that the application of the weakening rules can be pushed upwards in the proof tree, towards its leafs. Then, these rule applications can be eliminated by generalizing the identity axiom from $A \Rightarrow A$ to $A, \Gamma \Rightarrow \Delta, A$. Finally, the contraction rules can be absorbed into the rules for logical connectives. For example, the left conjunction rule (L∧) of **LK** can be viewed as a choice of formula A_i from the conjunction $A_1 \wedge A_2$, requiring the use of left contraction when both are needed in the proof. This is avoided in **LK*** by retaining both formulas in the antecedent, since the generalized identity axiom ignores any unused formulas. In the book of Troelstra and Schwichtenberg [22], the multiset based sequent calculus without structural rules is denoted **G3cp** for classical logic. As previously noted, these systems also lack the rules for negation since they treat that logical connective as an abbreviation. However, since this omission results in slightly longer proofs, **LK*** is preferred for proof searching.

When the root-first proof search procedure fails to prove the upper sequent of an applicable rule of inference, it is usually required to investigate all other possible rule applications to conclude that a sequent is not provable. This is called backtracking and can be avoided in the case of invertible rules. Rules are *invertible* if whenever the lower sequent of a rule is provable then so are the upper sequents. For example, the left conjunction rule (L∧) discussed earlier was not invertible in system **LK**, since while the possible lower sequent $A \wedge B \Rightarrow A$ is provable, the upper sequent $B \Rightarrow A$ is not.

Definition 2 *(Invertibility) A rule of a sequent calculus is invertible, when the lower sequent of the rule is provable if and only if the upper sequents are provable.*

Because the failure to prove the upper sequent of an invertible rule does imply the unprovability of its lower sequent, it is not necessary to consider any other rule applications in this case. Besides solving the issues caused by the structural rules, proof searching in **LK*** avoids backtracking completely, since all of its rules of inference are invertible. The proof search procedure for **LK*** is guaranteed to terminate, since for each rule of the sequent calculus the total number of logical connectives in the formulas of the upper sequents is strictly smaller than that of the lower sequent. A common proof search strategy for **LK*** is to apply rules with one upper sequent before those with multiple upper sequents, as this avoids having to apply the unary premise rule in multiple branches of the proof tree.

Example 3 *Proof of axiom* (A2) *in sequent calculus* ***LK****, *obtained by successful proof search.*

$$
\begin{prooftree}
\AxiomC{$A \Rightarrow C, A$}
\AxiomC{$A \Rightarrow C, A$}
\AxiomC{$B, A \Rightarrow B, C$}
\AxiomC{$C, B, A \Rightarrow C$}
\RightLabel{(L$\to$)}
\BinaryInfC{$B \to C, B, A \Rightarrow C$}
\RightLabel{(L$\to$)}
\BinaryInfC{$B, A, A \to (B \to C) \Rightarrow C$}
\RightLabel{(L$\to$)}
\BinaryInfC{$A, A \to B, A \to (B \to C) \Rightarrow C$}
\RightLabel{(R$\to$)}
\UnaryInfC{$A \to B, A \to (B \to C) \Rightarrow A \to C$}
\RightLabel{(R$\to$)}
\UnaryInfC{$A \to (B \to C) \Rightarrow (A \to B) \to (A \to C)$}
\RightLabel{(R$\to$)}
\UnaryInfC{$\Rightarrow (A \to (B \to C)) \to ((A \to B) \to (A \to C))$}
\end{prooftree}
$$

2.5. Calculi for Intuitionistic Logic

Intuitionistic logic (**IL**) originated from the dispute between mathematicians in the early 20th century. Brouwer insisted that mathematics should be limited only to constructive concepts and arguments. His intuitionist point of view is known as the Brouwer-Heyting-Kolmogorov interpretation, and can be roughly summarized for the propositional scenario as follows.

- A proof of $A \wedge B$, consists of the proofs of A and B.
- A proof of $A \vee B$, consists of a proof of either A or B.
- A proof of $A \to B$, is an algorithm that transforms any proof of A into a proof of B.

From this perspective, a proof of the disjunction $A \vee B$ is obtained by giving a proof of either A or B. This makes the law of the excluded middle $p \vee \neg p$ generally unacceptable in intuitionistic logic. Since a proof of either p or $\neg p$ is not always available. For example, the Riemann hypothesis is an open problem for which there exists currently neither a proof of the statement nor of its negation.

The Hilbert-style calculus **HJ** for intuitionistic logic is obtained from the system **HK** by removing the axiom scheme of double negation (A11), or the axiom scheme of excluded middle (A14) if it was used instead. Conveniently, the sequent calculus **LJ** for intuitionistic logic is obtained from the system **LK** by restrain the sequents to be single-succedent. I.e. the sequents in **LJ** have the following form, where each A_i $(0 \leq\ i \leq\ n)$ is a formula and B is either a formula or empty.

$$\underbrace{A_1, \ldots, A_n}_{\text{antecedent}} \quad \Rightarrow \quad \underbrace{B}_{\text{succedent}}$$

Axiom:

$$A \Rightarrow A$$

Cut rule:

$$\frac{\Gamma \Rightarrow A \qquad A, \Delta \Rightarrow B}{\Gamma, \Delta \Rightarrow B}\ \textsf{(cut)}$$

Rules for logical connectives:

$$\frac{A_i, \Gamma \Rightarrow B}{A_1 \wedge A_2, \Gamma \Rightarrow B}\ \textsf{(L}\wedge\textsf{)} \quad i \in \{1,2\} \qquad \frac{\Gamma \Rightarrow A \qquad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\ \textsf{(R}\wedge\textsf{)}$$

$$\frac{A, \Gamma \Rightarrow C \qquad B, \Gamma \Rightarrow C}{A \vee B, \Gamma \Rightarrow C}\ \textsf{(L}\vee\textsf{)} \qquad \frac{\Gamma \Rightarrow A_i}{\Gamma \Rightarrow A_1 \vee A_2}\ \textsf{(R}\vee\textsf{)} \quad i \in \{1,2\}$$

$$\frac{\Gamma \Rightarrow A \qquad B, \Delta \Rightarrow C}{A \to B, \Gamma, \Delta \Rightarrow C}\ \textsf{(L}\to\textsf{)} \qquad \frac{A, \Gamma \Rightarrow B}{\Gamma \Rightarrow A \to B}\ \textsf{(R}\to\textsf{)}$$

$$\frac{\Gamma \Rightarrow A}{\neg A, \Gamma \Rightarrow}\ \textsf{(L}\neg\textsf{)} \qquad \frac{A, \Gamma \Rightarrow}{\Gamma \Rightarrow \neg A}\ \textsf{(R}\neg\textsf{)}$$

Structural rules:

$$\frac{\Gamma, A, B, \Delta \Rightarrow C}{\Gamma, B, A, \Delta \Rightarrow C}\ \textsf{(LE)} \qquad \frac{A, A, \Gamma \Rightarrow B}{A, \Gamma \Rightarrow B}\ \textsf{(LC)}$$

$$\frac{\Gamma \Rightarrow B}{A, \Gamma \Rightarrow B}\ \textsf{(LW)} \qquad \frac{\Gamma \Rightarrow}{\Gamma \Rightarrow A}\ \textsf{(RW)}$$

Figure 2.4: Sequent calculus **LJ** for intuitionistic logic.

Due to the restriction to being a single-succedent sequent calculus, the right exchange (RE) and right contraction (RC) rules are no longer relevant, since these rules require at least two formulas in the succedent. Presented in Figure 2.4, the sequent calculus **LJ** is cut-free through cut-elimination, though it does not have an ideal invertible variant like **LK*** for proof searching. The right disjunction rule (R$\vee$) clearly hints towards this, since it is impossible to retain both A_1 and A_2 in the succedent of its upper sequent. The investigation of a suitable variant of **LJ** for proof searching is postponed to the next chapter.

Since the intuitionistic systems are defined as a restriction of their classical counterparts, it is easy to see that the provable formulas of intuitionistic logic are a proper subset of those of classical logic. Therefore, it is the case that every sequent provable in **LJ** is also provable in **LK**. This is not the true for the converse direction, though by Glivenko [23] it is known that a formula A is provable in classical logic if and only if $\neg\neg A$ is provable in intuitionistic logic.

Example 4 *The double negation of the law of excluded middle $\neg\neg(A \lor \neg A)$ is provable in* ***LJ****, as demonstrated by the following proof.*

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{A \Rightarrow A}{A \Rightarrow A \lor \neg A}\,(\mathsf{R}\lor)}{A, \neg(A \lor \neg A) \Rightarrow}\,(\mathsf{L}\neg)}{\neg(A \lor \neg A) \Rightarrow \neg A}\,(\mathsf{R}\neg)}{\neg(A \lor \neg A) \Rightarrow A \lor \neg A}\,(\mathsf{R}\lor)}{\neg(A \lor \neg A), \neg(A \lor \neg A) \Rightarrow}\,(\mathsf{L}\neg)}{\neg(A \lor \neg A) \Rightarrow}\,(\mathsf{LC})}{\Rightarrow \neg\neg(A \lor \neg A)}\,(\mathsf{R}\neg)$$

Thus, intuitionistic logic is powerful enough to check the provability of formulas in classical logic by checking whether their double negation is provable in intuitionistic logic. Glivenko's observation can therefore be interpreted as the *embedding* of classical logic into intuitionistic logic, $F \mapsto \neg\neg F$. Because of this relation between the two logics, one might view intuitionistic logic as a more general logic than classical logic. Following this line of thought, classical logic can be seen as an extension of intuitionistic logic by the law of excluded middle.

2.6. Superintuitionistic Logics

The perspective of classical logic being an extension of intuitionistic logic, provokes the question of what other logics can be obtained by extending intuitionistic logic. In the following, a logic is a set of formulas closed under modus ponens and uniform substitution of propositional variables for arbitrary formulas. Allowing us to identify a given logic with the set of its provable formulas. For a finite set of *characterizing* formulas $\mathcal{A}$, the axiomatic extension of intuitionistic logic **IL** $+ \mathcal{A}$ is the smallest logic containing **IL** and all formulas in $\mathcal{A}$. For example, classical logic is the axiomatic extension of intuitionistic logic through addition of the law of excluded middle, i.e. **CL** $=$ **IL**$+A \lor \neg A$. Collectively, the axiomatic extensions over intuitionistic logic, including intuitionistic logic itself, are called superintuitionistic logics. A logic is consistent if it is not the set of all formulas, i.e. it does not contain a formula and its negation. Classical logic is the largest consistent axiomatic extension of intuitionistic logic, and *intermediate logics* are precisely the consistent superintuitionistic logics. As was already noticed by Godel [24], there are infinitely many intermediate logics. Prime examples of intermediate logics include Jankov, Gödel-Dummett, Kreisel-Putnam and Scott logic. In case of the first, it is obtained through addition of the weak law of excluded middle.

$$\begin{aligned}
\text{Jankov Logic: } \mathbf{JN} &= \mathbf{IL} + \neg A \lor \neg\neg A \\
\text{Gödel-Dummett Logic: } \mathbf{LC} &= \mathbf{IL} + (A \to B) \lor (B \to A) \\
\text{Kreisel-Putnam Logic: } \mathbf{KP} &= \mathbf{IL} + (\neg A \to (B \lor C)) \to ((\neg A \to B) \lor (\neg A \to C)) \\
\text{Scott Logic: } \mathbf{SL} &= \mathbf{IL} + ((\neg\neg A \to A) \to (A \lor \neg A)) \to (\neg\neg A \lor \neg A)
\end{aligned}$$

Sequent calculi for intermediate logics are obtained by the addition of axioms $\Rightarrow A$, where $A \in \mathcal{A}$, to the intuitionistic sequent calculus **LJ**. Unfortunately, cut-elimination does not hold for intermediate logics in general. The lack of analyticity for the sequent calculi of intermediate logics is a major obstacle for the development of a decision procedure for these logics. In the next chapter, this problem is alleviated using cut-restricted sequent calculi, after which the development of a decision procedure for intermediate logics is discussed.

Chapter 3

Methodology

Conventional proof calculi such as the sequent calculus are not expressive enough to provide analyticity for intermediate logics, as well as many other non-classical logics of interest. More precisely, we have seen that the sequent calculi for intermediate logics that were obtained by the addition of axioms to sequent calculus **LJ** do not preserve the cut-elimination property. Making them unsuitable for proof searching. This led the structural proof theory community to obtain analyticity through extension of the structural language of the sequent calculus. This resulted in numerous new proof formalisms such as hypersequent calculi, display calculi and labelled sequent calculi. For example, Ciabattoni et al. [25] presented a solution through cut-free hypersequent calculi for intermediate logics and related logics. However, even though the hypersequent calculus is a natural generalization of the sequent calculus, which reasons on multiple sequents instead of just one, these kinds of enriched proof formalisms are more difficult to implement in automated theorem provers and each requires existing optimizations to be either adapted or redeveloped. Furthermore, Ramanayake [18] notes that the literature of these more exotic proof formalisms contains considerable duplication in terms of their argumentation.

This chapter begins with a discussion of the alternative approach by Ciabattoni et al. [4, 5, 6], who restrict applications of the cut rule to obtain analytical sequent calculi for non-classical logics that facilitate an embedding of intermediate logics into intuitionistic logic. This restriction allows the use of sequent calculi for proof searching, enabling the reuse of plentiful research into sequent and tableaux calculi based procedures. The second half of this chapter will discuss the proof search strategies and important optimizations as found in the literature review of intuitionistic decision procedures by Dyckhoff [26].

3.1. Cut-restricted Sequent Calculi

The new paradigm of cut-restriction aims to preserve some form of analyticity in sequent calculi where cut-elimination is not possible. This novel idea has been demonstrated for large classes of logics, including commutative substructural and some normal modal logics. Since the superintuitionistic logics discussed earlier are contained in the more general hierarchy of substructural logics [19], we can focus on the sharper results for intermediate logics presented in Ciabattoni et al. [5].

The idea of cut-restriction is to allow only axiom instances of the logic to be used as cut-formulas in the applications of the cut rule. A set of axiom instances Θ is obtained by evaluating a bounding function ψ that takes as arguments a set of axioms $\mathcal{A}$ and a formula F (usually, the formula that we wish to prove). A cut-restricted sequent calculus is then a standard sequent calculus where every cut rule application and axiom instance occurrence from $\{\Rightarrow A \mid A \in \mathcal{A}\}$ is restricted to the following context, with $A \in \Theta$.

$$\dfrac{\Rightarrow A \qquad \dfrac{\cdots}{A, \Gamma \Rightarrow B}}{\dfrac{\Gamma \Rightarrow B}{\cdots}}\ \text{(cut)}$$

While the subformula property does not hold for these sequent calculi, they do satisfy a relaxed form of analyticity. Specifically, every provable sequent in a cut-restricted sequent calculus has a proof in it, where every formula appearing in the proof is a subformula of the end sequent or of some axiom instance.

Naturally, it is preferable for proof searching that the bounding function is as restrictive as possible, in the sense that the set of instances is small. Since the smallest set of possible cut-formulas imposes the largest restriction on the proof search space. Below are four examples of bounding functions, listed in order of decreasing restrictiveness, where $A_1, \ldots, A_n$ is a non-repeating conjunction if all formulas A_i are pairwise distinct. Since these functions were originally presented for substructural logics, they were adapted to the context of intermediate logics by replacing the substructural fusion operator by conjunction.

1. The variable-bounding function $\psi_v(\mathcal{A}, F)$ contains all instances of formulas in $\mathcal{A}$ whose variables have been substituted by variables occurring in F.
2. The formula-bounding function $\psi_f(\mathcal{A}, F)$ contains all instances of formulas in $\mathcal{A}$ whose variables have been substituted by subformulas occurring in F.
3. The set-bounding function $\psi_s(\mathcal{A}, F)$ contains all instances of formulas in $\mathcal{A}$ whose variables have been substituted by non-repeating conjunctions of subformulas occurring in F.
4. The multiset-bounding function $\psi_m(\mathcal{A}, F)$ contains all instances of formulas in $\mathcal{A}$ whose variables have been substituted by conjunctions of subformulas occurring in F.

The first three bounding functions result in a finite set of axiom instances, while the fourth results in an infinite set, as it allows repeating conjunctions of subformulas of F. The upper bound of the restriction on cut formulas, capturing all intermediate logics, would be the infinite set of all axiom instances. In general, assuming that the targeted intermediate logic has a cut-free hypersequent calculus as in Ciabattoni et al. [25], the multiset-bounding is sufficient for a sound and complete cut-restricted sequent calculus. Fortunately, the usage of the resulting infinite set of axiom instances can be avoided. Due to the presence of all the usual structural rules in sequent calculi for intermediate logics, the required set of axiom instances collapses into that of the set-bounding function.

The best bounding function to use for an intermediate logic is non-trivial and depends on the axioms that were chosen to obtain the axiomatic extension. Under certain conditions, it is possible to use the formula-bounding function, or even the variable-bounding function, to obtain an even smaller set of axiom instances. In the following definition, let $[A/B]$ denote the uniform substitution of A for all occurrences of B.

Definition 3 *(Ω-propagation property) Formula A has the Ω-propagation property for a set of binary connectives Ω if for propositional variables p, q and r, and every $\circ \in \Omega$, the following holds:*

$$A[q/p], A[r/p] \Rightarrow A[q \circ r/p] \text{ is provable in } \textbf{LJ}.$$

For an intermediate logic **IL** $+ \mathcal{A}$ which allows the usage of the multiset-bounding function, the formula-bounding function can be used if all formulas in $\mathcal{A}$ have the $\{\land\}$-propagation property. Furthermore, for an arbitrary intermediate logic **IL** $+ \mathcal{A}$, the variable-bounding function can be used if all formulas in $\mathcal{A}$ have the $\{\land, \lor, \rightarrow\}$-propagation property. Though it should be noted that there are other circumstances in which the variable-bounding function can be used.

Of the four intermediate logics presented in Section 2.6., both Jankov logic **JN** and Gödel-Dummett logic **LC** allow the use of the preferred variable-bounding function. In the case of Jankov logic, this can be validated by observing that the weak law of excluded middle, $\neg A \vee \neg\neg A$, satisfies the $\{\wedge, \vee, \rightarrow\}$-propagation property. The situation for Gödel-Dummett logic is more complicated. Using the standard characterizing linearity axiom $(A \rightarrow B) \vee (B \rightarrow A)$, the cut-restricted sequent calculus fails to prove $\neg p \vee \neg\neg p$ using the variable-bounding function, a known theorem of Gödel-Dummett logic. Instead, the best bounding function to use in this case is the formula-bounding function, as the linearity axiom has the $\{\wedge\}$-propagation property. However, Ciabattoni et al. [5] note that the variable-bounding function can be used when the set of characteristic formulas is changed to $\{(A \rightarrow B) \vee ((A \rightarrow B) \rightarrow A), \neg A \vee \neg\neg A\}$, even though $(A \rightarrow B) \vee ((A \rightarrow B) \rightarrow A)$ does not satisfy the $\{\vee, \rightarrow\}$-propagation property. In summary, we have the following sets of axiom instances for Jankov logic and Gödel-Dummett logic, respectively.

$$\begin{aligned}
\Omega_{\textbf{JN}} &= \{\neg A \vee \neg\neg A \mid A \in \texttt{var}(F)\} \\
\Omega_{\textbf{LC}} &= \{(A \rightarrow B) \vee ((A \rightarrow B) \rightarrow A) \mid A, B \in \texttt{var}(F)\} \\
&\quad \cup\ \{\neg A \vee \neg\neg A \mid A \in \texttt{var}(F)\}
\end{aligned}$$

Example 5 *Consider Jankov logic **JN**, and its set of characteristic axioms* $\{\neg A \vee \neg\neg A\}$*. Since all formulas in this set satisfy the* $\{\wedge, \vee, \rightarrow\}$*-propagation property, we may use the variable-bounding function. When attempting to prove the formula* $p \rightarrow q$ *in the cut-restricted sequent calculus for **JN**, the cut-formulas in the proof are restricted to the resulting set of* $\psi_v(\{\neg A \vee \neg\neg A\}, p \rightarrow q)$*. Which in this case would be* $\{\neg p \vee \neg\neg p, \neg q \vee \neg\neg q\}$*.*

3.2. Logical Embeddings of Intermediate Logics

By definition of cut-restricted sequent calculi, every provable formula F has a proof in which the axiom instances and cut-formulas are restricted to the context as seen below on the left, with $\mathcal{P}$ a subproof. This axiom instance can be eliminated from the proof by replacing it with the proof on the right. In this new proof derivation, the sequent $A, \Gamma \Rightarrow B$ is weakened to contain the conjunction of all possible cut formulas $\wedge\Theta$. By propagating this conjunction downwards in the proof tree, we obtain a proof of $\wedge\Theta \Rightarrow F$ with one less axiom instance and cut rule application.

$$\dfrac{\dfrac{\dfrac{\Rightarrow (A \in \Theta) \qquad \dfrac{\mathcal{P}}{A, \Gamma \Rightarrow B}}{\Gamma \Rightarrow B}\,\text{(cut)}}{\cdots}}{\Rightarrow F} \qquad \mapsto \qquad \dfrac{\dfrac{\dfrac{\dfrac{\mathcal{P}}{A, \Gamma \Rightarrow B}}{\wedge\Theta, \Gamma \Rightarrow B}\,(\text{L}\wedge)^*}{\cdots}}{\wedge\Theta \Rightarrow F}$$

By repeating this procedure for every application of the cut rule, and combining the collected Θ conjunctions into one using the left contraction rule, we obtain a proof of $\wedge\Theta \Rightarrow F$. Conversely, this also implies that every provable sequent in the cut-restricted sequent calculus has a proof in which all the cut-formulas are distinct, since the contraction rule can be used upwards to duplicate the cut-formulas. Without any axiom instance $\{\Rightarrow A \mid A \in \mathcal{A}\}$ and cut rule application in the proof of $\wedge\Theta \Rightarrow F$, the sequent has the same proof in the sequent calculus for intuitionistic logic **LJ**.

Taking one more step downwards in the proof using the right implication rule, we obtain a proof of $\Rightarrow \wedge\Theta \rightarrow F$ in **LJ**. This argument results in the following embedding of intermediate logics $\mathbf{IL} + \mathcal{A}$ into intuitionistic logic **IL**, using a suitable bounding function ψ as described earlier.

$$F \quad \mapsto \quad \wedge\psi(\mathcal{A}, F) \rightarrow F$$

Similar to the possible embedding of classical logic into intuitionistic logic, this embedding allows the use of any intuitionistic proof search procedure for proof searching in intermediate logics. By checking if $\wedge\Theta \rightarrow F$ is provable in **LJ** instead of checking if F is provable in a sequent calculus for $\mathbf{IL} + \mathcal{A}$. While not as mature as techniques for classical logic, intuitionistic proof search is reasonably well understood and allows for many optimizations. The next part of this chapter will focus on intuitionistic proof search strategies for proving the formulas of the shape $\wedge\Theta \rightarrow F$ using the sequent calculus.

3.3. Refined Calculi for Intuitionistic Logic

So far, we have discussed the cut-restricted sequent calculi for intermediate logics and the embedding of these logics into intuitionistic logic. Therefore, it is possible to either use the cut-restricted sequent calculi directly, or to use the logical embeddings to perform the proof search in intuitionistic logic instead. Either way, since the calculi for intermediate logics are defined as extensions of sequent calculus **LJ** for intuitionistic logic, there is enough reason to discuss refined sequent calculi for intuitionistic logic and their modifications to facilitate efficient proof searching.

While Gentzen [21] used the sequent calculus **LJ** to prove the decidability of intuitionistic logic, the calculus is not well-suited for backwards proof search. Especially since the left implication rule (L$\rightarrow$), with its lower sequent $A \rightarrow B, \Gamma, \Delta \Rightarrow C$, requires exploration of all possible context splits of the antecedent into the sequences Γ and Δ. Similarly to the modifications of sequent calculus **LK***, this issue is resolved through absorption of the structural rules into the rule for logical connectives of the sequent calculus and the usage of formula multisets instead of sequences. Specifically, for the left implication rule, the context of the lower sequent is duplicated into the upper sequents instead of being split. According to the naming by Troelstra and Schwichtenberg [22], this resulting sequent calculus presented in Figure 3.1 is known as **G3ip** and omits the negation from the language in favour of $\bot$ as by their convention.

Whereas sequent calculi **LJ** and **G3ip** are both single-succedent calculi, Maehara [27] introduced the multi-succedent variant **m-G3ip**. Thus allowing multiple formulas in the succedent of the sequents. Being a multi-succedent makes this calculus roughly correspond to intuitionistic tableaux calculi [26], which are essentially a notational inversion of sequent calculi. Importantly, Egly and Schmitt [28] showed that proofs in this multi-succedent system can be much smaller than those in their single-succedent variants.

While being an improvement over **LJ**, the left implication rule (L$\rightarrow$) of **G3ip** remains problematic. Its left upper sequent is not measurably smaller than its lower sequent, which causes non-termination of the proof search procedure unless equipped with some form of loop detection. Dyckhoff [29, 30] and Hudelmaier [31] rediscovered a method that replaces the left implication rule by four new rules obtained through case analysis of the left subformula of the implication in the lower sequent.

Axioms:

$$A, \Gamma \Rightarrow A \qquad\qquad \bot, \Gamma \Rightarrow A$$

Rules of inference:

$$\frac{A, B, \Gamma \Rightarrow C}{A \wedge B, \Gamma \Rightarrow C}\ (\mathsf{L}\wedge) \qquad\qquad \frac{\Gamma \Rightarrow A \qquad \Gamma \Rightarrow B}{\Gamma \Rightarrow A \wedge B}\ (\mathsf{R}\wedge)$$

$$\frac{A, \Gamma \Rightarrow C \qquad B, \Gamma \Rightarrow C}{A \vee B, \Gamma \Rightarrow C}\ (\mathsf{L}\vee) \qquad\qquad \frac{\Gamma \Rightarrow A_i}{\Gamma \Rightarrow A_1 \vee A_2}\ (\mathsf{R}\vee) \quad i \in \{1, 2\}$$

$$\frac{A \to B, \Gamma \Rightarrow A \qquad B, \Gamma \Rightarrow C}{A \to B, \Gamma \Rightarrow C}\ (\mathsf{L}\!\to) \qquad\qquad \frac{A, \Gamma \Rightarrow B}{\Gamma \Rightarrow A \to B}\ (\mathsf{R}\!\to)$$

Figure 3.1: Sequent calculus **G3ip** for intuitionistic logic.

Dyckhoff named the calculus with these new rules **LJT**, though it has since commonly been referred to as **G4ip**. Using an appropriate measure on sequents, this calculus is guaranteed to terminate, avoiding the need for difficult to implement loop detection. The contribution of Hudelmaier [31] included the assurance of a linear rather than exponential depth proofs using fresh propositional variables in the rules (L∨→) and (L→→) to avoid the duplication of formulas in the upper sequents.

$$\frac{p, A, \Gamma \Rightarrow B}{p, p \to A, \Gamma \Rightarrow B}\ (\mathsf{L}\!\to\!\Phi) \qquad\qquad \frac{A \to (B \to C), \Gamma \Rightarrow D}{(A \wedge B) \to C, \Gamma \Rightarrow D}\ (\mathsf{L}\!\to\!\wedge)$$

$$\frac{A \to p, B \to p, p \to C, \Gamma \Rightarrow D}{(A \vee B) \to C, \Gamma \Rightarrow D}\ (\mathsf{L}\!\to\!\vee) \quad \text{fresh variable } p$$

$$\frac{A, p \to C, B \to p, \Gamma \Rightarrow p \qquad C, \Gamma \Rightarrow D}{(A \to B) \to C, \Gamma \Rightarrow D}\ (\mathsf{L}\!\to\to) \quad \text{fresh variable } p$$

Figure 3.2: Left implication rules of sequent calculus **G4ip**.

Similar to the multi-succedent calculus of Maehara, Dyckhoff [29] introduced the multi-succedent sequent calculus **LJT***, also known as **m-G4ip**. The connections between multi-succedent sequent calculi and tableaux calculi let Avellone et al. [32, 33, 34] continue the development as a tableaux calculus, their methods justified by Kripke semantics. They were able to reintroduce explicit rules for logical negation, which can be seen as special instances of the left implication rules of **G4ip**. This tableaux calculus, adapted back to sequent calculus notation and named **LJ*** for simplicity, is presented in Figure 3.3. The non-invertible rules of this calculus are (L¬∧), (L¬→), (L¬¬), (R¬), (L→→), (L→¬) and (R→). As described in Section 2.4., the failure to prove a sequent using a rule that is not invertible requires the consideration of other possible rule applications. Proof searching through a calculus with non-invertible rules therefore requires a sound backtracking mechanism. This makes the procedure more complicated in practice, but still manageable. For example, backtracking can be avoided for the rules (L→→) and (L→¬) when their right upper sequent is not provable due to these rules being invertible with respect to their second premise.

Axioms:

$$A, \Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, \top \qquad \bot, \Gamma \Rightarrow \Delta$$

Rules of inference:

$$\frac{A, B, \Gamma \Rightarrow \Delta}{A \wedge B, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\wedge) \qquad \frac{\Gamma \Rightarrow \Delta, A \qquad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B}\,(\mathsf{R}\wedge) \qquad \frac{A, \Gamma \Rightarrow \qquad B, \Gamma \Rightarrow}{\neg(A \wedge B), \Gamma \Rightarrow \Delta}\,(\mathsf{L}\neg\wedge)$$

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B}\,(\mathsf{R}\vee) \qquad \frac{A, \Gamma \Rightarrow \Delta \qquad B, \Gamma \Rightarrow \Delta}{A \vee B, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\vee) \qquad \frac{\neg A, \neg B, \Gamma \Rightarrow \Delta}{\neg(A \vee B), \Gamma \Rightarrow \Delta}\,(\mathsf{L}\neg\vee)$$

$$\frac{A, \Gamma \Rightarrow}{\Gamma \Rightarrow \Delta, \neg A}\,(\mathsf{R}\neg) \qquad \frac{A, \Gamma \Rightarrow}{\neg\neg A, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\neg\neg) \qquad \frac{A, \neg B, \Gamma \Rightarrow}{\neg(A \to B), \Gamma \Rightarrow \Delta}\,(\mathsf{L}\neg\to)$$

$$\frac{A, \Gamma \Rightarrow B}{\Gamma \Rightarrow \Delta, A \to B}\,(\mathsf{R}\to) \qquad \frac{A, B, \Gamma \Rightarrow \Delta}{A, A \to B, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\to A)$$

$$\frac{A, \Gamma \Rightarrow \qquad B, \Gamma \Rightarrow \Delta}{\neg A \to B, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\to\neg) \qquad \frac{A \to (B \to C), \Gamma \Rightarrow \Delta}{(A \wedge B) \to C, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\to\wedge)$$

$$\frac{A \to p, B \to p, p \to C, \Gamma \Rightarrow \Delta}{(A \vee B) \to C, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\to\vee) \quad \text{fresh variable } p$$

$$\frac{A, p \to C, B \to p, \Gamma \Rightarrow p \qquad C, \Gamma \Rightarrow \Delta}{(A \to B) \to C, \Gamma \Rightarrow \Delta}\,(\mathsf{L}\to\to) \quad \text{fresh variable } p$$

Figure 3.3: Sequent calculus **LJ*** for intuitionistic logic.

The proof search strategy for **LK*** preferred the applications of invertible unary premise rules over those with multiple premises to find smaller proofs. A natural extension of this strategy would be to consider non-invertible rules after those which are invertible, leaving the non-invertible binary premise rules for last.

Avellone et al. [32] noted that if non-invertible rules (L¬→), (L¬¬) and (L¬∧) are scheduled after all the other rules, they may be treated as invertible rules without loss of completeness. This strategy results in the following classification of rules in according to their behaviour with respect to branching and backtracking, ordered by priority in the proof search procedure. Here, the right non-invertible rules precede the left non-invertible rules, since backtracking was proven not to be unnecessary in the case of the succedent being a singleton formula.

$\mathcal{C}_1 = \{(\mathsf{L}\wedge),\ (\mathsf{R}\vee),\ (\mathsf{L}\to A),\ (\mathsf{L}\to\wedge),\ (\mathsf{L}\to\vee),\ (\mathsf{L}\neg\vee)\}$

$\mathcal{C}_2 = \{(\mathsf{R}\wedge),\ (\mathsf{L}\vee)\}$

$\mathcal{C}_3 = \{(\mathsf{R}\to),\ (\mathsf{R}\neg)\}$

$\mathcal{C}_4 = \{(\mathsf{L}\to\to),\ (\mathsf{L}\to\neg)\}$

$\mathcal{C}_5 = \{(\mathsf{L}\neg\to),\ (\mathsf{L}\neg\neg)\}$

$\mathcal{C}_6 = \{(\mathsf{L}\neg\wedge)\}$

3.4. Proof Search Optimizations

During proof searching in **LJ***, the upper sequents of its rules occasionally have an empty succedent. Whenever this is the case, or the succedent only contains formula occurrences of $\bot$, the proof search procedure can revert to the more efficient classical proof search using the sequent calculus **LK***. This is possible because of the following theorem found in the book of Ono [20], where the mentioned sequent calculi may be replaced by their equivalent variants **LK*** and **LJ*** respectively.

Theorem 1 *For all multisets of formulas* Γ *and* Δ*, the sequent* $\Gamma \Rightarrow \Delta$ *is provable in* ***LK*** *if and only if the sequent* $\neg\Delta, \Gamma \Rightarrow$ *is provable in* ***LJ****.*

The classical provability is also be useful during the exploration of non-invertible rules in **LJ***. Since the provable formulas of classical logic are a superset of those of intuitionistic logic, the failure to prove a sequent in **LK*** implies the unprovability of the sequent in **LJ***.

Further significant optimization of intuitionistic proof search follows from the addition of boolean simplification and replacement rules [32, 33, 34]. The boolean simplification rules of Figure 3.4 can be applied to the formulas of the sequent to reduce them as much as possible before every recursive step of the proof search. These are particularly useful when the formulas have many $\top$ and $\bot$ occurrences.

$$\begin{array}{lll@{\qquad}lll@{\qquad}lll@{\qquad}lll}
(A \wedge \bot) & \mapsto & \bot & (A \vee \bot) & \mapsto & A & (A \rightarrow \bot) & \mapsto & \neg A & & & \\
(\bot \wedge A) & \mapsto & \bot & (\bot \vee A) & \mapsto & A & (\bot \rightarrow A) & \mapsto & \top & (\neg\bot) & \mapsto & \top \\
(A \wedge \top) & \mapsto & A & (A \vee \top) & \mapsto & \top & (A \rightarrow \top) & \mapsto & \top & (\neg\top) & \mapsto & \bot \\
(\top \wedge A) & \mapsto & A & (\top \vee A) & \mapsto & \top & (\top \rightarrow A) & \mapsto & A & & &
\end{array}$$

Figure 3.4: Boolean simplification rules

Boolean simplification rules may be triggered more often by relying on the replacement rules presented below, where $[A/B]$ denotes the uniform substitution of A for all occurrences of B. When a propositional variable p appears in the antecedent of a sequent (L rep), all occurrences of p in the sequent may be replaced by $\top$. If the propositional variable instead appears negated $\neg p$ in the antecedent (L$\neg$ rep), all occurrences of p may be replaced by $\bot$. Classical proof search benefits from an even stronger replacement rule, when a propositional variable p appears in the succedent of a sequent (R rep), all occurrences of p may be replaced by $\bot$. For classical logic, the (R rep) rule supersedes the (L$\neg$ rep) rule, as the latter is a special case of the former, but is unsound for intuitionistic logic.

$$\frac{(\Gamma \Rightarrow \Delta)[\top/p]}{p, \Gamma \Rightarrow \Delta}\text{(L rep)} \qquad \frac{(\Gamma \Rightarrow \Delta)[\bot/p]}{\neg p, \Gamma \Rightarrow \Delta}\text{(L}\neg\text{ rep)} \qquad \frac{(\Gamma \Rightarrow \Delta)[\bot/p]}{\Gamma \Rightarrow \Delta, p}\text{(R rep)}$$

Both the boolean simplification and replacement rules are invertible and particularly useful for the proof search in intermediate logics. Since even though the embedding of these logics into intuitionistic logic may increase the size of the goal formula dramatically, the number of propositional variables remains the same. Meaning that only a few applications of the replacement rules already greatly reduces the size of the sequent to be proven.

Chapter 4

Results

4.1. Proof Search Implementation

A popular programming language used for the implementation of automated theorem provers for intuitionistic logic is Prolog [35, 36, 29, 37, 33]. The declarative programming language allows for a straightforward encoding of the axioms and inference rules of proof systems. Internally, the Prolog engine attempts to find a resolution refutation of the negation of the input query. Typically, the proof search is guided by a depth-first search strategy, exploring rule applications based on their order of appearance in the source code. As in the tutorial of Otten [38], every attempted rule application analyses the formulas in the sequent to find a suitable active formula. Dyckhoff [26] notes the following about such implementations.

> Implementations of the mentioned calculi spend a great deal of time looking along lists to find a formula of a certain form. A better approach is to take the next formula and either analyse it or put it aside in a suitable place for later use. For example, succedent conjunctions can be put aside until all non-branching rules have been dealt with. This can be regarded as a naive form of *focusing*.

This inspired the Haskell implementation of the automated theorem prover SuperJ, which utilizes such a native form of focusing. It is available at `edu.nl/6tdqn` and went through many iterations, consisting of 190 commits resulting in 647 lines of code. The functional programming language Haskell is known for its strong type system and benefits from some existing automated theorem prover benchmarking infrastructure from projects by Claessen and Rosén [39] and Fiorentini [40]. SuperJ is capable of proving formulas in classical, intuitionistic and a wide range of intermediate logics through sequent calculi **LK***, **LJ*** and the logical embeddings presented in Chapter 3. It uses all the proof search optimizations mentioned in Section 3.4..

Formulas are represented using the data type in Haskell shown below, simplified for brevity. During the initial input parsing stage, propositional variables are assigned increasing integer values, hence the `Var Int` constructor. Using integers as variable names increases the performance compared to using strings, allows for efficient substitution operations and the creation of fresh propositional variables, the last of which is required by the (L$\to\vee$) and (L$\to\to$) rules of the sequent calculus **LJ***.

```haskell
data F = Top | Bot | Var Int | F :& F | F :| F | F :> F | Neg F
```

The sequent calculi are implemented using two components each, a schedule that determines the priority of formulas and a recursive function that applies the rules to the sequent based on the formula with the highest priority. Besides the (L$\to A$) rule, all the rules of **LJ*** (including the replacement rules) only have one active formula. The (L$\to A$) rule is the only rule that has two active formulas, but can be applied before the other left implication rules with a higher priority. This allows us to use the classification of rules shown in Section 3.3. to directly determine the priority of formulas.

For example, the following Haskell code snippet shows the priority of formulas in the antecedent (L denotes formulas left of the sequent arrow, i.e. the antecedent) for the sequent calculus **LJ***.

```
L -> \case
  Bot             -> C0; Top               -> C0
  (Var _)         -> C1; (Neg (Var _))     -> C1; (Neg (_ :| _))   -> C1
  (_ :& _)        -> C1; ((_ :& _) :> _)   -> C1; ((_ :| _) :> _)  -> C1
  (_ :| _)        -> C2; (Neg _ :> _)      -> C4; ((_ :> _) :> _)  -> C4
  (Neg (Neg _))   -> C5; (Neg (_ :> _))    -> C5; (Neg (_ :& _))   -> C6
  _ -> CX
```

The priority of formulas in inversely related to the number of its classification, with C0 being the highest priority and C6 the lowest. CX is a special classification for formulas that are not the active formula of any rule. The priority of formulas is used by the view function to select the next active formula in the recursive prove function. Part of the implementation of the prove function is shown below, containing the (L$\bot$), (L rep), (L$\wedge$), (L$\vee$) and (L$\to\to$) rules of **LJ***.

```
prove r = case view r of
  Just (L, f, s) -> case f of
    -- Category 0
    Bot -> True
    -- Category 1
    Var p -> prove (subst p Top s)
    a :& b -> prove (add L a $ add L b s)
    -- Category 2
    a :| b -> all prove [add L a s, add L b s]
    -- Category 4
    (a :> b) :> c
      | not (prove (add L c s)) -> False
      | (p, t) <- fresh s
      , prove (add L a $ add L (b :> p) $ add L (p :> c) $ setR p t)
      -> True
    -- Backtrack
    _ -> prove (lock L f s)
```

The non-invertible (L$\to\to$) in implemented using Haskell pattern guards, which allows for backtracking. When that happens, the formula is locked by giving it the CX classification. The formula may be unlocked by a replacement rule triggered substitution or application of another non-invertible rule. Franzén [35] showed the correctness of this *locking* approach in the context of a single-succedent calculus. The complete module of the intuitionistic proof search procedure is available in Appendix A.

The logical embedding of intermediate logic into intuitionistic logic is implemented in a separate module. The user provided characterizing formulas of the intermediate logic are checked for Ω-propagation properties from Section 3.1., from which the corresponding bounding function is derived. Using the bounding function, the embedding is applied to the input formula, which is then passed to the intuitionistic proof search procedure. Additionally, SuperJ contains predefined embeddings for Jankov and Gödel-Dummett logic using the sets $\Omega_{\mathbf{JN}}$ and $\Omega_{\mathbf{LC}}$ of Section 3.1. respectively.

Limitations

Perhaps ideally, SuperJ would have been verified through implementation in a proof assistant, e.g. Coq or Agda, this was beyond of the scope of this project. Instead, the soundness of the automated theorem prover has been tested with respect to intuitionistic logic, integrated as a test suite with GitHub Actions workflows.

By profiling the Haskell implementation of SuperJ while running the test suite, it became clear that the majority of the time was spent on applying substitutions triggered by the replacement rules. This can be optimized by using a graph-based representation for formulas, as is done in the automated theorem prover PITP [34]. This is especially beneficial when the sequent contains many identical subformulas, as is the case when solving the logical embeddings of intermediate logics.

A fully featured automated theorem prover is not only be able to decide if a formula belongs to the logic, but also present a proof derivation if it does, or a counter model otherwise. SuperJ does not yet provide this functionality, as this was not the focus of the presented research. While sequent calculus proofs could be fairly easily constructed during the proof search, countermodel extraction is slightly more involved. The papers of Dyckhoff and Negri [41, 42] provide some insight into countermodel construction for non-classical logics.

4.2. Experimental Evaluation

The benchmarking of automated theorem provers for non-classical logics is a difficult problem and an active area of research [43, 44]. Currently, curated dataset of formulas for testing automated theorem provers for intermediate logic are not yet available. As the language of intermediate logics are the same as classical and intuitionistic logic, we can use existing benchmark formulas of these logics. Since intermediate logics are an extension of intuitionistic logic, it was decided to use the propositional part of the ILTP problem library for intuitionistic logic [45] to evaluate the SuperJ prover. In addition, benchmark results for the high performance SAT-based theorem prover intuitRIL are used for comparison. This intermediate propositional logic prover, due to Fiorentini and Ferrari [7], is the only automated theorem prover known that supports the same selection of tested intermediate logics. Their methods have similar theoretical foundations, but are justified semantically using Kripke models.

Version 1.1.2 of the ILTP problem library contains 274 propositional formulas with status and difficulty rating information. The problems are formatted according to TPTP syntax and categorized in three domains, Logic Calculi (LCL), Syntactic (SYN) and Intuitionistic Syntactic (SYJ). LCL contains 2 problems that were judged by Siklóssy et al. [46] to be among the hardest of the theorems found in *Principia Mathematica* [47] while the 20 problems of SYN mainly consist of those due to Pelletier [48]. With 252 problems, The SYJ is the largest domain. It can be divided into 12 families of difficult problems collected such as the pigeonhole principle, labelled SYJ201 to SYJ212. For each of these families, there are 20 instances of increasing complexity. Finally, the 12 remaining problems in the SYJ domain come from the test formula set of JProver [49].

The automated theorem provers were evaluated on all 274 propositional problems of the ILTP library. The test were conducted on a 3.6 GHz AMD Ryzen 5 3600 system with 32 GB of RAM running Ubuntu 23.10 with kernel version 5.15. Running time was restricted to 60 seconds per individual problem. Table 4.1 shows the benchmark results comparing SuperJ and intuitRIL, with the best results marked in bold. The solved column indicates the number of problems solved within the time limit, and the time column indicates the total time in seconds for those problems.

		SuperJ (**IL**)		intuitRIL (**IL**)		SuperJ (**JN**)		intuitRIL (**JN**)		SuperJ (**LC**)		intuitRIL (**LC**)	
Domain	Total	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time	Solved	Time
LCL	2	**2**	**0.023**	2	0.024	**2**	**0.023**	2	0.024	**2**	**0.023**	2	0.024
SYJ1	12	**12**	**0.138**	12	0.145	**12**	**0.138**	12	0.144	8	0.092	**12**	**0.144**
SYJ201	20	20	6.533	**20**	**2.765**	14	53.583	**20**	**2.754**	1	0.022	**20**	**2.754**
SYJ202	20	9	28.834	**10**	**14.282**	4	15.417	**10**	**14.322**	1	0.012	**10**	**14.352**
SYJ203	20	**20**	**0.232**	20	0.242	20	5.162	**20**	**0.243**	3	1.724	**20**	**0.244**
SYJ204	20	**20**	**0.234**	20	0.241	20	15.192	**20**	**0.244**	3	0.326	**20**	**0.242**
SYJ205	20	**20**	**0.232**	20	0.242	12	70.909	**20**	**0.248**	0	—	**20**	**0.242**
SYJ206	20	11	25.887	**20**	**0.240**	11	29.026	**20**	**0.239**	4	3.946	**20**	**0.243**
SYJ207	20	20	0.674	**20**	**0.613**	**20**	**0.231**	20	0.612	**20**	**0.496**	20	1.054
SYJ208	20	**20**	**0.856**	20	2.296	**20**	**0.771**	20	2.306	10	1.446	**17**	**165.925**
SYJ209	20	8	6.862	**20**	**0.243**	9	56.864	**20**	**0.240**	4	23.207	**20**	**0.264**
SYJ210	20	**20**	**0.230**	20	0.241	20	16.641	**20**	**0.240**	4	4.636	**20**	**0.313**
SYJ211	20	20	97.895	**20**	**0.241**	12	45.389	**20**	**0.241**	1	3.132	**20**	**97.985**
SYJ212	20	20	0.720	**20**	**0.242**	12	48.558	**20**	**0.241**	5	19.868	**20**	**0.302**
SYN	20	20	11.423	**20**	**0.240**	19	0.217	**20**	**0.239**	19	0.218	**20**	**0.252**

Table 4.1: ILTP performance results comparing SuperJ and intuitRIL

intuitRIL outperforms (or matches) SuperJ in intuitionistic logic, particularly so for domains SYJ206 and SYJ209. Except for SYJ208, SuperJ is only ever faster with a minimal constant factor, this might be due to there being a slightly smaller preprocessing overhead compared to intuitRIL that uses a clausification procedure. For Jankov logic, intuitRIL solved the same number of problems within almost the same amount of time as for intuitionistic logic. There is no real slowdown by the addition of the axiom instantiations for this prover. The SuperJ prover is more sensitive to the addition of the axiom instantiations; e.g., it finds a proof earlier for problems in SYJ209 but is slower in SYJ201-SYJ205, SYJ211, and SYJ212. Finally, for Gödel-Dummett logic, intuitRIL again managed to solve most of the problems within the time limit. However, compared to itself for other logics, it is slower for domains SYJ208 and SYJ211. The performance of SuperJ on the same logic is below that of intuitRIL, except for domains LCL and SYJ207. Perhaps this is not so surprising, considering the axiom instances of Gödel-Dummett logic contain (nested) implications, triggering many non-invertible rule applications in the proof search.

Chapter 5

Conclusion

This work presented the theoretical foundations and implementation of the automated theorem prover SuperJ, showing the feasibility of an all-purpose theorem prover for intermediate logics. The new implementation was evaluated and compared to the current state-of-the-art theorem prover intuitRIL. The benchmark results show that, despite its general nature, SuperJ performs well but is not quite as efficient for Gödel-Dummett logic. It is expected that the performance gap between the two automated theorem provers could be closed by using SAT-based techniques, as intuitRIL does. It would be interesting to see if the integration of a modern SAT solver such as CaDiCaL [50] outperforms the current SAT-based implementation that uses the older MiniSat solver.

Future Work

The embedding approach used by SuperJ is convenient since it allows for the reuse of existing proof search algorithms for intuitionistic logic. However, the information of which formulas are axiom instances is lost through the embedding during proof search. Therefore, it might be beneficial to adapt the notion of sequents. For example, in the sequent $\Theta; \Gamma \Rightarrow \Delta$, the set Θ could be used to store the set of axiom instances available for the proof search. The goal is then be to prove the sequent $\Theta; \Rightarrow F$ for some goal formula F, in which the cut rule can simply select a formula from Θ to move into the antecedent Γ.

$$\frac{\Theta; A, \Gamma \Rightarrow \Delta}{A, \Theta; \Gamma \Rightarrow \Delta}\text{(cut)}$$

The proof search procedure would still be guaranteed to terminate, as the set of available axiom instances is finite and decreases with each application of the cut rule. Crucially, this approach allows for the implementation of strategies such as axiom selection techniques. For example, the **Q**∞ technique [51] prioritizes axiom instances that have at least one propositional variable in common with the sequent to be proved. In our specific context, we can use an even stronger condition. For an axiom instance A to be considered as a cut formula in the cut rule above, it must be the case that $A \in \psi(\mathcal{A}, \bigwedge \Gamma \Rightarrow \bigvee \Delta)$, i.e. it must be derivable from the corresponding formula of the sequent at the current node in the proof tree. While re-evaluation of the bounding function ψ would be costly, this condition also implies that all the propositional variables of A must occur in the sequent $\Gamma \Rightarrow \Delta$, which is much easier to check. This may be quite effective when combined with the replacement rules of Section 3.4., which essentially remove propositional variables from the sequent.

The methodology of cut restriction extends to the larger class of substructural logics. An obstacle to generalize the current implementation to these logics is that the set of axiom instances becomes infinite due to the requirement of the multiset-bounding function. Perhaps there exist some conditions during proof search that can be used to restrict this set of axiom instances to be finite.

Bibliography

[1] Jasmin Blanchette, Laura Kovács, and Dirk Pattinson. “11th International Joint Conference, IJCAR 2022, Haifa, Israel, Proceedings”. In: *Automated Reasoning*. Springer International Publishing, 2022. DOI: 10.1007/978-3-031-10769-6.

[2] Revantha Ramanayake and Josef Urban. “32nd International Conference, TABLEAUX 2023, Prague, Czech Republic, Proceedings”. In: Springer International Publishing, 2023. DOI: 10.1007/978-3-031-43513-3.

[3] Mauricio Osorio, Juan Antonio Navarro, and José Arrazola. *Applications of Intuitionistic Logic in Answer Set Programming*. 2003. DOI: 10.48550/arXiv.cs/0305046.

[4] Agata Ciabattoni, Timo Lang, and Revantha Ramanayake. “Bounded Sequent Calculi for Non-classical Logics via Hypersequents”. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Cham: Springer International Publishing, 2019, pp. 94–110. DOI: 10.1007/978-3-030-29026-9_6.

[5] Agata Ciabattoni, Timo Lang, and Revantha Ramanayake. “Bounded-analytic sequent calculi and embeddings for hypersequent logics”. In: *The Journal of Symbolic Logic* 86.2 (2021), pp. 635–668. DOI: 10.1017/jsl.2021.42.

[6] Agata Ciabattoni, Timo Lang, and Revantha Ramanayake. “Cut-Restriction: From Cuts to Analytic Cuts”. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2023, pp. 1–13. DOI: 10.1109/LICS56636.2023.10175785.

[7] Camillo Fiorentini and Mauro Ferrari. “SAT-Based Proof Search in Intermediate Propositional Logics”. In: *Automated Reasoning*. Springer International Publishing, 2022, pp. 57–74. DOI: 10.1007/978-3-031-10769-6_5.

[8] Guido Fiorino. “An O(n log n)-SPACE Decision Procedure for the Propositional Dummett Logic.” In: *Journal of Automated Reasoning* 27.3 (2001), pp. 297–311. DOI: 10.1023/a:1017515831550.

[9] Guido Fiorino. “Space-efficient Decision Procedures for Three Interpolable Propositional Intermediate Logics”. In: *Journal of Logic and Computation* 12.6 (2002), pp. 955–992. DOI: 10.1093/logcom/12.6.955.

[10] Roman Kuznets and Björn Lellmann. “Interpolation for intermediate logics via injective nested sequents”. In: *Journal of Logic and Computation* 31.3 (2021), pp. 797–831. DOI: 10.1093/logcom/exab015.

[11] Bastiaan Haaksema, Jens Otten, and Revantha Ramanayake. “Implementing Intermediate Logics”. In: *Joint Proceedings of the 5th International Workshop on Automated Reasoning in Quantified Non-Classical Logics (ARQNL)*. 2024.

[12] Ori Lahav and Yoni Zohar. “On the Construction of Analytic Sequent Calculi for Sub-classical Logics”. In: *Logic, Language, Information, and Computation*. Springer Berlin Heidelberg, 2014, pp. 206–220. DOI: 10.1007/978-3-662-44145-9_15.

[13] Ori Lahav and Yoni Zohar. “SAT-Based Decision Procedure for Analytic Pure Sequent Calculi”. In: *Automated Reasoning*. Springer International Publishing, 2014, pp. 76–90. DOI: 10.1007/978-3-319-08587-6_6.

[14] David Fuenmayor and Christoph Benzmüller. *Higher-order Logic as Lingua Franca – Integrating Argumentative Discourse and Deep Logical Analysis*. 2020. DOI: 10.48550/arXiv.2007.01019.

[15] Christoph Benzmüller. "Cut-elimination for quantified conditional logic". In: *Journal of Philosophical Logic* 46.3 (2016), pp. 333–353. DOI: 10.1007/s10992-016-9403-0.

[16] Tobias Gleißner, Alexander Steen, and Christoph Benzmüller. "Theorem Provers for Every Normal Modal Logic". In: *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*. Vol. 46. EasyChair, 2017, pp. 14–30. DOI: 10.29007/jsb9.

[17] Christoph Benzmüller, Nik Sultana, Lawrence C. Paulson, and Frank Theiß. "The higher-order prover Leo-II". In: *Journal of Automated Reasoning* 55.4 (2015), pp. 389–404. DOI: 10.1007/s10817-015-9348-y.

[18] Revantha Ramanayake. *Unifying structural proof theory via bounded sequent calculi*. 2020. DOI: 10.55776/P33548.

[19] Nikolaos Galatos, Peter Jipsen, Tomasz Kowalski, and Hiroakira Ono. *Residuated Lattices: An Algebraic Glimpse at Substructural Logics*. Vol. 151. Studies in Logic and the Foundations of Mathematics. Elsevier, 2007. ISBN: 9780444521415.

[20] Hiroakira Ono. *Proof Theory and Algebra in Logic*. 1st. Short Textbooks in Logic. Springer Singapore, 2019. ISBN: 9789811379963.

[21] Gerhard Gentzen. "Untersuchungen über das Logische Schliessen I, II". In: *Mathematische Zeitschrift* 39.1 (1935), pp. 176–210, 405–431. DOI: 10.1007/bf01201353.

[22] Anne Sjerp Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. 2th. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2000. ISBN: 9781139168717.

[23] Valery Glivenko. "Sur quelques points de la logique de M. Brouwer". In: *Bulletins de la classe des sciences* 15.5 (1929), pp. 183–188.

[24] Kurt Godel. "Eine Interpretation des Intuitionistis-chen Aussagenkalkuls". In: *Ergebnisse eines mathematisches Kolloquiums* 4 (1933), pp. 39–40.

[25] Agata Ciabattoni, Nikolaos Galatos, and Kazushige Terui. "From axioms to analytic rules in nonclassical logics". In: *2008 23rd Annual IEEE Symposium on Logic in Computer Science*. 2008, pp. 229–240. DOI: 10.1109/LICS.2008.39.

[26] Roy Dyckhoff. "Intuitionistic decision procedures since Gentzen". In: *Advances in Proof Theory*. Springer International Publishing, 2016, pp. 245–267. DOI: 10.1007/978-3-319-29198-7_6.

[27] Shôji Maehara. "Eine Darstellung der intuitionistischen Logik in der klassischen". In: *Nagoya mathematical journal* 7 (1954), pp. 45–64.

[28] Uwe Egly and Stephan Schmitt. "On intuitionistic proof transformations, their complexity, and application to constructive program synthesis". In: *Fundamenta Informaticae* 39.1-2 (1999), pp. 59–83. DOI: 10.3233/FI-1999-391204.

[29] Roy Dyckhoff. "Contraction-Free Sequent Calculi for Intuitionistic Logic". In: *The Journal of Symbolic Logic* 57.3 (1992), pp. 795–807. DOI: 10.2307/2275431.

[30] Roy Dyckhoff. "Contraction-free sequent calculi for intuitionistic logic: a correction". In: *The Journal of Symbolic Logic* 83.4 (2018), pp. 1680–1682. DOI: 10.1017/jsl.2018.38.

[31] Jörg Hudelmaier. "An O(n log n)-Space Decision Procedure for Intuitionistic Propositional Logic". In: *Journal of Logic and Computation* 3.1 (1993), pp. 63–75. DOI: 10.1093/logcom/3.1.63.

[32] Alessandro Avellone, Guido Fiorino, and Ugo Moscato. "Optimization techniques for propositional intuitionistic logic and their implementation". In: *Theoretical Computer Science* 409.1 (2008), pp. 41–58. DOI: 10.1016/j.tcs.2008.08.013.

[33] Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. "fCube: An Efficient Prover for Intuitionistic Propositional Logic". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Springer Berlin Heidelberg, 2010, pp. 294–301. DOI: 10.1007/978-3-642-16242-8_21.

[34] Mauro Ferrari, Camillo Fiorentini, and Guido Fiorino. "Simplification Rules for Intuitionistic Propositional Tableaux". In: *ACM Transactions on Computational Logic* 13.2 (2012). DOI: 10.1145/2159531.2159536.

[35] Torkel Franzén. *Algorithmic aspects of intuitionistic propositional logic*. Swedish Institute of Computer Science, 1987.

[36] An Sahlin, Torkel Franzén, and Seif Haridi. "An intuitionistic predicate logic theorem prover". In: *Journal of Logic and Computation* 2.5 (1992), pp. 619–656. DOI: 10.1093/logcom/2.5.619.

[37] Jens Otten. "Clausal Connection-Based Theorem Proving in Intuitionistic First-Order Logic". In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 245–261. DOI: 10.1007/11554.

[38] Jens Otten. "How to Build an Automated Theorem Prover". In: *The 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. 2019. URL: http://jens-otten.de/tutorial_tableaux19/.

[39] Koen Claessen and Dan Rosén. "SAT Modulo Intuitionistic Implications". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 622–637. DOI: 10.1007/978-3-662-48899-7_43.

[40] Camillo Fiorentini. "Efficient SAT-based Proof Search in Intuitionistic Propositional Logic". In: *Automated Deduction – CADE 28*. Cham: Springer International Publishing, 2021, pp. 217–233. DOI: 10.1007/978-3-030-79876-5_13.

[41] Roy Dyckhoff and Sara Negri. "Proof analysis in intermediate logics". In: *Archive for Mathematical Logic* 51.1-2 (2011), pp. 71–92. DOI: 10.1007/s00153-011-0254-7.

[42] Sara Negri. "Proofs and Countermodels in non-classical logics". In: *Logica Universalis* 8.1 (2014), pp. 25–60. DOI: 10.1007/s11787-014-0097-1.

[43] Jens Otten and Thomas Raths. "Problem Libraries for Non-Classical Logics". In: *Automated Reasoning in Quantified Non-Classical Logics*. Vol. 33. EPiC Series in Computing. EasyChair, 2014, pp. 31–36. DOI: 10.29007/mkdw.

[44] Alexander Steen, David Fuenmayor, Tobias Scholl, Geoff Sutcliffe, and Christoph Benzmüller. "Automated Reasoning in Non-classical Logics in the TPTP World". In: *Eighth Workshop on Practical Aspects of Automated Reasoning*. 2022. DOI: 10.48550/arXiv.2202.09836.

[45] Thomas Raths, Jens Otten, and Christoph Kreitz. "The ILTP problem library for intuitionistic logic". In: *Journal of Automated Reasoning* 38 (2007), pp. 261–271. DOI: 10.1007/s10817-006-9060-z.

[46] Laurent Siklóssy, A Rich, and Vesko Marinov. "Breadth-first search: some surprising results". In: *Artificial Intelligence* 4.1 (1973), pp. 1–27. DOI: 10.1016/0004-3702(73)90006-4.

[47] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. University Press, 1963, 1927. ISBN: 9780521846868.

[48] Francis Jeffry Pelletier. "Seventy-five problems for testing automatic theorem provers". In: *Journal of automated reasoning* 2.2 (1986), pp. 191–216. DOI: 10.1007/BF02432151.

[49] Stephan Schmitt, Lori Lorigo, Christoph Kreitz, and Aleksey Nogin. "JProver: Integrating Connection-Based Theorem Proving into Interactive Proof Assistants". In: *Automated Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 421–426. DOI: 10.1007/3-540-45744-5_34.

[50] Armin Biere, Katalin Fazekas, Mathias Fleury, and Maximillian Heisinger. "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020". In: *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*. Vol. B-2020-1. Department of Computer Science Report Series B. University of Helsinki, 2020, pp. 51–53.

[51] Qinghua Liu, Zishi Wu, Zihao Wang, and Geoff Sutcliffe. "Evaluation of Axiom Selection Techniques". In: *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR)*. 2020.

Appendix A

Implementation

The Haskell module below shows the Haskell implementation of the intuitionistic proof search procedure. The complete source code of the automated theorem prover SuperJ is available at `edu.nl/6tdqn`.

```haskell
{-# LANGUAGE FlexibleInstances, LambdaCase #-}
module Prover.Intuition (prove) where

import           Data.Formula
import           Data.Sequent
import qualified Prover.Classic as Cl

class Provable a where
  prove :: a -> Bool

-- | Check formula provability
instance Provable Formula where
  prove = prove . fromFormula (\case
    L -> \case
      Bot           -> C0; Top             -> C0
      (Var _)       -> C1; (Neg (Var _))   -> C1; (Neg (_ :| _))  -> C1
      (_ :& _)      -> C1; ((_ :& _) :> _) -> C1; ((_ :| _) :> _) -> C1
      (_ :| _)      -> C2; (Neg _ :> _)    -> C4; ((_ :> _) :> _) -> C4
      (Neg (Neg _)) -> C5; (Neg (_ :> _))  -> C5; (Neg (_ :& _))  -> C6
      _ -> CX
    R -> \case
      Top      -> C0; Bot      -> C0
      (_ :| _) -> C1; (_ :& _) -> C2
      (Neg _)  -> C3; (_ :> _) -> C3
      _ -> CX;
    )

-- | Check sequent provability
instance Provable Sequent where
  prove r | nullR r = Cl.prove (toFormula r)
  prove r = case view r of
    Just (L, f, s) -> case f of
      -- Category 0
      Bot -> True
      Top -> prove s
      a | member R a s -> True
      -- Category 1
      Var p -> prove (subst p Top s)
      Neg (Var p) -> prove (subst p Bot s)
      a :& b -> prove (add L a $ add L b s)
      Neg (a :| b) -> prove (add L (Neg a) $ add L (Neg b) s)
      (a :& b) :> c -> prove (add L (a :> b :> c) s)
      (a :| b) :> c -> case fresh s of
        (p, t) -> prove (add L (a :> p) $ add L (b :> p) $ add L (p :> c) t)
      -- Category 2
      a :| b -> all prove [add L a s, add L b s]
      a :> b
        | member L a s -> prove (add L b s)
        | not (Cl.prove (toFormula r)) -> False
      -- Category 4
      Neg a :> b
        | not (prove (add L b s)) -> False
        | prove (add L a $ delR s) -> True
      (a :> b) :> c
        | not (prove (add L c s)) -> False
        | (p, t) <- fresh s
        , prove (add L a $ add L (b :> p) $ add L (p :> c) $ setR p t) -> True
      -- Category 5
      Neg (Neg a) -> prove (add L a $ delR s)
      Neg (a :> b) -> prove (add L a $ add L (Neg b) $ delR s)
      -- Category 6
      Neg (a :& b) -> all (\c -> prove (add L (Neg c) $ delR s)) [a, b]
      -- Backtrack
      _ -> prove (lock L f s)
    Just (R, f, s) -> case f of
      -- Category 0
      Top -> True
      Bot -> prove s
      a | member L a s -> True
      -- Category 1
      a :| b -> prove (add R a $ add R b s)
      -- Category 2
      a :& b -> all prove [add R a s, add R b s]
      -- Category 3
      Neg a | prove (add L a $ delR s) -> True
      a :> b | prove (add L a $ setR b s) -> True
      -- Backtrack
      _ -> not (nullR s) && prove (lock R f s)
    Nothing -> False
```