Formalised Proof Theory

Jesse Wu

A thesis submitted in partial fulfillment of the degree of
Bachelor of Computer Science (Honours) at
The Research School of Computer Science
Australian National University

March 2015

Typeset in Palatino by TEX and LATEX 2ε.

Except where otherwise indicated, this thesis is my own original work.

Jesse Wu
26 March 2015

Acknowledgements

My thanks go to my supervisors, Dr Rajeev Goré and Dr Jeremy Dawson. Their belief in my abilities and their aid throughout the project will always be remembered and appreciated.

The discussions, or rather the weekly Friday commiserations, with other students in the honours cohort, as well as John, Leana and Tom have also been a helpful and memorable part of the journey.

I'd also like acknowledge my family and friends, for their support, understanding and even the various diversions during the creation of this thesis.

Abstract

This work describes syntactic proofs for the admissibility of structural rules in various sequent calculi, using an Interactive Theorem Prover (ITP) to ensure the correctness of all proofs. The development of efficient calculi for automated reasoning, forms the key motivation behind this research. Consequently, the calculi involved are all free of structural rules, and rules necessitating backtracking are avoided whenever possible.

A heavy focus is placed on the benefits and difficulties involved in formalising meta-theorems about sequent calculi using proof assistants. Included are admissibility proofs, implemented using the proof assistant Isabelle/HOL, for calculi describing S4 and S4.3 without explicit structural rules. These proofs are based upon a core code base developed by Jeremy Dawson. The formalisation aspect of this work includes extensions of this existing code to handle the complexities of these calculi. The pen and paper proofs for both calculi also cover gaps in the existing literature.

A chapter is also dedicated to introducing a new automated theorem prover for Intuitionistic Propositional Logic (**Int**). The prover IntLSJGC builds upon the work presented in [Goré et al. 2014], by utilising an underlying sequent calculus which allows global caching techniques to have greater impact. Results demonstrate that IntLSJGC is competitive with state of the art provers.

Contents

Chapter 1

Introduction

Logical calculi are systems of rules intended to provide a method for producing deductions of formulae. Such calculi have significant theoretical and practical value in various fields, especially in Mathematics and Artificial Intelligence. This text is motivated by the importance of producing efficient sequent calculi from an automated reasoning perspective, in order to implement fast Automated Theorem Provers (ATPs). ATPs play an important role in various areas, including proving Mathematical theorems, the verification of hardware and software systems as well as in logic programming [Gonthier 2007; Engel 2010; Osorio et al. 2003].

Sequents are syntactical representations of formulae and logical consequence. Sequent calculi operate on these expressions, using rules which may be divided into two general classes – a rule is either logical or structural. The former type typically decompose or otherwise act on logical connectives, while the latter modify the structure of sequents. The three main structural rules are called Weakening, Contraction and Cut.

For automated reasoning, the main issue with basic sequent calculi stems from the nature of structural rules. The presence of such rules result in particularly inefficient search procedures, often imposing a need for heuristics in order to avoid infinite looping. Even then, structural rules can easily create redundant formulae, or unnecessary backtracking to previous sections of a search procedure. Calculi which are free of structural rules are thus highly desirable.

From the theoretical side, eliminating one particular structural rule, Cut, is a key theorem for sequent calculi. Cut-elimination proofs often lead into properties such as consistency, interpolation and Beth definability [Dawson and Goré 2010]. However, these proofs are generally very complex, to the point where and pen and paper proofs of syntactic cut-elimination are prone to errors. The case analysis involved is almost always long and tedious, commonly leading to authors resorting to expressions such as "the other cases are similar", or "we omit details". Sadly, such omissions can lead to incorrect proofs, although the cut-elimination theorems are generally still correct [Goré 2009]. The problem is exacerbated by the strict page limits imposed on many publications.

One method of avoiding these issues is by using a program to mechanically check proofs. Interactive Theorem Provers (ITPs) allow users to generate proofs using a syntax which can be verified by a computer. While such proofs often originate in pen and paper form, by using an ITP one can ensure that all cases are properly covered,

and that the inductive principles used in a proof are correctly applied.

The goal of this thesis is to show how structural rules can be absorbed for progressively more complex sequent calculi. The main contributions of this work to the literature are proofs for calculi describing the Modal Logics S4 and S4.3. Each proof is accompanied by a corresponding formalised instance using the proof assistant Isabelle/HOL (2005) (henceforth referred to as just "Isabelle").

While the formal proofs for S4 and S4.3 are new, the formalisation itself is heavily based upon an existing code base produced by Jeremy Dawson and used previously in [Dawson and Goré 2006; Dawson and Goré 2010]. The extent of this work's contribution to machine checking proofs can be summarised as extending the existing Isabelle theories to deal with more complex logics.

In terms of the structure of this work, Chapters 2 and 3 provide an introduction to the theoretical side of proof systems for logic, with a focus on sequent calculi and the types of proofs necessary when developing efficient calculi. Included is a survey of the literature, a recap of existing methods for Classical Propositional Logic, and the motivation behind our work.

Chapter 4 of this thesis introduces the central ideas behind a general formalisation of sequent calculi and cut-admissibility lemmas using Isabelle. While intended to be accessible to as many readers as possible as a standalone work, the use of the Isabelle manual [Paulson 2003] will be required to understand the code in its entirety.

The core results of this text can be found in Chapter 5. The chapter outlines a number of new pen and paper proofs for structural rule admissibility, for calculi corresponding to the logics S4 and S4.3, and walks through the corresponding statements in the Isabelle formalisation. The machine checked proofs for each theorem can be found in the Appendix.

Chapter 6 differs from the rest of this text, and deals with an implementation of automated theorem prover based on a cut-free sequent calculus. In essence, it demonstrates an application of a calculus where the structural rules have been absorbed – performing efficient reasoning based on backward proof search.

Chapter 2

Background

This chapter introduces the notation and many constructs used throughout this work. Due to the extensive number of definitions utilised for proof theory, this chapter, along with Chapter 3 will also serve as references for the theory of this text. A short glossary is provided in the endmatter, but the glossary only re-iterates the most crucial definitions and some new notation introduced after these chapters. While intended to be mostly self-contained, some basic knowledge of logic and interactive theorem proving is recommended.

2.1 Logic

Typically, the symbols of a particular logic consist of atomic propositions, a set of connectives, and perhaps a number of quantifiers. Logical formulae are then recursively defined from these symbols. For example, in Backus-Naur form, the syntax for atoms p and formulae φ can be defined as follows:

$$p ::= p_0 \mid p_1 \mid p_2 \mid ...$$
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \Box\varphi$$

An example formula is $(p_0 \vee (p_1 \to p_2))$. The exact semantics of each logical operator can vary slightly depending on the logic considered. These semantics will be covered more formally in Section 2.1.1.

The subformula relation is defined in a similar recursive fashion: φ is a subformula of φ, and φ is a subformula of $\neg\varphi$, and for $\circ \in \{\wedge, \vee, \to\}$ if $\varphi_0 \circ \varphi_1$ is a subformula of φ, then so are φ_0 and φ_1 individually.

The logics investigated in this thesis will be propositional only. Thus the use of the quantifiers such as $\forall, \exists$ will primarily be restricted to details concerning formalisations in a proof assistant.

A few additional details on notation, many of which follow the example set by [Troelstra and Schwichtenberg 2000]:

- To avoid confusion with the modal operator $\Box$ the standard QED symbol is replaced by $\dashv$.

- Throughout this thesis φ, ψ as well as capital letters $A, B, \ldots$ will be used to indicate arbitrary formula.
- Γ, Δ, Σ will denote arbitrary multisets of formulae, unless otherwise stated.
- We will use "," as shorthand for multiset union. For example, the expression Γ, Δ stands for the multiset $\Gamma \cup \Delta$.
- Γ, A is the union of the (multisets) Γ and $\{A\}$.
- $\Gamma - A$ is the multiset difference $\Gamma \setminus \{A\}$. Note that $\{a, a, b\} \setminus \{a\} = \{a, b\}$ and $\{a, a, b\} \setminus \{c\} = \{a, a, b\}$.
- For any multiset $\Gamma = \{\varphi_1, \cdots, \varphi_n\}$ the multiset $\Box\Gamma$ represents $\{\Box\varphi_1, \cdots, \Box\varphi_n\}$.
- Outer parenthesis around formulae will generally be omitted. We will use the usual binding precedences to save on the use of parentheses. In particular, $\forall, \exists, \neg$ bind stronger than $\wedge, \vee$ which in turn bind more strongly than $\rightarrow, \leftrightarrow$. For example $A \vee B \rightarrow \neg C$ should be interpreted as $((A \vee B) \rightarrow (\neg C))$.
- We make use of the following abbreviations:

$$\begin{aligned} \bot &:= p_0 \wedge \neg p_0 \\ \top &:= \bot \rightarrow \bot \\ \varphi \leftrightarrow \psi &:= (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi) \\ \neg\varphi &:= \varphi \rightarrow \bot \\ \Diamond\varphi &:= \neg\Box\neg\varphi \end{aligned}$$

2.1.1 Validity

Each atomic proposition can be assigned a truth value. In the case of Classical Logic, the only allowed values are "True" (**T**) or "False" (**F**). The truth value of a formula can then be determined recursively according to the operators involved. Continuing the example for Classical Propositional Logic we have:

$$\varphi \wedge \psi = \begin{cases} \mathbf{T} & \text{if } \varphi = \mathbf{T} \text{ and } \psi = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\varphi \vee \psi = \begin{cases} \mathbf{T} & \text{if } \varphi = \mathbf{T} \text{ or } \psi = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\varphi \rightarrow \psi = \begin{cases} \mathbf{T} & \text{if } \varphi = \mathbf{F} \text{ or } \psi = \mathbf{T} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

$$\neg\varphi = \begin{cases} \mathbf{T} & \text{if } \varphi = \mathbf{F} \\ \mathbf{F} & \text{otherwise} \end{cases}$$

However, the logics discussed in this text cannot be adequately covered using just a simple evaluation procedure. The intuitionistic implication, as well as modal operators require a more complex model for semantic understanding.

Kripke frames and models are commonly used to represent the semantics of a particular logic. The definitions given in this work are based on those of [Goré 1994]. A Kripke frame is a pair $\langle W, R\rangle$ where W is a non-empty set (of possible worlds) and R is a binary relation over W. A Kripke model is a triple $M = \langle W, R, \vartheta\rangle$, where ϑ is a function mapping worlds and atomic variables to truth values. A forcing relation $\Vdash$ can then be defined – for a model M, world $w \in W$ and proposition p, let $M, w \Vdash p$ if $\vartheta(w, p) \mapsto \mathbf{T}$.

In addition, define $M \Vdash \varphi$ to hold if for every w we have $M, w \Vdash \varphi$. For a set of formulae Γ we say that $M \Vdash \Gamma$ if $M \Vdash \varphi$ for all $\varphi \in \Gamma$. In terms of global logical consequence, define $\Gamma \models \varphi$ if for all models M we have $M \models \Gamma$ implies $M \models \varphi$.

The semantics for Classical Propositional Logic can be expressed with a Kripke model using only one world w. In fact, the definition is very similar to the one given above:

- $M, w \Vdash \varphi \wedge \psi$ iff $w \Vdash \varphi$ and $w \Vdash \psi$
- $M, w \Vdash \varphi \vee \psi$ iff $w \Vdash \varphi$ or $w \Vdash \psi$
- $M, w \Vdash \varphi \to \psi$ iff $w \nVdash \varphi$ or $v \Vdash \psi$
- $M, w \Vdash \neg\varphi$ iff $w \nVdash \varphi$

Classical Modal logics build upon the above, by including the $\Box$ and $\Diamond$ operators:

- $M, w \Vdash \Box\varphi$ iff $\forall v \in W.\ wRv$ implies $v \Vdash \varphi$
- $M, w \Vdash \Diamond\varphi$ iff $\exists v \in W.\ wRv$ and $v \Vdash \varphi$

Note that these operators deal with potentially many worlds. There are a number of modal logics, including K, S4, S4.3, and S5, which are all variants of this, differing by their definition of the relation R.

For Intuitionistic Propositional Logic (**Int**), the relation R is reflexive and transitive. Furthermore, the valuation function θ is defined so that for an atomic proposition p, once $\vartheta(w, p) \mapsto \mathbf{T}$ for some world w, for all worlds v with wRv it must be that $\vartheta(v, p) \mapsto \mathbf{T}$. This property is *persistence*. However, unlike Classical logic, a proposition which is not true in a world w may become true later on (in some world reachable using R).

- $M, w \Vdash \varphi \wedge \psi$ iff $w \Vdash \varphi$ and $w \Vdash \psi$
- $M, w \Vdash \varphi \vee \psi$ iff $w \Vdash \varphi$ or $w \Vdash \psi$
- $M, w \Vdash \varphi \to \psi$ iff $\forall v \in W$, $w \leq v$ and $v \Vdash \varphi$ implies $v \Vdash \psi$
- $M, w \Vdash \neg\varphi$ iff $M, w \nVdash \varphi$

Given definitions of models and frames, a formula A is *valid in a model*, (W, R, ϑ), if for all worlds $w \in W$ it holds that $w \Vdash A$. A formulae A is *valid in a frame*, (W, R), if it is valid in all models based on (W, R).

2.1.2 Sequent Calculi

A sequent is an expression $\Gamma \vdash \Delta$ where both Δ and Γ are multisets of formulae. Γ is called the *antecedent*, while Δ is the *succedent*.

In relation to Kripke models, a sequent $\Gamma \vdash \Delta$ is *falsifiable* if there exists a model $\langle W, R, \vartheta \rangle$ and a world $w \in W$ where every element of Γ is true at w and every member of Δ is not true at w. A sequent is *valid* if it is not falsifiable. From a simplified perspective, a sequent $\Gamma \vdash \Delta$ can be considered valid when the conjunction of all formulae in Γ implies the disjunction of all formulae in Δ.

Sequent Rules A rule r, applicable either "downwards" from a number of sequents (forward proof) or "upwards" from a sequent (backward proof), has the form:

$$r \frac{\Gamma_1 \vdash \Delta_1 \qquad \Gamma_2 \vdash \Delta_2}{\Gamma_0 \vdash \Delta_0}$$

- Sequents above the line, in this case $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$, are called the *premises* of r and $\Gamma_0 \vdash \Delta_0$ is the *conclusion*. A rule may have zero or more premises, but always one conclusion.

- Within a rule, instances of Γ, Δ and Σ without any preceding logical operators will be referred to as the *context*[1]. The formulae not in the context of a rule conclusion will be referred to as *principal*, formula not within the context of premises are *side formulae*, and formula in the context of a rule application are *parametric*. For example, in the rule L$\wedge$ given in Figure 2.1, $A_0 \wedge A_1$ is principal. A_i is a side formula. Γ and Δ in both the premises and conclusion form the context.

- There are two main categories of sequent rules. *Logical* rules act upon a logical operator. A *structural* rule, on the other hand, manipulates sequents without modifying logical operators – the expressions for such rules generally do not contain logical connectives. Examples of both types of rules are given in Figure 2.1. Note that while axioms may not contain logical connectives, these are always considered logical rules.

- A rule is *sound* if whenever the premises are not falsifiable, then the conclusion is not falsifiable. That is, if the premises are valid, then the conclusion must be also. Alternatively, if the conclusion is falsifiable, then at least one of the premises must also be falsifiable.

A **sequent calculus** is a system of rules. An example calculus for Classical Propositional Logic is shown in Figure 2.1.

Derivations A *derivation* using a sequent calculus is a tree, with a conclusion as root (the bottommost sequent of a derivation tree) and where each internal node is an instance of a rule. The root sequent is also called the *endsequent*. A *sequent proof* is a

[1] For example, Γ by itself is considered part of the context of a rule. On the other hand $\Box\Gamma$ is principal.

Axioms

$$\text{id}\ \frac{}{A \vdash A} \qquad \text{L}\bot\ \frac{}{\bot \vdash}$$

Structural rules: Weakening (W) and Contraction (C)

$$\text{LW}\ \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \qquad \text{RW}\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}$$

$$\text{LC}\ \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \qquad \text{RC}\ \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}$$

Logical rules

$$\text{L}\wedge\ \frac{\Gamma, A_i \vdash \Delta}{\Gamma, A_0 \wedge A_1 \vdash \Delta}\ (i = 0, 1) \qquad \text{R}\wedge\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$$

$$\text{L}\vee\ \frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash A_i, \Delta}{\Gamma \vdash A_0 \vee A_1, \Delta}\ (i = 0, 1)$$

$$\text{L}\to\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \qquad \text{R}\to\ \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}$$

Figure 2.1: Sequent calculus G1cp.

derivation where every leaf has zero premises. Figure 2.2 gives an example of a proof using the G1cp calculus.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{id}\ \dfrac{}{p_0 \vdash p_0}}{\text{RW}\ \dfrac{p_0 \vdash p_0}{p_0 \vdash p_0, p_1}} \qquad \dfrac{\text{id}\ \dfrac{}{p_1 \vdash p_1}}{\text{LW}\ \dfrac{p_1 \vdash p_1}{p_0, p_1 \vdash p_1}}}{\text{L}\to\ \dfrac{p_0 \vdash p_0, p_1 \qquad p_0, p_1 \vdash p_1}{p_0, p_0 \to p_1 \vdash p_1}}}{\text{L}\wedge\ \dfrac{p_0, p_0 \to p_1 \vdash p_1}{p_0, p_0 \wedge (p_0 \to p_1) \vdash p_1}}}{\text{L}\wedge\ \dfrac{p_0 \wedge (p_0 \to p_1), p_0 \wedge (p_0 \to p_1) \vdash p_1}{\text{LC}\ \dfrac{p_0 \wedge (p_0 \to p_1), p_0 \wedge (p_0 \to p_1) \vdash p_1}{p_0 \wedge (p_0 \to p_1) \vdash p_1}}}}{\text{R}\to\ \dfrac{p_0 \wedge (p_0 \to p_1) \vdash p_1}{\vdash (p_0 \wedge (p_0 \to p_1)) \to p_1}}$$

Figure 2.2: A sequent proof using G1cp

During proof search – constructing a derivation – it is often the case that many rules can be applied to the same sequent. For example, given the sequent $A \wedge B, C \wedge D \vdash A \vee B$, there are choices to be made for which rule to apply, as well as the principal formulae to be used for that rule. In this case, using G1cp, the possible rules are R$\vee$ and L$\wedge$. The latter rule can also be applied with $A \wedge B$ principal, or with $C \wedge D$ principal. Automated theorem provers typically implement heuristics to perform these choices so that the resulting derivation is as small as possible.

- Generally in proof search, rules are applied "backwards", starting with a sequent as a conclusion and resulting in the premises. Branching occurs whenever a rule is applied with more than one premise.

- In a derivation, a non-invertible rule requires *backtracking*. Backtracking occurs when an application of a non-invertible rule r does not lead to a proof. This is undesirable as search must then return to the sequent corresponding to the conclusion of r. This conclusion is unprovable only if it can be shown that *all* potential premises (from all applicable rules) are unprovable.
- A rule is *syntactically invertible* if whenever the conclusion is derivable then the premises are also derivable. This can also be said for individual premises, that is, a premise is invertible if it can be derived whenever the conclusion is derivable.
- A sequent calculus has the *subformula property* if all formulae within any derivation using the calculus are subformulae of the root sequent. Typically this means that the premises of each rule must only contain formulae which are subformula of the conclusion.
- In terms of the validity of a formula φ, a sequent calculus is *sound* and *complete* for a logic when $\emptyset \models \varphi$ holds if and only if $\emptyset \vdash \varphi$ can be proved using the rules of the calculus.

The notion of derivations leads to other concepts about rules, most importantly that rules may be *admissible* or *derivable*. A sequent calculus rule R is said to be admissible if any sequent derived using a set of rules including R can also be derived by the same set of rules with R removed. A rule r is derivable within some calculus, if the conclusion of r can be derived without using r, given derivations of the premises of r.

2.1.3 Types of Reasoning

While both sequent calculi and Kripke models can be used to check the validity of formulae, the two differ significantly in their styles of reasoning. These are referred to as *syntactic* and *semantic* reasoning respectively. The main difference between the two can be inferred from their definitions. Sequent calculi transform syntactic expressions, a process which can be done by applying rules without knowing or understanding anything about the logic involved. On the other hand, constructing a model requires knowledge of truth values and how these interact both within and between worlds.

There exist other methods for performing both syntactic and semantic reasoning. For example, tableaux calculi and natural deduction systems are other examples of syntactic systems. In this text we only deal with sequent calculi and Kripke models.

Another important distinction should be made between *object-level* and *meta-level* proofs and reasoning. In the context of logical proofs, the former involves reasoning about formulae, while the latter involves reasoning about a proof system and the proofs that the system produces. For example, an automated reasoner based on a sequent calculus performs object-level reasoning – typically deciding whether a formula is valid. On the other hand, reasoning about the size of proofs generated by a sequent calculus constitutes meta-level reasoning.

To clearly separate the two styles of proof, and noting that derivations which are not sequent proofs are generally uninteresting, for the remainder of the text "proof(s)"

will refer to meta-theorems while "derivation(s)" is used to indicate object-level theorems (almost always sequent proofs). The main results of this work are meta-theorems about structural rule admissibility within sequent calculi.

2.2 Interactive Theorem Proving

There are two main types of theorem provers. We will refer to provers which only perform object-level reasoning as ATPs, and those which facilitate meta-level reasoning as ITPs. The difference between the two is subtle. The former merely take input objects, and output either a statement of validity or a proof. The latter allow humans to produce proofs and allow the modelling of concepts and systems, acting as an assistant which ensures the proofs and models are correctly stated.

Interactive theorem proving is then the process of using ITPs to create computer checked proofs. The motivation behind this process is that proofs developed with such software will not be susceptible to the human errors common in pen and paper proofs. Learning and using an ITPs is itself similar to the process of learning and coding with a new programming language.

ITPs themselves are almost always based on a certain logic (typically some higher-order logic), and often contain or use ATPs. The use of ATPs within such software allows for significant portions of proofs to be automated, particularly when proving identities or simplifying subgoals within a proof process [Paulson and Blanchette 2010]. The overall correctness of ITPs can be formally verified, but typically they are accepted as trustworthy given that their core code is heavily scrutinised.

2.2.1 Shallow and Deep Embeddings

In order to perform reasoning about a calculus using an ITP, one must first encode that calculus within the language of the ITP. There are two different ways of doing so, which are generally referred to in the literature as either "shallow" or "deep" embeddings. The difference between the two is somewhat analogous to the difference between object-level and meta-level reasoning.

Shallow embeddings represent formula directly in the inbuilt logic of an ITP. This allows for ease of reasoning using the embedded calculus, and potentially utilising the inbuilt ATPs – producing proofs of formulae using that calculus. This type of embedding is very useful for proving things in the logic, at an object-level.

Deep embeddings require constructing most, if not all, concepts from the ground up. In the case of sequent calculi, this means encoding things such as the syntax for formula, defining multisets, and the representation of sequents. This almost certainly requires more work than a shallow embedding, both in creating datatypes as well as in proving basic identities and theorems which can be used later on either manually, or possibly by an ATP.

The advantage of a deep embedding is the representation of proofs themselves. When modelling sequent calculi, this allows for the formalisation of, and reasoning

about, derivation trees. Subsequently, deep embeddings are often obligatory for proving results about the calculus itself, rather than results about formulae in the corresponding logic.

2.2.2 Isabelle

There exist a number of publicly available ITPs. Two were considered for this project, Abella and Isabelle. The main feature of Abella is its use of two levels of logic. Referred to as a "specification" and "reasoning" logic, these perform object and meta-level reasoning as described earlier. Rather than explicitly encoding deep embeddings, the advantage of Abella is its ability to shallowly encode a sequent calculus in the specification logic. Proving properties about the derivations produced using the specification logic can then be done using the reasoning logic. Unfortunately the specification logic of Abella does not allow multiple formulae in the succedent of a sequent, without going to a deep embedding. This in turn means losing the main advantage of Abella's two logic approach to reasoning.

Isabelle, on the other hand, is a much more complex prover than Abella. The documentation and features of the prover are considerable, generally requiring a rather long learning process. In comparison to Abella, however, there is a large repository of existing Isabelle files, created by Jeremy Dawson, dealing with sequent calculi and cut-admissibility. Rather than starting from scratch with Abella, and with the benefits of the two-leveled logic approach out of the question, the formalisations of this work are implemented in Isabelle/HOL (2005).

As an overview, proofs in Isabelle are based on *resolution*, using higher order unification [Paulson 2003]. Isabelle theorems are generally stated as goals using strings, and proofs carried out by applying tactics in succession. The proof assistant includes a number of automated reasoners, which can be called upon within proofs (using tactics) to simplify subgoals, or even prove some theorems automatically. These reasoners can be given sets of theorems as well as equalities to use, and can aid considerably in shortening the amount of time and effort required for a proof.

The code for the formalised results proven in this thesis builds upon the work produced by Jeremy Dawson over the years. This work has been referenced in various publications including [Dawson and Goré 2006] and [Dawson and Goré 2010], and provides a foundation for proofs involving sequent calculi. These include a plethora of definitions and lemmas to deal with multisets, sequents, rules and inductive reasoning. The extent and use of this pre-existing code base[2], which will be referred to as the "base formalisation", will be discussed in Chapter 4, and clear indications provided where new code has been developed.

Regrettably, these theories use the syntax of an older version of Isabelle, and so the formalisations are not compatible with the newest version (2013) of Isabelle. While attempts have been made to port the code, the effort required to do so would be enormous, especially considering certain features are no longer included in Isabelle 2013.

[2] Available at: `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/`

Chapter 3

Proof Theory

Proof theory emerged from Hilbert's attempt to produce "finitistic" proofs of the consistency of the axioms of mathematics [von Plato 2014]. While Gödel's incompleteness theorem prevented the program from attaining its original goals, significant efforts were made towards producing and analysing formal mathematical proofs. Gentzen's landmark paper [Gentzen 1935] was a key development in the analysis of logical systems, introducing sequent calculi as well as his Hauptsatz, cut-elimination, which results in a number of important properties.

Troelstra and Schwichtenberg [2000] divide proof theory into two parts: structural proof theory and interpretational proof theory. This work only investigates structural proof theory, with a focus on eliminating structural rules from sequent calculi by absorbing their effects into logical rules. Recall that structural rules are those which do not act on logical connectives. Poggiolesi [2011] makes a central distinction between "logical" and "structural" variants of sequent calculi. The latter contains explicit structural rules, while the former absorbs these into the logical rules. In this thesis, the motivation behind absorbing structural rules is the usefulness of logical variants of sequent calculi from a backward proof search perspective.

As a case study, consider the traditional (structural) calculus for Classical Propositional Logic, named G1cp (or LK as introduced by Gentzen). The version presented in Figure 3.1 is a minor variant of the standard calculus given in the literature [Troelstra and Schwichtenberg 2000]. The only difference is the non-inclusion of an axiom for $\bot$, instead $\bot$ is interpreted as $P \wedge \neg P$ for any atomic formula P.

Figure 3.2 is an example of a derivation (recalling that "derivation" is used rather than "sequent proof" to avoid confusion between sequent and meta-level proofs) using the G1cp calculus. The key points of interest in this derivation are the uses of Weakening as well as the instance of Contraction below the two applications of the L$\wedge$ rule. For an automated reasoner, rather than applying Weakening to obtain an instance of the id rule, it seems more efficient to instead use an id rule of the form:

$$\text{id}\ \frac{}{\Gamma, A \vdash A, \Delta}$$

This new rule can be thought to be the original id rule combined with a number of applications of either LW or RW so that there are now contexts Γ and Δ. A new id rule can now be applied without requiring explicit instances of the Weakening rules to reduce a sequent to the form $A \vdash A$ without context.

Axioms

$$\text{id}\ \frac{}{A \vdash A}$$

Structural rules: Weakening (W) and Contraction (C)

$$\text{LW}\ \frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \qquad \text{RW}\ \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, A}$$

$$\text{LC}\ \frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta} \qquad \text{RC}\ \frac{\Gamma \vdash \Delta, A, A}{\Gamma \vdash \Delta, A}$$

Logical rules

$$\text{L}\wedge\ \frac{\Gamma, A_i \vdash \Delta}{\Gamma, A_0 \wedge A_1 \vdash \Delta}\ (i = 0, 1) \qquad \text{R}\wedge\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}$$

$$\text{L}\vee\ \frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash A_i, \Delta}{\Gamma \vdash A_0 \vee A_1, \Delta}\ (i = 0, 1)$$

$$\text{L}\to\ \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta} \qquad \text{R}\to\ \frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}$$

Figure 3.1: Sequent calculus G1cp.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{id}\ \dfrac{}{p_0 \vdash p_0}}{\text{RW}\ \dfrac{p_0 \vdash p_0}{p_0 \vdash p_0, p_1}} \quad \dfrac{\text{id}\ \dfrac{}{p_1 \vdash p_1}}{\text{LW}\ \dfrac{p_1 \vdash p_1}{p_0, p_1 \vdash p_1}}}{\text{L}\to\ \dfrac{p_0 \vdash p_0, p_1 \qquad p_0, p_1 \vdash p_1}{p_0, p_0 \to p_1 \vdash p_1}}}{\text{L}\wedge\ \dfrac{p_0, p_0 \to p_1 \vdash p_1}{p_0, p_0 \wedge (p_0 \to p_1) \vdash p_1}}}{\text{L}\wedge\ \dfrac{p_0 \wedge (p_0 \to p_1), p_0 \wedge (p_0 \to p_1) \vdash p_1}{\text{LC}\ \dfrac{p_0 \wedge (p_0 \to p_1), p_0 \wedge (p_0 \to p_1) \vdash p_1}{p_0 \wedge (p_0 \to p_1) \vdash p_1}}}}{\text{R}\to\ \dfrac{p_0 \wedge (p_0 \to p_1) \vdash p_1}{\vdash (p_0 \wedge (p_0 \to p_1)) \to p_1}}$$

Figure 3.2: A derivation of $\vdash (p_0 \wedge (p_0 \to p_1)) \to p_1$ using G1cp

Similarly, one can postulate a new rule for L$\wedge$ that absorbs Contraction. Rather than projecting the principal conjunction ($A_0 \wedge A_1$) to one of its conjuncts (A_i where $i = 0, 1$), a new rule can be derived which keeps both conjuncts. The derivation of this rule uses LC and two instances of the original L$\wedge$ rule, exactly like in Figure3.2. Examining the R$\vee$ rule, the same process can be applied (using RC). The new rules are given below:

$$\text{L}\wedge\ \frac{\Gamma, A_0, A_1 \vdash \Delta}{\Gamma, A_0 \wedge A_1 \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash A_0, A_1, \Delta}{\Gamma \vdash A_0 \vee A_1 \Delta}$$

At this point, having replacing the original rules with the new ones, the structural rules may be removed altogether. The corresponding calculus is named G3cp in the literature (once again without the inclusion of an axiom for $\bot$). This calculus is presented in Figure 3.3. Note that the id rule is now only allowed to act on atomic propositions. This restriction is used in the literature to prove height-preserving properties,

Axioms

$$\text{id}\ \frac{}{\Gamma, P \vdash P, \Delta}\ P \text{ atomic}$$

Classical rules

$$\text{L}\wedge\ \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta} \qquad \text{R}\wedge\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}$$

$$\text{L}\vee\ \frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}$$

$$\text{L}\to\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \to \psi \vdash \Delta} \qquad \text{R}\to\ \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}$$

$$\text{L}\neg\ \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \qquad \text{R}\neg\ \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}$$

Figure 3.3: Sequent calculus G3cp.

$$\dfrac{\dfrac{\dfrac{\text{id}\ \dfrac{}{p_0 \vdash p_0, p_1} \qquad \text{id}\ \dfrac{}{p_0, p_1 \vdash p_1}}{\text{L}\to\ \dfrac{}{p_0, p_0 \to p_1 \vdash p_1}}}{\text{L}\wedge\ \dfrac{}{p_0 \wedge (p_0 \to p_1) \vdash p_1}}}{\text{R}\to\ \dfrac{}{\vdash (p_0 \wedge (p_0 \to p_1)) \to p_1}}$$

Figure 3.4: A derivation of $\vdash (p_0 \wedge (p_0 \to p_1)) \to p_1$ using G3cp

discussed in greater detail in Section 3.2.

The derivation for the same endsequent as Figure 3.2, now using the rules of G3cp, is given in Figure 3.4. The new derivation is considerably smaller compared to the original. The main benefit, however, is that the structural rules are not present. The importance of this for automated reasoners is highlighted in Section 3.1.

G3cp is a logical calculus, which has absorbed the structural rules of G1cp. Obviously, it must be required that G3cp can actually derive the same sequents as G1cp. Proving this requires showing that Weakening and Contraction have been absorbed correctly. Formally, this is the *admissibility* of the Weakening and Contraction rules. In general, a rule r will be said to be *admissible* for a calculus $\mathbf{S}$ if for all instances of r, whenever the premises of r are derivable in $\mathbf{S}$, then so is the conclusion. To avoid the trivial case, it will also be required that r is not a rule within $\mathbf{S}$. Note: in addition to "admissible", this is also stated as "$\mathbf{S}$ is *closed* under r" in the literature.

Weakening is said to be admissible for a calculus if the following meta-theorem can be proven: given a derivation of some arbitrary $\Gamma \vdash \Delta$, which does not use explicit Weakening rules, then $\Gamma_W, \Gamma \vdash \Delta, \Delta_W$ is also derivable without using explicit Weakening rules, for all multisets Γ_W and Δ_W.

Contraction is admissible for a calculus if: given that $\Gamma_C, \Gamma_C, \Gamma \vdash \Delta, \Delta_C, \Delta_C$ is derivable, then $\Gamma_C, \Gamma \vdash \Delta, \Delta_C$ is also derivable, all without explicit Contraction rules.

Cut is a special structural rule with two premises, which has not yet been mentioned. From a forward perspective, the rule eliminates an occurrence of a formula A

from the antecedent of one premise and the succedent of the other. There exist *context-sharing* (*additive*) and *context-independent* (*non-sharing*, *context-free*, *multiplicative*) versions of rules in the literature [Troelstra and Schwichtenberg 2000]. The distinction between these versions is important for resource sensitive logics. For the logics discussed in this work, the differences do not matter. The Cut rule which will be used, and referred to as "Cut", is the context-independent version:

$$\text{Cut}\ \frac{\Gamma_L \vdash \Delta_L, A \qquad A, \Gamma_R \vdash \Delta_R}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

The three rules, Cut, Weakening, Contraction as given above will be collectively referred to as *the* structural rules within this work.

Like admissibility for Weakening and Contraction, Cut is admissible for some calculus if: given two derivations of $\Gamma_L \vdash \Delta_L, A$ and $A, \Gamma_R \vdash \Delta_R$ without using Cut, a derivation of $\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R$ exists, also without using Cut.

The idea behind a sequent calculus is to provide a mechanical process of checking or creating valid formula within a particular logic. To be useful, such a calculus must be proven to correctly describe the logic involved. Formally, a calculus should be both sound and complete with respect to its underlying logic. The Cut rule, along with the other structural rules, is used when proving equivalence of natural deduction and sequent calculi systems. This in turn allows for proofs of completeness of a sequent calculus for its particular logic. The existence of the Cut rule, or its admissibility, is then a key property for sequent calculi. Indeed, Troelstra and Schwichtenberg state that "in order to prove that the various cutfree systems are adequate for our standard logics all we need to know is that these systems are closed under Cut".

3.1 Motivation for the Admissibility of Structural Rules

The admissibility of structural rules gives rise to a number of useful properties and consequences. Perhaps the most important consequence is the subformula property, which typically follows immediately given a calculus consisting only of the axiom and logical rules. The property itself is both important for automated reasoning as well as allowing elegant proofs for decidability [Poggiolesi 2011]. Cut-elimination is also commonly used as a foundation for proving interpolation theorems. The importance of these properties, amongst others, are given in [Troelstra and Schwichtenberg 2000].

The most important property for automated reasoning is the subformula property. During backward proof search, it is obviously undesirable to create formulae which are not relevant to the endsequent. The use of an explicit Cut rule within a calculus thus significantly detracts from the efficiency of reasoners applying backward proof search. In addition to the problems posed by allowing the creation of arbitrary new formulae, the rule may also be applied at any point of a proof tree, both leading to branching (via the creation of two premises) as well as necessitating heuristics in order not to be applied infinitely.

The other structural rules also pose problems for proof search. The main detriment of Weakening is that the rule may alter a provable conclusion into an unprovable

premise. This in turn can very easily result in wasted search effort – the exploration of the unprovable sequent – before the search procedure is forced to backtrack to the original sequent. One may naïvely consider simply never applying the Weakening rule. Indeed, the rule, considered individually, seems almost entirely useless from a backwards search perspective, as it can potentially remove important formula, while only offering little in terms of search space reduction.

The issue occurs in combination with the other rules of a calculus, which may not be applicable without applying Weakening beforehand. The example of Figure 3.2 demonstrates two instances where Weakening is required in order to apply the id rule of G1cp. Thus, within an automated procedure, the use of Weakening is often compulsory in order to terminate proof search.

For some calculi, such as G1cp, the Weakening rule is only required before axioms. However, the problem is exacerbated for more complex calculi with rules which do not allow arbitrary Γ and Δ in their conclusions. For example, the R$\Box$ rule in the system G1s for S4 (shown below), also cannot proceed without Weakening.

$$\text{R}\Box\ \frac{\Box\Gamma \vdash B, \Delta}{\Box\Gamma \vdash \Box B, \Delta}$$

Contraction is problematic in a manner similar to Cut, as a rule which is applicable backwards to any sequent and at any point of a search procedure. Like Cut, it is difficult to implement simple heuristics to limit the applications of Contraction. The necessary number of copies of a particular formulae cannot be bounded in an a priori fashion [Negri 2011], and hence the number of necessary backwards applications of the Contraction rule cannot be bounded during proof search. A naïve prover could potentially apply the Contraction rule indefinitely during proof search. Even without infinite applications, the manner in which contraction allows duplicating formulae arbitrarily easily leads to extraneous memory usage.

Clearly, the inclusion of structural rules during automated search can easily result in wasted exploration and, without appropriate heuristics, search may not terminate.

It should be noted that structural rules are not the only rules which lead to inefficient calculi. For example, the L$\to$ rule in the calculus G3i[1] maintains a copy of the principal formulae in one premise. Without loop-checking this rule can thus also lead to non-termination of backward proof search. There exist techniques to improve the efficiency of logical rules, see [Dyckhoff 1992] for the above problem, but these generally apply after the structural rules have already been absorbed. The problem posed by structural rules is also far more general.

$$\text{L}\to\text{(G3i)}\ \frac{\Gamma, A \to B \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \to B \vdash C}$$

[1] G3i is not shown in this work. Neither is G4i, the calculus corresponding to the solution by Dyckhoff. See, for example, [Troelstra and Schwichtenberg 2000] for the full calculi.

3.2 Proving the Admissibility of Structural Rules

There exist two main methods of arriving at a cut-free sequent calculus, either from a syntactic proof of the admissibility of Cut, or by a semantic analysis of the calculus and its corresponding logic. This text deals only with the syntactic style. While both styles of proof are commonly used in the literature [Troelstra and Schwichtenberg 2000], semantic proofs require a deep understanding of the semantics of the logic involved, and often require the use of concepts such as Hintikka structures, Henkin witnesses, or phase spaces [Clouston and Goré 2014; Hermant 2005; Okada 2002] in addition to the rules and derivations of the calculus involved. A syntactic proof, on the other hand, deals only with the rules and the structure of derivations. The self-contained nature of syntactic methods are then arguably more receptive to machine formalisation and formal verification.

A number of definitions (mostly following the nomenclature of [Troelstra and Schwichtenberg 2000]) will be used when describing syntactic proofs:

- In this text, capitalisation of the word "Cut" refers to the generic rule, "cut" in lowers case refers to an instance of the rule within a derivation tree.
- The principal formula of a cut, A, is called the **cut-formula**.
- The premises of a cut will also be referred to as the **cut-sequents**.
- Recall the definitions of principal and parametric formulae. A cut-formula is **principal** on the left (or right) if the formula is principal in the rule above the left (right) premise of the cut. If the cut-formula is not principal on some side, then it is **parametric** for that side.
- The **height** or equivalently **depth** of a tree T is the maximum length of any branch in T. The length of a branch is the number of nodes in the branch minus 1. The depth of a tree is also denoted as $|T|$.
- The **level** of a cut is the sum of the depths of the deductions of its premises.
- The **rank** of (a cut on) a formula A, interpreted as a tree, is given by $|A| + 1$.

Distinctions must be made between *proofs* of closure under Cut, cut-admissibility and cut-elimination. In this text, a proof of closure under Cut will be used to refer to a semantic analysis proving the admissibility of Cut. A proof of cut-admissibility will refer to a syntactic proof which takes two cut-free derivations (of two potential cut-sequents) and transforms these into another cut-free derivation (of the corresponding conclusion). Cut-elimination, on the other hand, is a syntactic proof which takes derivations that include explicit uses of the Cut rule, and then transforms these into derivations without Cut. These typically do so by successively removing the lowest level cut with maximal rank. While differing significantly in their approach, all three styles of proofs inevitably show that the rule Cut is admissible. In this text, both cut-admissibility and cut-elimination proofs are discussed.

Generally, syntactic proofs proceed by induction on the height of a derivation [Troelstra and Schwichtenberg 2000; Negri and von Plato 2001]. This requires proving that the property in question holds for derivations with unit height (axioms) and then, for every possible rule leading to a derivation with height $n+1$, the property follows either directly or by using the induction hypothesis on derivations with height n (typically the premise(s) of the last rule application).

As a simple example, consider an inductive proof of height-preserving weakening-admissibility for the calculus G3cp, shown in Figure 3.3.

Lemma 3.2.1 (weakening-admissibility)**.** *Given a derivation of* $\Gamma \vdash \Delta$ *with height* n*, then there is a derivation of* $\Gamma_W, \Gamma \vdash \Delta, \Delta_W$ *with height* n*.*

Proof. The axiom case is trivial, as the id rule allows arbitrary contexts. If Γ and Δ is an axiom, then so is $\Gamma_W, \Gamma \vdash \Delta, \Delta_W$. For the inductive step, let us examine the case where the last rule applied is L$\wedge$. The derivation is then

$$\mathrm{L}\wedge \dfrac{\begin{array}{c}\Pi \\ \Gamma, \varphi, \psi \vdash \Delta\end{array}}{\Gamma, \varphi \wedge \psi \vdash \Delta}$$

Here Π represents some derivation leading down to the premise of the L$\wedge$ rule, a premise which has height n. Using the induction hypothesis on this premise then gives the derivability of $\Gamma_W, \Gamma, \varphi, \psi \vdash \Delta, \Delta_W$ using a derivation with height n. Applying a new instance of the L$\wedge$ rule (where the context of the rule is now $\Gamma_W, \Gamma \vdash \Delta, \Delta_W$) then results in a derivation of the desired endsequent, with depth $n + 1$:

$$\mathrm{L}\wedge \dfrac{\Gamma_W, \Gamma, \varphi, \psi \vdash \Delta, \Delta_W}{\Gamma_W, \Gamma, \varphi \wedge \psi \vdash \Delta, \Delta_W}$$

As all rules in G3cp allow any contexts, the process used for L$\wedge$ can be applied for all other rules. With the addendum that for rules with multiple premises, the inductive hypothesis must be applied multiple times to introduce Γ_W and Δ_W in all premises. $\dashv$

Contraction-admissibility is more complex to prove, and is made significantly easier by first proving inversion lemmas. For G3cp, these lemmas state that if the conclusion of a rule is derivable (with height n), then the premises are also derivable (with height n). The proofs of invertibility for each rule proceed like the proof of weakening-admissibility, but now the inductive step must consider principal and parametric cases.

As a representative proof, the following lemma describes height-preserving inversion for L$\vee$ within G3cp.

Lemma 3.2.2 (invertibility of L$\vee$)**.** *If* $\Gamma, \varphi \vee \psi \vdash \Delta$ *is derivable, then so are* $\Gamma, \varphi \vdash \Delta$ *and* $\Gamma, \psi \vdash \Delta$*, with derivations of height* n*.*

Proof. If the original sequent is an axiom, then $\varphi \vee \psi$ cannot be principal, as it is not atomic. It must be that $\Gamma \vdash \Delta$ is an axiom, and so the premises are axioms as well.

For the inductive step, if $\varphi \vee \psi$ is principal in the last rule applied, then the derivation must have ended with the L$\vee$ rule, in which case the premises are known to be derivable:

$$\mathrm{L}\vee \frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta}$$

If $\varphi \vee \psi$ was parametric, apply the induction hypothesis on the premise(s) of the last rule, then apply a new instance of the same rule to obtain $\Gamma, \varphi \vdash \Delta$ and $\Gamma, \psi \vdash \Delta$. ⊣

Given inversion lemmas, height-preserving contraction-admissibility for G3cp can then be proved. To simplify the case analysis, this is typically done by first considering Contractions on one formula at a time (as in the rules LC and RC of G1cp).

Lemma 3.2.3 (admissibility of LC and RC)**.** *If $\Gamma, A, A \vdash \Delta$ is derivable, using a derivation with height n, then $\Gamma, A \vdash \Delta$ is derivable with height n. Similarly for $\Gamma \vdash \Delta, A, A$ and $\Gamma \vdash \Delta, A$.*

Proof. If $\Gamma, A, A \vdash \Delta$ is an axiom, and one A is principal, then $A \in \Delta$ and so $\Gamma, A \vdash \Delta$ is an axiom. If both copies of A are parametric, then $\Gamma \vdash \Delta$ is an axiom and the result follows. The axiom case is symmetric for $\Gamma \vdash \Delta, A, A$.

For the inductive step, if A is parametric the induction hypothesis can be applied to the premise. Otherwise, if A is principal, we show the cases when an implication rule is applied. Consider when the last rule applied is L$\to$, and $A = \varphi \to \psi$:

$$\mathrm{L}\to \frac{\begin{array}{cc}\Pi_L & \Pi_R \\ \Gamma, \varphi \to \psi \vdash \varphi, \Delta & \Gamma, \varphi \to \psi, \psi \vdash \Delta\end{array}}{\Gamma, \varphi \to \psi, \varphi \to \psi \vdash \Delta}$$

Applying depth-preserving invertibility of the L$\to$ rule gives derivations of height n for $\Gamma \vdash \varphi, \varphi, \Delta$ and $\Gamma, \psi, \psi \vdash \Delta$. The induction hypothesis can be applied on these to produce derivations of $\Gamma \vdash \varphi, \Delta$ and $\Gamma, \psi \vdash \Delta$, from which a new instance of the L$\to$ rule gives a derivation with height $n + 1$ of $\Gamma, \varphi \to \psi \vdash \Delta$.

When contracting in the succedent, and the last rule is R$\to$ the derivation is:

$$\mathrm{R}\to \frac{\begin{array}{c}\Pi \\ \Gamma, \varphi \vdash \psi, \Delta, \varphi \to \psi\end{array}}{\Gamma \vdash \Delta, \varphi \to \psi, \varphi \to \psi}$$

Once again, depth-preserving inversion gives a derivation of $\Gamma, \varphi, \varphi \vdash \psi, \psi, \Delta$ with height n. Applying the induction hypothesis twice provides a derivation of height n where the sequent no longer contains duplicates ($\Gamma, \varphi \vdash \psi, \Delta$). From this, another instance of R$\to$ concludes the proof. ⊣

Lemma 3.2.4 (contraction-admissibility)**.** *If $\Gamma_C, \Gamma_C, \Gamma \vdash \Delta, \Delta_C, \Delta_C$ is derivable then so is $\Gamma_C, \Gamma \vdash \Delta, \Delta_C$, all with height n.*

Proof. By an induction on the multisets Γ_C and Δ_C, repeatedly applying the results from Lemma 3.2.3. ⊣

As shown above, an induction principle on height alone is sufficient to prove the admissibility of Weakening, Contraction and inversion lemmas for G3cp. However, for cut-elimination, the standard proofs in the literature require a stronger nested induction on both the level of a cut and the rank of the cut. The resulting double induction then proceeds by replacing an original maximal rank cut of lowest level with one or more of lesser height, or rank, continuing until a base case is reached and the derivation can be transformed directly into one without a cut. Note that the resulting derivation produced by the overall transformation are almost always larger than the original, with the best known upper bounds for G1cp and G3cp still exponential in the height of original derivation [Buss 2012].

Unfortunately, proofs of cut-admissibility or cut-elimination are highly technical. Proofs for the inductive step must consider a large number of cases, for all possible rules applied above both premises of the cut, in combination with whether the cut-formula is principal or parametric on either side. The transformations for many cases may also require the admissibility of Weakening and Contraction. In the literature, the standard proof order (for a calculus without any structural rules[2]) is first proving weakening-admissibility, then inversion lemmas, then contraction-admissibility, and finally cut-elimination.

Recall that in cut-elimination, the original derivations contain applications of the Cut rule. The proof itself provides an algorithm to eliminate one instance of a cut at a time. The cases in a cut-elimination proof can be divided into three general classes – one or more of the premises of the cut is an axiom, the cut-formula is parametric in the last rule applied for at least one premise, the cut formula is principal in both premises. To provide a general understanding of how a cut-elimination proof proceeds, we give an example of one case of each class (still using G3cp).

Lemma 3.2.5. *Cut-elimination holds for G3cp.*

Proof. By successively eliminating the lowest level cut of maximal rank. The three key cases follow:

1. Axiom

To be an axiom, the cut-formula must be atomic. The original cut has form:

$$\text{Cut}\,\frac{\begin{array}{cc}\Pi_L & \\ \Gamma_L \vdash \Delta_L, P & \text{id}\,\dfrac{}{P, \Gamma_R \vdash \Delta_R}\end{array}}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

No transformations are necessary. There are two sub-cases to consider.

- If P is principal in the id rule, then to be an axiom it must be that $P \in \Delta_R$, hence the endsequent can be obtained from the left premise of the cut, $\Gamma_L \vdash \Delta_L, P$, by weakening in Γ_R and $\Delta_R \setminus \{P\}$. Note that the left premise must be cut-free, given that P is atomic, as the elimination always chooses the lowest level cut of maximal rank.

[2]For calculi which contain Weakening or Contraction as explicit rules, those rules can simply be applied within the cut-elimination proof.

- If P is parametric in the id rule, then $\Gamma_R \vdash \Delta_R$ is an axiom and is hence derivable. The endsequent can then be obtained by weakening in Γ_L and Δ_R.

2. Parametric on one side

If A is parametric on the right (the transformation is symmetric when parametric on the left), in an instance of a rule "R" then the cut has the form:

$$\text{Cut}\,\dfrac{\dfrac{\Pi_L}{\Gamma_L \vdash \Delta_L, A} \qquad \text{R}\,\dfrac{\dfrac{\Pi_R}{A, \Gamma'_R \vdash \Delta'_R}}{A, \Gamma_R \vdash \Delta_R}}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

This is transformed into a derivation with a new cut of lower level, which can then be eliminated using the induction hypothesis.

$$\dfrac{\dfrac{\Pi_L}{\Gamma_L \vdash \Delta_L, A} \qquad \dfrac{\Pi_R}{A, \Gamma'_R \vdash \Delta'_R}}{\text{R}\,\dfrac{\Gamma_L, \Gamma'_R \vdash \Delta_L, \Delta'_R}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}}\,\text{Cut}$$

Here, only one premise of R is shown explicitly. If the rule has multiple premises, the resulting transformation will require a new lower level cut for each of these. If R is a cut itself, the new instance of R will be a cut of smaller rank (as the cut-elimination strategy eliminates the lowest leveled cut of maximal rank first). Note that this parametric step can be applied to all rules in G3cp. However, this particular transformation cannot generally be applied to all calculi. For more complex rules which disallow arbitrary contexts in their premises, the new instance of rule R may not be applicable with the addition of Γ_L and Δ_R in the premise(s). This is described in greater detail in Section 4.2.4.

3. Principal both sides

Consider when the premises are principal instances of the R$\wedge$ and L$\wedge$ rule, $A = \varphi \wedge \psi$:

$$\dfrac{\text{R}\wedge\,\dfrac{\dfrac{\Pi^a_L}{\Gamma_L \vdash \Delta_L, \varphi} \qquad \dfrac{\Pi^b_L}{\Gamma_L \vdash \Delta_L, \psi}}{\Gamma_L \vdash \Delta_L, \varphi \wedge \psi} \qquad \text{L}\wedge\,\dfrac{\dfrac{\Pi_R}{\varphi, \psi, \Gamma_R \vdash \Delta_R}}{\varphi \wedge \psi, \Gamma_R \vdash \Delta_R}}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\,\text{Cut}$$

This is transformed using the inductive hypothesis for cuts of smaller rank, as well as an appeal to contraction-admissibility:

$$\dfrac{\text{Cut on } \psi\,\dfrac{\dfrac{\Pi^b_L}{\Gamma_L \vdash \Delta_L, \psi} \qquad \text{Cut on } \varphi\,\dfrac{\dfrac{\Pi^a_L}{\Gamma_L \vdash \Delta_L, \varphi} \qquad \dfrac{\Pi_R}{\varphi, \psi, \Gamma_R \vdash \Delta_R}}{\psi, \Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}}{\Gamma_L, \Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_L, \Delta_R}}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\,\text{Contraction-admissibility}$$

Unlike in the axiom and parametric cases, there is no standard transformation which applies to the third case. The construction of the new derivation varies considerably depending on the actual rules above the original cut. $\dashv$

The admissibility proofs of Weakening and Contraction, as well as the proof of cut-elimination, above are provided as outlines of the standard procedure followed in the literature. For a more detailed coverage of the proofs, we defer to the comprehensive literature for G3cp, see for example Troelstra and Schwichtenberg [2000] and Negri and von Plato [2001].

The proofs used later in this text do not entirely mimic the traditional methods. Due to the emphasis placed on efficiency of search, the requirement of the axiom rule to be atomic in the standard presentation of G3cp is objectionable. Using logical rules to break down a formula into atoms extend backward proof search unnecessarily, when compared to using an axiom rule which allows for any formula. Thus for the remainder of this text, the id rule for G3cp will be allowed to apply to any formula, not only atoms.

A consequence of lifting the atomic restriction on axioms is that strictly height-preserving proofs are no longer possible. In particular, the axiom case in the proof of Lemma 3.2.2 no longer allows for height-preserving invertibility. As we are not interested in bounding the size of cut-elimination proofs, but only in efficient logical sequent calculi, this is an acceptable trade-off. More relevant to the proofs of this text is the fact that the removal of height-preservation as a condition may require an additional induction principle to compensate. This is most obvious in a contraction-admissibility proof, where the original method applies height-preserving invertibility followed by an application of the induction hypothesis on height. If inversion is not height-preserving, the contraction-admissibility proof must resort to using another induction on rank (in a similar manner to the double induction used for Cut), in order to remove the duplicate formula resulting from inversion.

While this text only considers Weakening, Contraction and Cut as structural rules, another structural rule exists which behaves similarly to Cut. Named "Mix", the rule essentially removes multiple copies of a mix-formula in a similar way to Cut. That is, n copies of a formula A in the succedent of one sequent may be mixed with m copies of A in the antecedent of another sequent.

$$\text{Mix}\ \frac{\Gamma_L \vdash \Delta_L, A^n \qquad A^m, \Gamma_R \vdash \Delta_R}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

In Gentzen's original paper [Gentzen 1935], where the sequent calculi contain explicit Weakening and Contraction rules, mix-elimination is actually proved as opposed to cut-elimination. For the case where a contraction leads to the derivation of a cut-sequent, cut-elimination does not allow for a simple transformation into a new derivation with a lower level cut. The usual transformation of

$$\text{Cut}\ \frac{\begin{array}{c}\Pi_L \\ \Gamma_L \vdash \Delta_L, A\end{array} \qquad \text{LC}\ \dfrac{\begin{array}{c}\Pi_R \\ A, A, \Gamma_R \vdash \Delta_R\end{array}}{A, \Gamma_R \vdash \Delta_R}}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

into

$$\text{Cut}\,\dfrac{\dfrac{\Pi_L}{\Gamma_L \vdash \Delta_L, A}\quad \text{Cut}\,\dfrac{\dfrac{\Pi_L}{\Gamma_L \vdash \Delta_L, A}\quad \dfrac{\Pi_R}{A, A, \Gamma_R \vdash \Delta_R}}{A, \Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}}{\text{LC}\,\dfrac{\Gamma_L, \Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}{\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}}$$

results in a new derivation which does not reduce the level or rank of the bottommost cut. However, using Mix allows the elimination of both contraction-formulae at the same time.

Mix is undesirable compared to Cut, as it implies that one copy of a formula may be "worth" multiplies of another. While for many uses of sequent calculi this may not be an issue, proving admissibility of Mix rather than Cut is insufficient to show the adequacy of a calculi in modelling a resource sensitive system. Recently it has been shown in Isabelle by Jeremy Dawson that Mix is not actually necessary to deal with Contraction. In the formalised proof of cut-elimination the case is handled by tracing up the derivation above the Contraction and performing transformations based on the first non-Contraction rule [Dawson 2014a].

In this text only cut-elimination is examined; in fact, as we desire calculi without explicit structural rules, the issue with Contraction never materialises.

3.3 Related Work

Due to the importance of Cut the development of most calculi or logics are typically accompanied by some discussion of cut-elimination or closure under Cut. As a result, the literature is too large to be adequately discussed in its entirety. Even when limited to syntactic admissibility proofs, there remains too vast a number of publications. It should be noted that the previous sections of this chapter can be considered a brief overview of origins of proof theory, as well as covering the traditional method for proving syntactic admissibility. The introductions to the actual calculi analysed in this work, within Chapter 5, also serve to describe the related work for each logic in a more specific context. This section then focuses only on a selected number of publications that are particularly pertinent to formalised proofs.

Due to the recent introduction of ITPs when compared to the history of proof theory, it is no wonder that the majority of cut-admissibility proofs in the literature are given using pen and paper. The potential issue with such proofs arises from a combination of human error, journal page limits, and the technical, yet repetitive, case analysis required for cut-elimination. People are notoriously bad at doing repetitive tasks, with minor alterations between cases, and phrases such as "the other case is similar" are commonly employed within proofs [Goré 2009]. The problem is exacerbated given length limits on papers, but even in textbooks cases are often omitted, typically left as exercises (see for example the cut-elimination proof for G3s by Troelstra and Schwichtenberg [2000]).

Examples of error-prone or incomplete proofs are given by the history of cut-elimination for the provability logic GL, mostly using variations of the sequent cal-

culus GLS. Overviews of the issue itself have appeared within a number of papers, namely [Goré and Ramanayake 2008; Goré 2009; Dawson and Goré 2010]. An initial "proof" of cut-elimination for GL was given by Leivant [1981] for a sequent calculus based on sets. This turned out to be incorrect, as shown using a counter-example by Valentini [1983]. Valentini proceeded to provide a new proof (albeit very terse and informal), still based on sets, using a third induction parameter named "width". But then Moen [2001] claimed that Valentini's method does not terminate for multiset-based sequents, a controversial claim which remained unresolved until 2008.

Two alternative proofs were published for GL, based on multisets, by Negri [2005] and Mints [2005]. The former utilises a labelled sequent calculus, which draws upon the semantics of the logic within the calculus rules, and so cannot be considered an entirely proof theoretic solution. The latter proof follows the standard syntactical method, but does not give transformations when Contraction is applied above a premise of cut. Neither of these proofs specifically deal with the claim surrounding sets and multisets, and so to address Moen's claim, Goré and Ramanayake [2008] present a translation of Valentini's arguments and give a full pen and paper proof which deals with the complications for multisets. The paper also points out a flaw in Moen's reasoning, showing that Valentini's proof is "mostly okay".

As illustrated by the history for GL, contentions for cut-elimination proofs can easily take several papers spanning multiple years to resolve. Machine-checked proofs offer one avenue for settling such debates without such a drawn out process. Indeed, a formalised proof using Isabelle, has been developed for the sequent calculus GLS [Dawson and Goré 2010], based on the construction by Goré and Ramanayake, putting the issue to rest once and for all.

While machine checked proofs can be developed independently, typically they are based on pen and paper proofs with perhaps minor alterations. As ITPs require every possibility in a case analysis to be fully fleshed out, as well as ensuring that inductive principles are correctly applied, they are particularly useful for both verifying as well as identifying issues within pen and paper lemmas. Examples of formalisations used to verify pen and paper proofs include [Dawson and Goré 2010; Herbelin and Lee 2009; Tews 2013].

A common trend among the re-examination of pen and paper proofs, is the discovery of errors in the proof itself, but that the overall lemmas or theorems remain correct. As an example, Tews [2013] describes a formalisation of Coalgebraic logics, using the proof assistant Coq, based on the pen and paper proofs of Pattinson and Schröder [2010]. This formalisation uncovered four errors, beyond nit-picking or those of a typographical nature, in the original work. However, with some modifications to the proofs, the actual statement of the corresponding lemmas were shown to hold.

As another benefit of ITPs, Tews [2013] notes that the nature of a number of proof assistants can also allow for the extraction of verified programs. In the context of cut-elimination, these can potentially include automated theorem provers (following a completeness proof) or programs to compute interpolants (using cut-elimination as a basis for proving interpolation). The only real downside to using ITPs is the effort necessary to fully formalise calculi or logics. Dawson's Isabelle code base, which

includes the formalised cut-elimination proof for GLS, has about 2000 theorems, and over 20,000 lines of Isabelle code. Tews' Coq formalisation of Coalgebraic Logics is similarly large, containing 1300 theorems and 36,000 lines of Coq code.

As an aside, it should be noted that most proofs of cut-elimination are calculi specific, but there are also publications which take a more general approach. These investigate conditions or properties, leading to cut-elimination, if satisfied by a sequent calculus. While most of the work in this area has been for display or conditional logics [Belnap 1982; Goré 1998; Wansing 1995; Pattinson and Schröder 2009], a proof of cut-elimination for a general class of propositional logics is presented by Rasga [2005]. This proof applies for all calculi which satisfy certain properties, named as "cut suitable conditions". One would imagine that a formalisation of this would significantly simplify the development of proofs for future calculi.

Unfortunately, the conditions of Rasga pose fairly significant limits on the structure of rules in any potential calculi. In terms of automated reasoning, one of the cut suitable conditions is that an axiom rule must either have no context or an explicit Weakening rule must be present in the calculus. While perhaps acceptable for forward search, this is an essentially insurmountable drawback for backward proof search. Redundant formula are often present throughout the search space, which would prevent the application of any axiom rule disallowing contexts, unless Weakening is also included as an explicit rule.

Chapter 4

Isabelle Formalisation

This chapter outlines the complexities involved with formalising logics in the language of a proof assistant. While this text only provides a concrete formalisation in Isabelle (2005) syntax, many of the ideas presented here could potentially be extended to other proof assistants. The intended goal is to have a relatively general base formalisation of the necessary datatypes and inductive properties required to prove cut-elimination for a wide range of calculi.

Section 4.1 introduces the basic ideas required to formally represent sequent calculi. Progressively, this leads into discussions of some of the requirements of more complex logics and how these are dealt with in the corresponding Isabelle formalisation. The inductive principles used to prove admissibility results are given in Section 4.2. Much of the Isabelle code in this chapter has been described in past publications [Dawson and Goré 2006; Dawson and Goré 2010]. The various additional features introduced by this thesis will be highlighted whenever appropriate.

In terms of notation, theorems will generally be stated in English, and named as they are in the corresponding Isabelle proofs.

4.1 Representation

A combination of deep and shallow embeddings will be used. While the actual formalisation of formulae, sequents and rules are "deeply embedded" within the metalogic of Isabelle, the use of "shallow derivations" simplifies many proofs. Explicit derivation trees (deep derivations) are also used, but only when absolutely required within the cut-admissibility proof.

4.1.1 Sequents and Rules

Formula are defined recursively as a new datatype. This representation has the advantage of being very general, allowing formula connectives to be specified purely as a string and allowing any number of operands for connectives though the use of a list.

```
datatype formula =
      FC string (formula list) (* formula connective *)
    | FV string                (* formula variable *)
    | PP string                (* primitive proposition *)
```

The actual strings used to correspond to logical operators are given below:

Connective	$\wedge$	$\vee$	$\rightarrow$	$\neg$	$\Box$
String	`&&`	`v`	`->`	`--`	`Box`

Multisets in the base formalisation are defined slightly differently to the standard multiset theory provided with Isabelle. Notably, ordering on multisets $\leq$ is defined in a similar way to $\subseteq$ for sets. That is, $N \leq M$ if, for all $x \in N$, the number of occurrences of x in N is no more than in M.

Given this definition of order on multisets, super and sub-sequents can be defined. A sequent $\Gamma_1 \vdash \Delta_1$ is a super-sequent of $\Gamma_2 \vdash \Delta_2$ if $\Gamma_2 \leq \Gamma_1$ and $\Delta_2 \leq \Delta_1$. Similarly, $\Gamma_2 \vdash \Delta_2$ is a sub-sequent of $\Gamma_1 \vdash \Delta_1$.

The usefulness of the above ordering on multisets and sequents should be evident from the nature of admissibility proofs, especially when applying contraction-admissibility results in the formalised setting of an ITP.

In Isabelle syntax, the empty multiset is represented as `{#}`, a multiset consisting of a single element `A` is `{#A#}`. Sequents are then defined as a pair of multisets, of any type, although henceforth we will only consider multisets of formulae. The syntax for a sequent is `(X |- Y)`, where `X` and `Y` are arbitrary multisets of formulae.The antecedent and succedent of a sequent can be accessed using the functions `antec` and `succ` respectively.

A sequent rule is also defined as a pair `(ps, c)`, where `ps` is a list of premises and `c` is a single conclusion. Once again, the formalisation is generic (the actual definition is of a type for single step inference encoded as `types 'a psc = "'a list * 'a"`), but for this work the premises and conclusion will always be sequents. As an example, consider an axiom for some formula variable `A` given as:

`([], ({#A#} |- {#A#}))` corresponding to $\dfrac{}{A \vdash A}$

This deep embedding of rules is necessary to prove some properties about calculi. The most important of which is proving that one set of rules is a subset of another, which in turns allows for a modular representation of calculi as a collection of various sets of rules. This usefulness of this modularity is made obvious in Chapter 5.

4.1.2 Derivations

As described in Section 3.2, meta-level proofs generally follow an inductive procedure on derivations. The previous proofs have all considered explicit derivation trees, typically represented with Π. However, most transformations do not actually require use of the height of Π in creating some new derivation, they only refer to the endsequents derived by Π. In particular, an induction on height can often be mimicked by an induction on rule applications. Rather than using an actual derivation tree, where the induction hypothesis is applied on a sub-tree of height n, it is normally simpler in a formal proof to deal only with the last rule, where the induction hypothesis is applied on the premises of that rule.

This leads to a shallow embedding of derivations, as inductively defined sets:

```
consts derrec :: "'a psc set => 'a set => 'a set"
```

where `derrec rls prems` is the set of sequents derivable using the rules `rls` from a set `prems` of premises. Formally this is encoded as:

$c \in \texttt{prems} \Longrightarrow c \in \texttt{derrec rules prems}$

$[\!|\, (ps, c) \in \texttt{rules}\ ;\ ps \subseteq \texttt{derrec rules prems}\ |\!] \Longrightarrow c \in \texttt{derrec rules prems}$

Note that in Isabelle syntax, $[\![X\ ;\ Y\ ;\ Z]\!] \Longrightarrow P$, states that given X, Y, Z it follows that P holds. The brackets are omitted if there is only one assumption.

While shallow derivations are used whenever possible, there are cases where reasoning about the height of derivation is unavoidable, for these cases a deep embedding of derivation trees is defined. Such trees are also generic, but will be considered to consist of a root sequent, and a number of subtrees, or an unfinished tree. A derivation tree is formalised to be `valid`[1] if it has no unfinished leafs, and where each node is instance of a rule.

```
datatype 'a dertree = Der 'a ('a dertree list)
    | Unf 'a     (* unfinished and unproved leaf *)
```

Lemmas relating the two styles of derivations are given, so that proofs may alternate between representations depending on whichever style is necessary or easiest for the proof property. One such lemma is Lemma 4.1.1:

Lemma 4.1.1 (`derrec_valid`)**.** *A sequent* `seq` *is derivable, shallowly, from an empty set of premises using* `rls` *iff there exists some derivation tree* `dt` *which is valid with respect to* `rls` *and has a conclusion* `seq`*:*

```
(?seq : derrec ?rls {}) = (EX dt. valid ?rls dt & conclDT dt = ?seq)
```

In Isabelle syntax, the question marks `?` indicate arbitrary terms which can be instantiated, the colon `:` corresponds to $\in$, and `EX` and `&` correspond to the operators $\exists$ and $\wedge$ in Isabelle's object level logic.

The actual ways in which shallow and deep derivations are used for proofs are explained in Section 4.2.

4.1.3 Variations of Rules

Contexts. Sequent rules which incorporate arbitrary Γ or Δ, within their antecedents or succedents respectively, represent an infinite number of rules which may be applied in practice. For example, the following rules are all instances of the L$\wedge$ rule of G3cp:

$$\mathrm{L}\wedge\ \frac{\varphi, \psi \vdash \varphi}{\varphi \wedge \psi \vdash \varphi} \qquad \mathrm{L}\wedge\ \frac{\varphi \to \psi, \varphi, \psi \vdash \psi}{\varphi \to \psi, \varphi \wedge \psi \vdash \psi} \qquad \mathrm{L}\wedge\ \frac{\ldots \varphi, \psi \vdash \ldots}{\ldots \varphi \wedge \psi \vdash \ldots}$$

In the first rule instance, Γ is empty while Δ is the single formula φ. In the second instance $\Gamma = \{\varphi \to \psi\}, \Delta = \{\psi\}$. Note that in the principal formula $\varphi \wedge \psi$, both φ and

[1] `valid` should not be confused with the definition of validity for sequents!

ψ are arbitrary formula. Importantly, the principal section of the rule can be modelled on its own by:

```
([{#A#} + {#B#} |- {#}], ({#A & B#} |- {#})
```

This representation is essentially the third instance above. The basis for encodings of rules in Isabelle follow this style, only showing the principal formulae of a rule and ignoring context.

Each sequent calculus rule which allows some Γ or Δ is then formalised as an inductively defined set, by extending the core part of each rule with arbitrary contexts. In Isabelle this addition of a particular context is handled by a predicate `extend`, taking two arguments as follows: `extend seq ctx = seq + ctx`, where `seq` can be thought of as the original sequent, and `ctx` is the context to be added. A function `pscmap` can then be used to map some predicate to all premises and the conclusion of a rule. For example, the schema for the L$\wedge$ rule can be given in Isabelle syntax as the extension of the core rule by *any* context `?flr`:

```
pscmap (extend ?flr) ([{#A#} + {#B#} |- {#}], ({#A && B#} |- {#})}
```

If `?flr` is represented by $\Gamma \vdash \Delta$ then the above corresponds exactly to the standard presentation of the L$\wedge$ rule:

$$\text{L}\wedge \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta}$$

All rules of G3cp allow any context – every rule includes both Γ on the left as well as Δ on the right in **both** the premises and the conclusion. Hence `pscmap` is sufficient for the calculus. Unfortunately, this simplicity is not generally a feature of calculi. For example, the multiple succedent variant of G3i (which we shall call LM, following the example of [Harland et al. 2000]) for Intuitionistic logic drops Γ in its premise. Furthermore, the calculus G3-LC for Linear Logic is once again different. The R$\rightarrow$ rule keeps implications in the succedent of its premise, but does not include any formulae aside from those from the principal formula.

$$\text{R}\rightarrow\text{(LM)} \frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi, \Delta} \quad \text{R}\rightarrow\text{(G3-LC)} \frac{\Gamma, \varphi_i \vdash \psi_i, \Delta_{-i}}{\Gamma \vdash \varphi \rightarrow \psi, \Delta}$$

In the G3-LC rule, Δ_{-i} is the multiset of all implications only in Δ with the exception of the ith implication $\varphi_i \rightarrow \psi_i$.

Mapping the *same* context to premises and conclusions, or some *arbitrary* context to the conclusion only are the only cases implemented in the original Isabelle code base. Unfortunately, there is no obvious way to create a general function to handle every possibility. When formalising calculi where rules may require specific contexts, new functions must be created on an individual case by case basis. Examples of our contribution to the base formalisation include a function which extends contexts as in the R$\rightarrow$ rule of LM, as well as a method to restrict contexts to non-modal formulae. Details for the latter are given in Section 5.2.

Rules with Multiple Principal Formulae. The rules of G3cp only contain one principal formula. However, in the case of modal logics, there are situations where rules of a calculus require that multiple formulae be modified. For example, this is the case with the $\vdash \Box$ rule of the sequent calculus LK4 for the logic K4. Here, all formulae within Γ are required in both boxed and non-boxed form in the premise:

$$\vdash \Box \frac{\Gamma, \Box\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}$$

Such rules are handled using a mapping function on multisets – `mset_map`. Where `mset_map f M` applies the function `f` to all elements of the multiset `M`. The rule above could be encoded as (where `Box f == FC "Box" [f]`):

```
([(gamma + mset_map Box gamma |- {#A#})], (mset_map gamma |- {#Box A#}))
```

Rules with an Indeterminate Number of Premises. Another issue with more complex calculi are the use of rules where the number of premises depend on the formulae within the conclusion. An example is the S4.3 rule given in [Goré 1994], which introduces boxes in the succedent:

$$\text{S4.3}\Box \frac{S_1 \qquad S_2 \qquad \cdots \qquad S_k}{\Box\Gamma \vdash \Box A_1, \cdots, \Box A_k}$$

where for $1 \leq i \leq k$ we have $\quad \vec{\Phi} = \{A_1, \cdots, A_k\}$
$\vec{\Phi}_{-i} = \vec{\Phi} \setminus \{A_i\}$
$S_i = \Box\Gamma \vdash A_i, \Box\vec{\Phi}_{-i}.$

Such rules are not present in G3cp, and are also not considered in the base formalisation. Like dealing with multiple principal formula, this requires a mapping function from some formula within the conclusion which then generates new premises as required. A more detailed instance of such a formalisation is described in Section 5.2.

4.2 Reasoning and Meta-Level Proofs

In this section we discuss the details involved in proving meta-level theorems of various calculi. These take the form of induction principles on both the shallow and deep embeddings of derivations as described earlier.

The main focus of this text is proving structural rule admissibility for the sake of efficient calculi describing S4 and S4.3. The calculi for these systems are themselves based upon G3cp. As noted in Chapter 3, the axiom rule of G3cp will be allowed to apply to formula variables, rather than only atomic propositions. This in turn disallows proofs of height-preserving invertibility or contraction-admissibility as are typical within the literature.

As a consequence, for the proofs of this section, no considerations are made for height-preservation in the statement of admissibility theorems. Indeed, none of the admissibility results in the remainder of this work are explicitly height preserving, although such results are still possible for weakening-admissibility. This has the added

benefit of simplifying the formalisation, by allowing a shallow embedding of derivations to suffice for all proofs except when dealing with Cut.

Notation: in this section the symbol $\in$ will be used to refer to list membership, as well as set membership. Instance of the former are always instances of some premise p belonging in a list of premises ps from a rule instance (ps, c).

4.2.1 Shallow Induction Lemmas

Shallow embeddings will be used for the majority of the formalised proofs in this work. The inductive principles of this section thus form the core foundations of all admissibility proofs. As the shallow embedding of derivations uses inductively defined sets, Isabelle provides an induction principle for proving properties for such derivations. A slightly simplified version is used as a basis for proving results about weakening-admissibility and invertibility.

Lemma 4.2.1 (`drs.inductr`)**.**
For a derivable sequent $x \in$ `derrec rls prems`, if property P holds for all initial sequents `prems` (such initial assumptions are always empty for this work), and also for all $(ps, c) \in$ `rls` it can be shown that $\forall p \in ps.\ P\ p$ implies $P\ c$, then $P\ x$ also holds.

Lemma 4.2.1 essentially replaces the pen and paper induction on height, with an induction on rule applications – if every rule preserves a property P from premises to conclusion, then so does the whole derivation. The downside of this principle is that there is never an explicit derivation tree, only the exact premises of a rule application are accessible, so transforming sub-trees in some height-preserving way is impossible.

For proofs of contraction-admissibility, without height-preservation, a stronger induction principle is required (as described in Section 3.2). A double induction, on premises (instead of explicit derivation height) and formula size (as measured by any well-founded subformula relation) is used. The implementation of this first encodes an inductive step condition:

Definition 4.2.2 (`gen_step`)**.**
For a formula A, a property P, a subformula relation `sub`, a set of rules `rls`, a set of `rls`-derivable sequents `derivs` (e.g. `derrec rls prems`), and a particular rule `r` $= (ps, c)$,

`gen_step` $P\ A$ `sub rls derivs r` means:

if *forall A' such that $(A', A) \in$ `sub`*
and all `rls`-derivable sequents D the property $P\ A'\ D$ holds
and for every premise $p \in ps$ both $p \in$ `derivs` and $P\ A\ p$ holds

then *$P\ A\ c$ holds.*

In English, this step condition states that if a property P is true for formulae of lower rank A', and for the (derivable) premises of a particular rule then the property holds for the conclusion of that rule. The main theorem, named `gen_step_lem` and given as Theorem 4.2.3, states that if this step case can be proved for all possible rule instances then P holds for all cases.

Theorem 4.2.3 (gen_step_lem)**.**

if *item A is in the well-founded part of the subformula relation* `sub`
and sequent `S` *is* `rls`*-derivable*
and for all formulae A', all possible rules `r` *in* `rls`*, and all* `rls`*-derivable sequents* `derrec rls {}`*, the induction step condition* `gen_step` P A' `sub (derrec rls {}) r` *holds*

then `P A S` *holds.*

Note that the principles above apply only for single derivations, as only one rule `r` is considered. For cut-admissibility proofs, a similar construction to that of Theorem 4.2.3 is employed involving two rules. In the base formalisation these are implemented as `gen_step2ssr` and `gen_step2ssr_lem`:

Definition 4.2.4 (gen_step2ssr)**.**
For a formula A, a property P, a subformula relation `sub`*, a set of rules* `rls`*, and rule instances* `rl` $= (ps_l, c_l)$`, rr` $= (ps_r, c_r)$*,*
`gen_step2ssr` P A `sub rls (rl, rr)` *means:*

if *forall A' such that $(A', A) \in$* `sub`
and all `rls`*-derivable sequents D_l and D_r the property P A' (D_l, D_r) holds*
and for every premise $p_l \in ps_l$ the property P A (p_l, c_r) holds and for every premise $p_r \in ps_r$ the property P A (c_l, p_r) holds

then *P A (c_l, c_r) holds.*

Here, the property P is a property of a formula A, and two rule instances. As a result the assumptions of the step condition now require that P holds for the conclusion of the left rule and the premises of right, as well as for the conclusion of the right rule and the premises of the left. The main theorem, `gen_step2ssr_lem`, once again requires the step condition to hold for all possible rule instances. Further details about this definition, in the context of cut-admissibility, are given shortly in Section 4.2.4.

Theorem 4.2.5 (gen_step2ssr_lem)**.**

if *A is in the well-founded part of the subformula relation* `sub`
and sequents S_l and S_r are `rls`*-derivable*
and for all formulae A', and all rules `rl` *and* `rr`*, the induction step condition*
`gen_step2ssr` P A' `sub rls (rl, rr)` *holds*

then `P A (`S_l`,` S_r`)` *holds.*

4.2.2 Weakening

In Isabelle, the main statement for the admissibility of Weakening is `wk_adm`. We extend the base formalisation by also introducing statements and lemmas for weakening of single formulae. These significantly simplify the proof of weakening-admissibility for S4.3, as proven in Chapter 5.2.

wk_adm rls
: This states that Weakening is admissible for `rls`. In particular, for any `rls`-derivable sequent S, the addition of any other sequent As to this is also `rls`-derivable. Formally, given $S \in$ `derrec rls {}` then $S + As \in$ `derrec rls {}`. This is part of the base formalisation.

wk_adm_single_antec rls
: For any `rls`-derivable sequent S, and any single formulae A,
if $S \in$ `derrec rls {}` then $S+$`({#A#} |- {#})` $\in$ `derrec rls {}`.

wk_adm_single_succ rls
: For any `rls`-derivable sequent S, and any single formulae A,
if $S \in$ `derrec rls {}` then $S+$`({#} |- {#A#})` $\in$ `derrec rls {}`.

Lemma 4.2.6 (`wk_adm_sides`)**.** *If* `wk_adm_single_antec` *and* `wk_adm_single_succ` *both hold for a set of rules* `rls`*, then so does* `wk_adm`*.*

Proof. By multiset induction, repeatedly applying the results for single formulae. ⊣

In the base formalisation, a general lemma about weakening-admissibility suffices for a fairly broad range of calculi. If the conclusion of rules can be extended with an arbitrary context, then proving weakening-admissibility is very straightforward using an induction on height (premises in the formalisation). Weakening in formulae is as simple as instantiating the context of the rule as it was originally, but with the new formula also included. Note that this may require introducing new formulae in the premises as well, using the inductive hypothesis.

Lemma 4.2.7 (`wk_adm_ext_concl`)**.** *If all rules in a calculus can be extended by arbitrary contexts in the conclusion, then Weakening is admissible in that calculus.*

This lemma is essentially a sufficient condition for weakening-admissibility, in the same style as the conditions for cut-admissibility given by Rasga [2005]. For calculi where some rule(s) restrict contexts in their conclusion, the lemma `wk_adm_ext_concl` no longer applies, and so a proof of weakening-admissibility requires the standard inductive style proof involving a case analysis of the last rule applied.

4.2.3 Invertibility and Contraction

Invertibility and admissibility of Contraction are stated using `inv_rl` and `ctr_adm` respectively:

inv_rl rls (ps, c)
: Given a rule instance `(ps, c)`, within a particular calculus `rls`, `inv_rl` holds if whenever `c` is `rls`-derivable, then all the premises `ps` are also `rls`-derivable.

ctr_adm rls A
: Contraction-admissibility is given as a property for a set of rules `rls`, and a single contraction-formula `A`: for all sequents `As`, if the only formula in `As` is

equal to `A` then for all `rls`-derivable sequents S where `As + As` is a sub-sequent of S, it holds that $S-$`As` is also `rls`-derivable.

The rather strange definition for `ctr_adm` is used so that proving contraction-admissibility is possible using `gen_step_lem`, while also allowing the contraction-formula to be present as part of either the antecedent or the succedent.

It should also be noted that the base formalisation only states admissibility for a single contraction-formula. However, a simple induction on multisets is sufficient to transform this result into an admissibility result for multiple formulae.

Like `wk_adm_ext_concl`, the proof of inversion lemmas only uses the simpler induction principle `drs.inductr` from Lemma 4.2.1. Proving `ctr_adm` uses the induction principle `gen_step_lem` as in Lemma 4.2.3, which allows a double induction over both height and rank. Like the pen and paper proofs, the Isabelle proofs of `ctr_adm` within this text make heavy use of inversion results.

It is also interesting to note that when using contraction-admissibility within a cut-admissibility proof, instead of picking a number of formula to contract on, it is simpler in Isabelle to show that the original sequent is a sub-sequent of the result after duplicating all formulae within the desired endsequent. As an example, consider a derivation of a sequent $\Gamma_C, \Gamma_C, \Gamma \vdash \Delta$ where we wish to use contraction-admissibility to derive $\Gamma_C, \Gamma \vdash \Delta$. In Isabelle, this is done by showing that the given sequent $\Gamma_C, \Gamma_C, \Gamma \vdash \Delta$ is a sub-sequent of $\Gamma_C, \Gamma_C, \Gamma, \Gamma \vdash \Delta, \Delta$.

4.2.4 Cut

In a similar manner to contraction-admissibility, cut-admissibility is stated in a somewhat roundabout manner so that it fits the format used in its corresponding inductive principles. For a cut-formula A, cut-admissibility is expressed as:

`(Xl |- mins A Yl, mins A Xr |- Yr) :  cas rls A)`

The property name is `cas`, and `rls` once again represents a calculus (set of rules). `mins` is multiset insertion, while `X` and `Y` for both the left and right represent arbitrary multisets. The pair of sequents in the cut-admissibility statement then simply correspond to two potential premises for an application of the Cut rule. The definition of `cas` states that if both these left and right sequents are `rls`-derivable, then so is the sequent, `(Xl + Xr |- Yl + Yr)`, which would result from applying Cut on A.

The formalised proofs of cut-admissibility in this work do not exactly follow the traditional style of cut-elimination presented in Section 3.2. Instead cut-admissibility is proved. Recall that such proofs are given cut-free derivations of cut-sequents, and transform these to give a derivation of the endsequent corresponding to an application of Cut. Unlike cut-elimination, in cut-admissibility proofs, a strategy for eliminating cuts from a derivation (i.e. starting from the lowest level cut of maximum rank) is not necessary. The original derivations are given to be cut-free.

The actual difference between the transformations used for cut-elimination and cut-admissibility is rather subtle. Instead of using a calculus with an explicit Cut

rule, only the statement of cut-admissibility is used in the Isabelle transformations. In other words, instead of transforming an actual cut instance into other instances of cut with lower rank or level, the proof begins with two *potential* cut-sequents and transforms these into instances of other *potential* cut-sequents of lower rank or level (assuming that cut-admissibility inductively holds for these) to produce a derivation for the desired endsequent.

When proving cut-admissibility in this manner, `gen_step2ssr_lem` is the only induction principle required for the rules of the G3cp calculus. The premises of the last-rule based induction principle are enough to create new transformations for every case. However, the deep embedding of explicit derivations trees is necessary for more complex calculi – where the transformations create derivations of sequents which are no longer premises of the rules applied above cut. A cut-admissibility proof for the G3cp calculus exists in the base formalisation.

The base formalisation includes an inductive principle on heights of derivation trees. However, this principle turns out to be inadequate for some transformations. To explain, we first illustrate how the existing principles work.

$$\mathcal{R}_l \dfrac{\begin{array}{c}\Pi_l \\ \mathcal{P}_{l1} \dots \mathcal{P}_{ln}\end{array}}{\mathcal{C}_l} \quad \mathcal{R}_r \dfrac{\begin{array}{c}\Pi_r \\ \mathcal{P}_{r1} \dots \mathcal{P}_{rm}\end{array}}{\mathcal{C}_r} \; \text{Cut?}$$

The above represents an arbitrary instance of two derivations, for which we wish to prove a property using induction. The generic statement of any inductive principle for this situation is roughly: a property P (admissibility of Cut in this work) applies to $\mathcal{C}_l$ with $\mathcal{C}_r$ if it can be shown to follow either directly or using the property on derivations which are somehow smaller than the ones leading to $\mathcal{C}_l$ and $\mathcal{C}_r$.

The shallow principle, defined earlier as `gen_step2ssr_lem`, allows the induction hypothesis to apply P to an original conclusion with the premises of the other derivation. If P is `cas` (cut-admissibility), this states that cut-admissibility for $\mathcal{C}_l$ with any of $\mathcal{P}_{r1} \dots \mathcal{P}_{rm}$ is permitted, and similarly for $\mathcal{C}_r$ with any of $\mathcal{P}_{l1} \dots \mathcal{P}_{ln}$. The issue lies with the requirement on the exact sequents present in the original derivations.

The base formalisation includes another lemma, `height_step2_tr_lem`, which is more flexible than `gen_step2ssr_lem`. Using explicit derivation trees, this allows the induction hypothesis to apply to one of the original conclusions with any sequent for which a derivation tree of lower height than the original second derivation. For `cas`, this states that cut-admissibility can be used for $\mathcal{C}_l$ with any derivation of lower height than $\mathcal{C}_r$, and vice versa.

For the particular transformations used later in Chapter 5, an even more general version is required. In particular, the restriction to using an exact copy of either $\mathcal{C}_l$ or $\mathcal{C}_r$ is lifted. The Isabelle formalisation is thus extended by a new principle where the inductive hypothesis no longer involves an original conclusion $\mathcal{C}_l$ or $\mathcal{C}_r$.

The assumption now is that the property holds for two new derivations, where some well-founded measure on the new derivations is strictly less than the same measure on the derivations leading to $\mathcal{C}_l$ and $\mathcal{C}_r$. For Cut, this measure will be derivation

height. Essentially, cut-admissibility of new derivations can be assumed, as long as their combined height is less than the level of the original cut-sequents $\mathcal{C}_l$ and $\mathcal{C}_r$.

The implementation of this new induction principle is once again based on a step condition[2]:

Definition 4.2.8 (sum_step2_tr)**.**
For a formula A, a property P, a subformula relation `sub`*, measures on derivation trees* `gsl` *and* `gsr`*, and explicit derivation trees* `dta` *and* `dtb`*,*

`sum_step2_tr` P A `sub gsl gsr (dta, dtb)` *means:*

if *forall A' such that $(A', A) \in$* `sub`
the property P A' (a, b) holds for all derivation trees `a` *and* `b`
and for all derivation trees `dta'` *and* `dtb'` *where*
`gsl dta' + gsr dtb' < gsl dta + gsr dtb`
the property P $A(dta', dtb')$ holds

then *P A (dta, dtb) holds.*

Instantiating the measures in sum_step2_tr with measures of derivation height results in the final principle used to prove cut-admissibility:

Theorem 4.2.9 (sumh_step2_tr_lem)**.**
Where `heightDT dt` *gives the height of an explicit derivation tree* `dt`*.*

if `A` *is in the well-founded part of the subformula relation* `sub`
and for all formulae `A'`*, and derivation trees* `dta` *and* `dtb`
the induction step condition `sum_step2_tr P A' sub heightDT heightDT (dta, dtb)` *holds*

then `P A (dta, dtb)` *holds*

It should be obvious that a proof using gen_step2ssr_lem, which has stronger restrictions, implies a proof using sumh_step2_tr_lem. In Isabelle, this allows combining both inductive principles during cut-admissibility proofs (Lemma 4.2.10). In the Isabelle cut-admissibility proofs for S4 and S4.3 of Chapter 5, most cases are proven using gen_step2ssr_lem. The proofs resort to the deep embedding of derivations only when necessary.

Lemma 4.2.10 (gs2_car_sumhs_tr)**.**
For cut-admissibility, if the step condition `gen_step2ssr` *holds, then the step condition* `sum_step2_tr` *holds.*
(For the same cut-formula, subformula relation, where both measures for `sum_step2_tr` *are* `heightDT`*, and the bottommost rules of the derivations in* `sum_step2_tr` *are the same as those in* `gen_step2ssr`*).*

Furthermore, like Lemma 4.2.7 for Weakening, a number of generic lemmas for the axiom and parametric cases are also implemented. All of these general cases only require the shallow embedding of derivations.

[2]The validity of this step condition was proved by Jeremy Dawson in the formalisation

Lemma 4.2.11 (`gs2_axls, gs2_axrs`)**.**
If weakening-admissibility for `rls` *holds, and the last rule $\mathcal{R}$ applied above the left (right) cut-sequent is an axiom, then the step condition* `gen_step2ssr` *holds for $\mathcal{R}$ on the left (right).*

Proof. The reasoning is the same as the axiom case in Lemma 3.2.5. The difference for the formalisation is that the axiom no longer requires atomic formulae, and any non-axiom derivations are known to be cut-free. ⊣

Lemma 4.2.12 (`lcg_gen_steps, rcg_gen_steps`)**.**
If weakening-admissibility for `rls` *holds, and the last rule $\mathcal{R}$ applied above the left (right) cut-sequent allows any contexts (in both premises and conclusion), and the cut-formula is not principle in $\mathcal{R}$, then the step condition* `gen_step2ssr` *holds for $\mathcal{R}$ on the left (right).*

Proof. This is the parametric case in Lemma 3.2.5. ⊣

The above two lemmas are a part of the base formalisation. For the calculi discussed in Chapter 5, we are required to implement a different variant of the second lemma for rules which do not allow contexts in the premise, but have their conclusion extended by some context.

Lemma 4.2.13 (`lcg_gen_steps_extconc, rcg_gen_steps_extconc`)**.**
If weakening-admissibility for `rls` *holds, and the last rule $\mathcal{R}$ applied above the left (right) cut-sequent includes some context in the conclusion only, and the cut-formula is not principle in $\mathcal{R}$, then the induction step condition* `gen_step2ssr` *holds for $\mathcal{R}$ on the left (right).*

Proof. We show the explicit transformation for $\mathcal{R}$ on the left, the case is symmetric if $\mathcal{R}$ is used to derive the right cut-sequent. As the cut-formula A is known to be parametric, the original derivation can be expressed in terms of its principle formula (denoted using subscript P) and its context:

$$\text{Cut?}\ \dfrac{\mathcal{R}\ \dfrac{\begin{matrix}\Pi_l\\ \mathcal{P}_{l1}\ldots\mathcal{P}_{ln}\end{matrix}}{\Gamma,\Gamma_P\vdash,\Delta_P,\Delta,A} \qquad \begin{matrix}\Pi_r\\ A,\Gamma'\vdash\Delta'\end{matrix}}{\Gamma,\Gamma_P,\Gamma'\vdash\Delta_P,\Delta,\Delta'}$$

The premises $\mathcal{P}_{l1}\ldots\mathcal{P}_{ln}$ contain no context, so a new instance of the rule $\mathcal{R}$ can be used from these to derive only the principal formulae within the left cut-sequent. We now have a derivation of $\Gamma_P\vdash\Delta_P$, from which using weakening-admissibility to introduce $\Gamma,\Gamma',\Delta,\Delta'$ gives the required endsequent. ⊣

Chapter 5

Formalisation Instances

In this chapter we provide formalisations in Isabelle, using the framework described previously, of sequent calculi for S4 and S4.3. Notably these calculi are all free of the structural rules Weakening, Contraction, and Cut. The admissibility of these structural rules are proved both with pen and paper proofs as well as in Isabelle/HOL. To our knowledge these give the first formalised admissibility proofs for each particular calculus. The pen and paper proof for S4.3 is also new.

Note that the pen and paper proofs for structural rule admissibility given in this chapter are my own. However, my supervisor Jeremy Dawson helped with a number of subgoals in the formalisation. Especially in instances which required convincing Isabelle that various statements (typically sequents) are in fact identical. An example, taken from the Isabelle proof of Contraction admissibility for S4, is the identity:

$$(\Box(\Gamma - B) \vdash \Box A) + (\Gamma' \vdash \Delta') = ((\Box\Gamma, \Gamma') - \Box B \vdash \Box A, \Delta')$$

5.1 S4: The Modal Logic of Reflexive and Transitive Frames

S4 is a Classical Modal Logic with uses in verification and answer set programming [Osorio and Pérez 2004]. The Kripke semantics for the logic follow from the description in Section 2.1.1, where the relation R is reflexive and transitive.

There exist both pen and paper [Ohnishi and Matsumoto 1957; Troelstra and Schwichtenberg 2000] and a formalised proof [Dawson 2014b] of mix-elimination for sequent calculi describing S4 which include explicit Weakening and Contraction rules. As stated previously, the inclusion of structural rules are detrimental for automated reasoning and so provides a practical motivation for proving admissibility results for calculus free of such rules.

Troelstra and Schwichtenberg also state cut-elimination for a sequent calculus G3s that no longer uses explicit structural rules. Unfortunately, the "proof" of their theorem only discusses one actual transformation, and in particular overlooks one non-trivial case – when Cut is applied on a formula $\Box A$, with both premises being an instance of the G3s R$\Box$ rule (shown below). In this case, the deduction cannot be transformed by simply permuting the Cut, or introducing a new Cut of smaller rank, on the sequents in the original deduction. Greater detail is given in Section 5.1.3

$$\text{R}\Box\ \frac{\Box\Gamma \vdash A, \Diamond\Delta}{\Gamma', \Box\Gamma \vdash \Box A, \Diamond\Delta, \Delta'}$$

Goubault-Larrecq [1996] acknowledges the problem posed by absorbing Weakening into the R$\Box$ rule. However, the solutions discussed are given in the context of typed λ-calculi for a minimal version of S4, interpreted as sequent calculus through a version of the Curry-Howard correspondence. Based on a proposal from Bierman and de Paiva [1996], Goubault-Larrecq replaces the R$\Box$ rule by a different rule with multiple premises (for subformula within the principal formula), along with both re-write and garbage collection rules for the λ terms involved. While this solution could possibly be extended to sequent calculi, the creation of new premises and hence branching is detrimental to backward proof search. The solution presented in this section also has the advantage of being significantly simpler.

Negri [2005] proves various admissibility theorems for S4, but the calculus involved is labelled. These labels include elements of the Kripke semantics within the calculus, and so the resulting theorems are thus not entirely syntactical proofs. Furthermore, there are rules in the calculus which deal only with reachability between worlds. While perhaps not as inefficient as the standard structural rules, these rules nevertheless do not operate on logical connectives. In particular to S4, from the perspective of automated reasoning, applying all possible instances of the transitivity rule (shown below) or checking whether the transitivity rule has been saturated can be a very time-consuming process.

$$\text{Transitivity}\ \frac{xRz, xRy, yRz, \Gamma \vdash \Delta}{xRy, yRz, \Gamma \vdash \Delta}$$

R is the accessibility relation. x, y, z are worlds.

5.1.1 Calculus

In this Chapter we provide the first, to our knowledge, formally verified proofs of Weakening and Contraction admissibility for a sequent calculus for S4. This also constitutes the first formalisation of cut-admissibility as opposed to mix-admissibility for S4. The rules of the calculus used are given in Figure 5.1. The calculus is essentially G3cp with the inclusion of modal rules.

The most notable features of the calculus of Figure 5.1 are: the absence of structural rules, the axiom rule allowing any formula, and the use of only two modal rules. The calculus relies on a pre-processing step to re-write certain formulae, as there are no rules for formulae of the form $\bot$ and $\Diamond\varphi$, as these are equivalent to $p_0 \wedge \neg p_0$ and $\neg\Box\neg\varphi$ respectively in Classical Modal Logics. A pre-processing stage for re-writing formulae is commonly used in ATPs, details are covered in Chapter 6).

The Refl and S4$\Box$ rules are based on the traditional rules L$\Box$ and R$\Box$ from the calculus known as G1s, the originals are shown in Figure 5.2. The new modal rules absorb the effects of Weakening and Contraction like the process given in Chapter 3.

Like G1cp, the traditional modal rules require applications of Contraction and Weakening. An example where a structural rule is necessary is shown in Figure 5.3.

Axioms

$$\text{id}\ \frac{}{\Gamma, \varphi \vdash \varphi, \Delta}$$

Classical rules

$$\text{L}\wedge\ \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta} \qquad \text{R}\wedge\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}$$

$$\text{L}\vee\ \frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}$$

$$\text{L}\to\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \to \psi \vdash \Delta} \qquad \text{R}\to\ \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}$$

$$\text{L}\neg\ \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \qquad \text{R}\neg\ \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}$$

Modal rules

$$\text{Refl}\ \frac{\Gamma, \varphi, \Box\varphi \vdash \Delta}{\Gamma, \Box\varphi \vdash \Delta} \qquad \text{S4}\Box\ \frac{\Gamma, \Box\Gamma \vdash \varphi}{\Sigma, \Box\Gamma \vdash \Box\varphi, \Delta}$$

Figure 5.1: Sequent calculus for S4.

$$\text{L}\Box\ \frac{\Gamma, \varphi \vdash \Delta}{\Gamma, \Box\varphi \vdash \Delta} \qquad \text{S4}\Box\ \frac{\Box\Gamma \vdash \varphi}{\Box\Gamma \vdash \Box\varphi}$$

Figure 5.2: Traditional modal rules for S4.

The new modal rules shown in Figure 5.1 absorb these structural rules. Note, however, that the "un-boxed" copy of Γ in the premise of the S4$\Box$ rule is not strictly necessary, as it can be obtained through applications of the Refl rule. However, for automated reasoning, un-boxing everything after an instance of S4$\Box$ is generally easier.

$$\cfrac{\cfrac{\cfrac{\cfrac{\text{id}\ \dfrac{}{\Box A \vdash \Box A}}{\text{S4}\Box\ \dfrac{\Box A \vdash \Box A}{\Box A \vdash \Box\Box A}}}{\text{RW}\ \dfrac{\Box A \vdash \Box\Box A}{\Box A \vdash \Box\Box A, B}}}{\text{R}\vee\ \dfrac{\Box A \vdash \Box\Box A, B}{\Box A \vdash \Box\Box A \vee B}}$$

Figure 5.3: An example derivation using the traditional modal rules.

The Isabelle encodings of rules corresponding to those in Figure 5.1 are shown in Figure 5.4. The encoding itself is defined in a modular manner. The foundations of the classical rules can be considered the principal parts of the logical rules of the G1cp/LK calculus (`lksne`), with a distinction made for rules acting on the antecedent and succedent (`lksil` and `lksir` respectively). These rules are then extended to correspond to G3cp (`lksss`) and then combined with the modal rules for S4.

The modularity is extremely useful when doing formalised proofs which do a case analysis of applied rules. By grouping rules together appropriately and combining these groups to form a calculus, many proofs can be significantly shortened.

As an example, consider an application of Cut where the cut-formula A is principal on both sides, the last rule applied on one side is modal, and the last rule on the other is classical. Clearly, the only valid classical rule is the id or axiom rule, as principality in a modal rule requires the cut-formula to be a $\Box$ formula. However, to formally prove this, a naïve approach would have to consider all cases of the G3cp rules separately. This would result in nine distinct cases (id, and four logical rules for both the antecedent and succedent). In the modular Isabelle formalisation, there is a theorem stating that no rule in `lksil` or `lksir` can act on a boxed formula. Thus only three cases must be examined, (id, `lksil`, `lksir`).

The modularity also allows the use of previous results for subsets of the calculus. The base formalisation includes a proof of cut-admissibility for the `lksss` rules, and rather than proving this again for the cut-admissibility of the `gs4_rls`, the previous result can be substituted in when required.

5.1.2 Weakening, Invertibility and Contraction

Weakening admissibility follows simply from the fact that all rules allow arbitrary contexts in their conclusions.

Theorem 5.1.1 (`gs4_wk_adm`)**.** *Weakening is admissible for the calculus* `gs4_rls`*.*

Proof. In Isabelle, the proof requires showing that all rules in `gs4_rls` allow any contexts in their conclusions. This then allows for an application of Lemma 4.2.7. The entirety of the proof runs as follows:

```
val _ = qed_goal "gs4_ext_concl" gs4.thy "ext_concl gs4_rls"
  (fn _ => [ (simp_tac (esimps [gs4_def]) 1),
    (REPEAT_ALL_NEW (rtac ext_concl_Un) 1),
    (ALLGOALS (simp_tac (esimps [lksss_extrs]))) ]) ;

bind_thm ("gs4_wk_adm", gs4_ext_concl RS wk_adm_ext_concl)
```

Note that there are tactics and theorems in the proof script that have not been covered by this work. Fully explaining these requires a deeper understanding of the Isabelle formalisation than is necessary from a proof theoretic perspective. It is sufficient to merely think of these tactics as convincing Isabelle that arbitrary Γ/Σ and Δ are indeed present in the rules of the calculus. ⊣

The theorem `gs4_ext_concl` proves that the conclusions of all rules in `gs4_rls` allow any context in their conclusion (`ext_concl`). `bind_thm` then combines this result with the previously discussed theorem `wk_adm_ext_concl`, which states that Weakening is admissible whenever `ext_concl` holds. The theorem named `gs4_wk_adm` then states that `wk_adm gs4_rls`.

The proof of Contraction is more involved. To simplify matters, inversion lemmas are first proved for the `lksss` and `lkrefl` rules within the calculus `gs4_rls`.

Proving invertibility for the Refl rule is straightforward, the result follows immediately from weakening-admissibility as the premise contains all formulae from the

```
consts
  gs4_rls :: "formula sequent psc set"
  lkrefl  :: "formula sequent psc set"
  lkbox   :: "formula sequent psc set"
  lksss   :: "formula sequent psc set"
  lksil   :: "formula sequent psc set"
  lksir   :: "formula sequent psc set"
  lksne   :: "formula sequent psc set"

inductive "lksir"
  intros
    andr : "([{#} |- {#A#}, {#} |- {#B#}], {#} |- {#A && B#}) : lksir"
    orr  : "([{#} |- {#A#} + {#B#}], {#} |- {#A v B#}) : lksir"
    negr : "([{#A#} |- {#}], {#} |- {#--A#}) : lksir"
    impr : "([{#A#} |- {#B#}], {#} |- {#A -> B#}) : lksir"

inductive "lksil"
  intros
    andl : "([{#A#} + {#B#} |- {#}], {#A && B#} |- {#}) : lksil"
    orl  : "([{#A#} |- {#}, {#B#} |- {#}], {#A v B#} |- {#}) : lksil"
    negl : "([{#} |- {#A#}], {#--A#} |- {#}) : lksil"
    impl : "([{#B#} |- {#}, {#} |- {#A#}], {#A -> B#} |- {#}) : lksil"

inductive "lksne"
  intros
    axiom : "([], {#A#} |- {#A#}) : lksne"
    ilI   : "psc : lksil ==> psc : lksne"
    irI   : "psc : lksir ==> psc : lksne"

(* Rules from G3cp *)
inductive "lksss"
  intros
    extI : "psc : lksne ==> pscmap (extend flr) psc : lksss"

(* The Refl rule *)
inductive "lkrefl"
  intros
    I : "([{#A#} + {#Box A#} |- {#}], {#Box A#} |- {#}) : lkrefl"

(* The S4 Box rule *)
inductive "lkbox"
  intros
    I : "([gamma + mset_map Box gamma |- {#A#}],
        mset_map Box gamma |- {#Box A#}) : lkbox"

(* The overall S4 calculus, as an extension of the G3cp calculus *)
inductive "gs4_rls"
  intros
    lksI  : "psc : lksss ==> psc : gs4_rls"
    reflI : "psc : lkrefl ==> pscmap (extend flr) psc : gs4_rls"
    (* Note Box rule allows extra formulae in conclusion only *)
    boxI  : "(p, c) : lkbox ==> (p, extend flr c) : gs4_rls"
```

Figure 5.4: S4 calculus as encoded in Isabelle

conclusion. In particular, the statement of invertibility states: given a derivation of $\Gamma, \Box\varphi \vdash \Delta$, it must be shown that $\Gamma, \varphi, \Box\varphi \vdash \Delta$ is derivable. The derivability of the latter is given by weakening-admissibility, weakening in φ to the antecedent of the original sequent.

For inversion of the G3cp rules, the key observation is that all rules in `gs4_rls` allow arbitrary contexts in their conclusion. As an example, consider invertibility for the R$\vee$ rule:

Lemma 5.1.2 (Invertibility of R$\vee$)**.** *If $\Gamma \vdash \varphi \vee \psi, \Delta$ is derivable, then so is $\Gamma \vdash \varphi, \psi, \Delta$, both using the rules of `gs4_rls`.*

Proof. By an induction on height.

Case 1. Axiom

If $\Gamma \vdash \varphi \vee \psi, \Delta$ is an axiom, and $\varphi \vee \psi$ is principal, then $\Gamma = \Gamma', \varphi \vee \psi$. The derivation for $\Gamma \vdash \varphi, \psi, \Delta$ is then:

$$\mathrm{L}\vee \dfrac{\mathrm{id}\dfrac{}{\Gamma', \varphi \vdash \varphi, \psi, \Delta} \quad \mathrm{id}\dfrac{}{\Gamma', \psi \vdash \varphi, \psi, \Delta}}{\Gamma', \varphi \vee \psi \vdash \varphi, \psi, \Delta}$$

If $\varphi \vee \psi$ is parametric in the axiom, then $\Gamma \vdash \Delta$ is an axiom, and so is $\Gamma \vdash \varphi, \psi, \Delta$.

Case 2. Principal

If $\Gamma \vdash \varphi \vee \psi, \Delta$ is not an axiom, but $\varphi \vee \psi$ is principal, then R$\vee$ must have been the last rule applied. Invertibility follows immediately from the premises of the R$\vee$ rule.

Case 3. Parametric

If $\Gamma \vdash \varphi \vee \psi, \Delta$ is not an axiom, and $\varphi \vee \psi$ is parametric, then an application of a new instance of that last rule (perhaps using the induction hypothesis) obtains the necessary endsequent. This is because all rules allow arbitrary contexts in their conclusion (and premises when the premises contain context). To illustrate, consider the two cases when the last rule used to originally derive $\Gamma \vdash \varphi \vee \psi, \Delta$ is either the Refl or the S4$\Box$ rule.

- If the last rule was Refl then $\Gamma = \Gamma', \Box A$ and the original derivation is:

$$\mathrm{Refl}\dfrac{\begin{matrix}\Pi\\ \Gamma', A, \Box A \vdash \Delta, \varphi \vee \psi\end{matrix}}{\Gamma', \Box A \vdash \Delta, \varphi \vee \psi}$$

  Applying the inductive hypothesis on the premises gives a derivation of $\Gamma', A, \Box A \vdash \Delta, \varphi, \psi$. Applying Refl to this gives the required $\Gamma', \Box A \vdash \varphi, \psi, \Delta$.

- If the last rule was S4$\Box$ then $\Gamma = \Sigma, \Box\Gamma'$ and $\Delta = \Box A, \Delta'$ and the original derivation looks like:

$$\text{S4}\Box\ \frac{\begin{array}{c}\Pi\\ \Gamma', \Box\Gamma' \vdash A\end{array}}{\Sigma, \Box\Gamma' \vdash \Box A, \Delta', \varphi \vee \psi}$$

To derive $\Gamma \vdash \varphi, \psi, \Delta$, simply apply a new instance of S4$\Box$ to the original premises, this time with φ, ψ as the context instead of $\varphi \vee \psi$:

$$\text{S4}\Box\ \frac{\begin{array}{c}\Pi\\ \Gamma', \Box\Gamma' \vdash A\end{array}}{\Sigma, \Box\Gamma' \vdash \Box A, \Delta', \varphi, \psi}$$

⊣

Theorem 5.1.3 (`inv_rl_gs4_refl` and `inv_rl_gs4_lks`)**.** *The `lkrefl` (Refl) and `lksss` (G3cp) rules are invertible within the calculus `gs4_rls`*

Proof. Proof for Refl given above. Inversion for all rules in `lksss` proceed like the proof for R$\vee$. The formal proofs are given in the Appendix. ⊣

The proof of contraction-admissibility is handled by proving that contraction of a single formulae is admissible. The cases for the G3cp and Refl rules are handled in the standard manner[1] using the invertibility results above. If the rule above the contraction is an instance of the S4$\Box$ rule, there are two possible cases. Either one or both copies of the contraction-formula exist within the context of the S4$\Box$ rule, or both copies are principal.

In the former case, case deleting one copy still leaves an instance of the rule. That is, if the contraction-formula is A, with A in the succedent, then the original rule is (where either $\Box\varphi = A$ or $A \in \Delta$):

$$\text{S4}\Box\ \frac{\Gamma, \Box\Gamma \vdash \varphi}{\Sigma, \Box\Gamma \vdash \Box\varphi, A, \Delta}$$

Applying the S4$\Box$ rule without introducing the second copy of A in the context gives a proof of $\Sigma, \Box\Gamma \vdash \Box\varphi, \Delta$ as required (note A is still in the succedent as $\Box\varphi = A$ or $A \in \Delta$). Similarly, if A is in the context Σ the new S4$\Box$ rule is then:

$$\text{S4}\Box\ \frac{\Gamma, \Box\Gamma \vdash \varphi}{\Sigma - A, \Box\Gamma \vdash \varphi, A, \Delta}$$

The harder case occurs when both instances of the contraction-formula A are principal. Due to the nature of the S4$\Box$ rule this requires A to occur in the antecedent only, as there cannot be two principal formulae in the succedent. As only boxed formulae are principal, A has form $\Box B$. The original rule is thus represented by:

$$\text{S4}\Box\ \frac{B, B, \Box B, \Box B, \Gamma, \Box\Gamma \vdash \varphi}{\Sigma, \Box B, \Box B, \Box\Gamma \vdash \varphi, \Delta}$$

The copies of $\Box B$ and B can be contracted upon, first using the induction hypothesis on height, and then on rank. The S4$\Box$ rule can then be applied to give the required conclusion.

[1] cf. Lemma 3.2.3, Troelstra and Schwichtenberg [2000] and Negri and von Plato [2001]

$$\text{S4}\Box\ \frac{B, \Box B, \Gamma, \Box\Gamma \vdash \varphi}{\Sigma, \Box B, \Box\Gamma \vdash \varphi, \Delta}$$

In the Isabelle proof, this step is unfortunately rather more tedious. A significant number of proof steps in the formalisation are dedicated to manipulating the ordering of formulae to convince the proof assistant that the S4$\Box$ rule can be applied after applying the induction hypotheses, and that the resulting sequent is indeed what is required.

Theorem 5.1.4 (`gs4_ctr_adm`)**.** *Contraction is admissible for the calculus* `gs4_rls`*.*

Proof. Pen and paper proof described above. The formalisation performs the same transformations, using the induction principle `gen_step_lem` of Theorem 4.2.3. See the Appendix for the full proof script. The Isabelle statement of the admissibility of Contraction for `gs4_rls` is: `ctr_adm gs4_rls ?A` ⊣

5.1.3 Cut

As described earlier in this chapter, the modular encoding of `gs4_rls` allows the rules applied above the premises of a cut to be divided into 5 main cases. The premises can be any combination of the id, Refl, S4$\Box$, `lksil`, or `lksir` rules.

Any combination of a modal rule and a rule from either `lksil` or `lksir` leads to a parametric case, as none of the logical rules of G3cp can have a principal boxed formula. Likewise, when Refl is the last rule applied above the left premise the cut is also parametric, as Refl can only be principle for formulae in the antecedent. These parametric cases can be handled as described in Chapter 3 and Chapter 4.

When the rules acting on both cut-sequents are principal, then there are three cases to consider. Either `lksss` is applied on both sides, or S4$\Box$ is on the left with either modal rule on the right. For the first case, the transformations are the same as in cut-elimination for G3cp. The latter two cases are the most difficult, and are the focus of this section. Re-iterating, these cases correspond to when the cut-formula is principal below an application of the S4$\Box$ rule on the left, and also principal in either the Refl or the S4$\Box$ rule on the right. As there are all modal rules, the Cut in question must be on a boxed formula, $\Box A$.

In the former case, the original Cut has form:

$$\text{Cut on } \Box A\ \frac{\text{S4}\Box\ \dfrac{\begin{matrix}\Pi_l\\ \Gamma_L, \Box\Gamma_L \vdash A\end{matrix}}{\Sigma, \Box\Gamma_L \vdash \Delta_L, \Box A} \qquad \text{Refl}\ \dfrac{\begin{matrix}\Pi_r\\ A, \Box A, \Gamma_R \vdash \Delta_R\end{matrix}}{\Box A, \Gamma_R \vdash \Delta_R}}{\Sigma, \Box\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}$$

This is transformed as follows:

$$\dfrac{\dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}\quad \dfrac{\dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}\ \text{S4}\Box}{\Sigma, \Box\Gamma_L \vdash \Delta_L, \Box A}\quad \dfrac{\Pi_r}{A, \Box A, \Gamma_R \vdash \Delta_R}}{A, \Sigma, \Box\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\ \text{Cut on } \Box A}{\Sigma, \Gamma_L, \Box\Gamma_L, \Box\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\ \text{Cut on } A}{\Sigma, \Gamma_L, \Box\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\ \text{Contraction-admissibility}}{\Sigma, \Box\Gamma_L, \Gamma_R \vdash \Delta_L, \Delta_R}\ \text{Refl}$$

Importantly, the new Cut on $\Box A$ has lower level, and the Cut on A is of smaller rank. Thus both can be eliminated by the induction hypotheses.

For the latter case, when S4$\Box$ is principal on both sides, the original Cut has form:

$$\text{Cut on } \Box A\ \dfrac{\text{S4}\Box\ \dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}}{\Sigma_L, \Box\Gamma_L \vdash \Delta_L, \Box A}\quad \text{S4}\Box\ \dfrac{\dfrac{\Pi_r}{A, \Box A, \Gamma_R, \Box\Gamma_R \vdash B}}{\Box A, \Sigma_R, \Box\Gamma_R \vdash \Box B, \Delta_R}}{\Sigma_L, \Sigma_R, \Box\Gamma_L, \Box\Gamma_R \vdash \Box B, \Delta_L, \Delta_R}$$

The normal process of reducing Cut level would apply Cut on the left cut-sequent and the premise of the right cut-sequent, as follows:

$$\text{Cut on } \Box A\ \dfrac{\text{S4}\Box\ \dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}}{\Sigma_L, \Box\Gamma_L \vdash \Delta_L, \Box A}\quad \dfrac{\Pi_r}{A, \Box A, \Gamma_R, \Box\Gamma_R \vdash B}}{A, \Sigma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash B, \Delta_L}$$

Unfortunately, this results in a deduction where we can no longer recover the $\Box B$ present in the conclusion of the original Cut. The nature of the calculus and the S4$\Box$ rule means that new box formulae cannot be introduced in any succedent which contains some context Δ (or where there are additional formula Σ in the antecedent). As stated earlier, this case is omitted in the cut-elimination theorem of [Troelstra and Schwichtenberg 2000].

To overcome this issue, without introducing the complications and the new branching rule as in the solution of [Goubault-Larrecq 1996], we modify the original derivation of the left premise, to produce one of equal height upon which we can still apply the induction hypothesis on level. The new application of the S4$\Box$ rule differs from the original by simply not adding any context in the conclusion. Formally, the Σ and Δ of the generic S4$\Box$ rule in Figure 5.1 are $\emptyset$ in the new S4$\Box$ instance below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}\quad \dfrac{\dfrac{\dfrac{\Pi_l}{\Gamma_L, \Box\Gamma_L \vdash A}\ \text{S4}\Box\ \text{(new)}}{\Box\Gamma_L \vdash \Box A}\quad \dfrac{\Pi_r}{A, \Box A, \Gamma_R, \Box\Gamma_R \vdash B}}{A, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash B}\ \text{Cut on } \Box A}{\Gamma_L, \Box\Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash B}\ \text{Cut on } A}{\Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash B}\ \text{Contraction-admissibility}}{\Sigma_L, \Sigma_R, \Box\Gamma_L, \Box\Gamma_R \vdash \Box B, \Delta_L, \Delta_R}\ \text{S4}\Box$$

In the formalised proof, this instance is the only case where the inductive principle `sumh_step2_tr_lem` is actually required. As the combined height of the derivations leading to $\Box\Gamma_L \vdash \Box A$ and $A, \Box A, \Gamma_R, \Box\Gamma_R \vdash B$ is lower than the level of the original Cut, the induction hypothesis on level can be applied.

The core of the Isabelle proof deals with the nine possible cases of rules on either side of a Cut – where either side is a rule from `lksss`, or `lkrefl`, or `lkbox`. The outermost proof uses the inductive principle `sumh_step2_tr_lem` as stated above. However, for all cases except when `lkbox` is the last rule applied on both premises, the proof instead converts to the simpler induction principle `gen_step2_tr_lem`, using the lemma `gs2_car_sumhs_tr`: and using Lemma 4.2.11 for the formalisation

A general outline for the proof of the nine cases follows:

- As mentioned earlier, the proof for the instance where a rule from `lksss` is the last applied on both sides above the cut follows from a pre-existing proof of cut-admissibility for `lksss`.
- The four cases with a modal rule on one side, and a `lksss` rule on the other, can be decomposed to either the axiom case, or where the Cut is parametric on at least one side. Both cases can then be proved using the appropriate lemmas described in Section 4.2.4.
- The remaining four cases correspond to when two modal rules appear on both sides of the cut.
  - The cut must be parametric for the two cases where either `lkrefl` is on both sides, or `lkrefl` is on the left and `lkbox` on the right. Due to the nature of the `lkrefl` rule, which cannot be principal in the succedent, both cases allow for an application of Lemma 4.2.12.
  - The parametric instance of the remaining two cases (`lkbox` on left, and either `lkbox` or `lkrefl` on the right) are handled using the same lemma. The principal cases correspond to the detailed transformations covered by the pen and paper proof earlier.

Theorem 5.1.5 (`gs4_cas`)**.** *Cut is admissible in the calculus* `gs4_rls`*.*

Proof. As described above. In Isabelle, `gs4_cas` corresponds to

```
(?Xl |- mins ?A ?Yl, mins ?A ?Xr |- ?Yr) : cas gs4_rls ?A
```

See the Appendix for the full formalised proof. ⊣

5.2 S4.3: The Modal Logic of Reflexive, Transitive and Linear Frames

S4.3 is a linear temporal logic which extends S4, the difference being the inclusion of the axiom $\Box(\Box A \to B) \vee \Box(\Box B \to A)$ which imposes linearity [Goré 1994]. In terms of Kripke frames, the relation R for S4.3 is reflexive, transitive and linear. Whereas the frames of S4 can be viewed as finite, reflexive and transitive trees, the linearity of S4.3 prevents branching and so has sequences of worlds rather than trees. The logic itself can be used for verification of algorithms and systems in which time has importance [Felty and Théry 1997].

There exists a syntactic pen and paper proof for S4.3 in the literature [Shimura 1991], however the calculus involved contains Weakening and Contraction as explicit rules, and mix-elimination is proved rather than cut. There also exist published semantic proofs of closure under Cut for both sequent and hypersequent calculi for S4.3 [Goré 1994; Indrzejczak 2012]. But, there are no syntactic pen and paper proofs of cut-admissibility in the literature, nor any machine checked proofs.

To our knowledge, there are also no published papers for S4.3 explicitly providing a sequent calculus containing only logical rules. The labelled calculi of [Negri 2005] and [Castellini 2006] are perhaps the closest representatives in the literature. As noted in Section 5.1, while these calculi do not use Weakening or Contraction, they explicitly include the semantics of the logic in the calculi, along with corresponding operations on world accessibility rather than logical operators.

Unlike S4, for which a number of ATPs have been developed, the development of automated theorem provers for S4.3 is quite lacking. This can be partly attributed to the lack of development of efficient sequent calculi for the logic. There do exist at least two ITPs allowing and automating reasoning in S4.3, see [Felty and Théry 1997; Paulson et al. 2013], however these do not possess the capabilities required for many practical purposes. The former is implemented in λProlog but uses a calculus which contains both Cut and Weakening as explicit rules. The prover also requires humans to drive proofs, and as a result the performance of the prover itself is rather limited. In fact, the authors state that their system is "still quite far from verifying algorithms". The second example is Isabelle, which includes a theory implementing the modal rules of [Goré 1994]. While allowing automated reasoning, the prover itself does not place a great emphasis on performance and only implements a very basic search procedure.

5.2.1 Calculus

The rules of the sequent calculus for S4.3 are listed in Figure 5.5. The calculus is based on the structural version given in [Goré 1994], but with Weakening absorbed into the modal rules. Note, in the S4.3$\Box$ rule of Figure 5.5, that $\vec{\Phi} = \{\varphi_1, \ldots, \varphi_n\}$ and $\vec{\Phi}_{-i} = \{\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i+1}, \ldots, \varphi_n\}$ for $1 \leq i \leq n$.

From the perspective of backward rule application, the S4.3$\Box$ rule can be thought of as producing a new premise for all boxed formula in its conclusion, each of these

Axioms

$$\text{id}\ \frac{}{\Gamma, \varphi \vdash \varphi, \Delta}$$

Classical rules

$$\text{L}\wedge\ \frac{\Gamma, \varphi, \psi \vdash \Delta}{\Gamma, \varphi \wedge \psi \vdash \Delta} \qquad \text{R}\wedge\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \varphi \wedge \psi, \Delta}$$

$$\text{L}\vee\ \frac{\Gamma, \varphi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \vee \psi \vdash \Delta} \qquad \text{R}\vee\ \frac{\Gamma \vdash \varphi, \psi, \Delta}{\Gamma \vdash \varphi \vee \psi, \Delta}$$

$$\text{L}\to\ \frac{\Gamma \vdash \varphi, \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \varphi \to \psi \vdash \Delta} \qquad \text{R}\to\ \frac{\Gamma, \varphi \vdash \psi, \Delta}{\Gamma \vdash \varphi \to \psi, \Delta}$$

$$\text{L}\neg\ \frac{\Gamma \vdash \varphi, \Delta}{\Gamma, \neg\varphi \vdash \Delta} \qquad \text{R}\neg\ \frac{\Gamma, \varphi \vdash \Delta}{\Gamma \vdash \neg\varphi, \Delta}$$

Modal rules

$$\text{Refl}\ \frac{\Gamma, \varphi, \Box\varphi \vdash \Delta}{\Gamma, \Box\varphi \vdash \Delta}$$

$$\text{S4.3}\Box\ \frac{\Gamma, \Box\Gamma \vdash \varphi_1, \Box\vec{\Phi}_{-1} \ \cdots\ \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i} \ \cdots\ \Gamma, \Box\Gamma \vdash \varphi_n, \Box\vec{\Phi}_{-n}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

Figure 5.5: Sequent calculus for S4.3.

formula being un-boxed separately in its own premise. Thus the general statement of the rule contains an indeterminate number of premises, one is necessary for each $\phi_i \in \vec{\Phi}$. For the sake of simplicity and clarity, at times only one of these premises will be shown as a representative for all n premises. That is, the rule will be represented in the following form:

$$\text{S4.3}\Box\ \frac{\Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

There are two different versions of the S4.3$\Box$ rule. Either the context (Σ and Δ) can contain any formulae, or they cannot include top-level Boxes. In the latter case, the $\Box\Gamma$ and $\Box\vec{\Phi}$ in the conclusion of the S4.3$\Box$ rule must correspond to exactly all the top-level boxed formulae within that sequent. The two versions of the calculus are in fact equivalent, following a proof of the admissibility of Weakening for the latter; however, for efficient backward proof search, the non-boxed version is preferred as it does not require backtracking.

Henceforth, Σ and Δ within the S4.3$\Box$ rule will be restricted from containing the $\Box$ operator at the top-level. In Isabelle, this is implemented by creating a new type of formula, based on the default formula type. HOL's `typedef` allows a concise method of declaring new types as a subset of an existing type:

```
typedef nboxfml =
    "{f::formula. ALL (a::formula). f ~= FC ''Box'' [a]}"
```

This declaration automatically produces two functions to cast between top-level non-boxed formulae and standard formulae:

```
Rep_nboxfml :: nboxfml => formula
Abs_nboxfml :: formula => nboxfml
```

A sequent where no formula have $\Box$ as the top-most connective can then be constructed as a sequent using `nboxfml` multisets, then casted to the type of normal formulae, and vice versa for the other direction. These casting functions for sequents are given in Isabelle as `nboxseq` and `seqnbox` respectively:

```
consts
  nboxseq :: "nboxfml sequent => formula sequent"
  seqnbox :: "formula sequent => nboxfml sequent"

defs
  nboxseq_def : "nboxseq seq == (mset_map Rep_nboxfml (antec seq)
                  |- mset_map Rep_nboxfml (succ seq))"
  seqnbox_def : "seqnbox seq == (mset_map Abs_nboxfml (antec seq)
                  |- mset_map Abs_nboxfml (succ seq))"
```

It should be noted that `Abs_nboxfml` is undefined when its input argument cannot be cast as a `nboxfml`, but otherwise the expected inverses hold:

```
Abs_nboxfml (Rep_nboxfml f) = f
f : nboxfml ==> Rep_nboxfml (Abs_nboxfml f) = f
```

The Isabelle formalisation of the overall calculus is based on the calculus for S4 given in Figure 5.4. The only change is in the S4.3$\Box$ rule, which requires the mapping function `nboxseq` to create a new premise for each individual boxed formula in the succedent. The code for this is given in Figure 5.6.

5.2.2 Weakening

As the S4.3$\Box$ rule does not allow arbitrary contexts, weakening-admissibility must be proved by induction, in this case on both height and rank. In order to simplify the case for the S4.3$\Box$ rule and its multiple premises, weakening-admissibility is proven for the antecedent and succedent separately, and only considering a single formula at a time. That is, given a sequent $\Gamma \vdash \Delta$ which is derivable, it must be shown for a single arbitrary formula A, that both $\Gamma, A \vdash \Delta$ and $\Gamma, \vdash \Delta, A$ are also derivable. The induction itself proceeds on the height of the derivation, as well as a sub-induction on the rank of the formula A being inserted.

In the axiom case, if $\Gamma \vdash \Delta$ is an axiom, then it will remain so with the addition of A on either side.

For the inductive step and for both antecedent and succedent cases, when the last rule used to derive the original $\Gamma \vdash \Delta$ is not the S4.3$\Box$ rule, the derivability of the new sequent follows immediately from the fact that any formulae are allowed in the context of those rules. The induction hypothesis is simply applied on the premise(s), and the original rule applied again.

```
(* Functions to unbox one formula for each premise *)
consts
  ithprem :: "formula multiset => formula list => formula => formula sequent"
  nprems  :: "formula multiset => formula list => formula sequent list"

(* The boxes in the succedent are treated as a list.
   "ms_of_list (remove1 Ai As)" is the multiset consisting of all
   elements in "As", with one copy of "Ai" removed. *)
defs
  ithprem_def : "ithprem Gamma As Ai ==
                 mset_map Box Gamma + Gamma |-
                 {#Ai#} + mset_map Box (ms_of_list (remove1 Ai As))"
  nprems_def : "nprems Gamma As == map (ithprem Gamma As) As"

consts
  gs43_rls :: "formula sequent psc set"
  s43box :: "formula sequent psc set"

(* The S4.3 box rule *)
inductive "s43box"
  intros
    I : "(nprems gamma As, mset_map Box gamma |-
          mset_map Box (ms_of_list As)) : s43box"

(* The overall S4.3 calculus, as an extension of the LK calculus *)
inductive "gs4_rls"
  intros
    lksI  : "psc : lksss ==> psc : gs4_rls"
    reflI : "psc : lkrefl ==> pscmap (extend flr) psc : gs4_rls"
    (* Note Box rule allows extra formulae in conclusion only *)
    boxI  : "(p, c) : lkbox ==> (p, extend (nboxseq flr) c) : gs4_rls"
```

Figure 5.6: S4.3 calculus as encoded in Isabelle

In the case of the S4.3$\Box$ rule, if A is not boxed, then it is allowed to be contained in the context of the rule's conclusion. The derivability of the original premises, followed by an application of a new S4.3$\Box$ rule including A as part of its context, then gives the required sequent. The difficulty arises when A is a boxed formula, say $A = \Box B$. For the sake of clarity, the representation of the original sequent can be split into its boxed and non-boxed components, so the original derivation is:

$$\text{S4.3}\Box\ \frac{\begin{matrix}\Pi\\ \Gamma', \Box\Gamma' \vdash \varphi_i, \vec{\Phi}_{-i}\end{matrix}}{\Sigma, \Box\Gamma' \vdash \Box\vec{\Phi}, \Delta'}$$

When A is to be added to the antecedent, the induction on height can be used to add $A = \Box B$ to each of the original premises. Following this by an application of the sub-induction on formula rank, allows the addition of B, giving the derivability of $B, \Box B, \Gamma', \Box\Gamma' \vdash \varphi_i, \vec{\Phi}_{-i}$. An application of the S4.3$\Box$ rule then completes the case:

$$\text{S4.3}\Box\ \frac{B, \Box B, \Gamma', \Box\Gamma' \vdash \varphi_i, \Box\vec{\Phi}_{-i}}{\Box B, \Sigma, \Box\Gamma' \vdash \Box\vec{\Phi}, \Delta'}$$

The final case to consider is that of adding $A = \Box B$ to the succedent. The goal once again is to use the S4.3$\Box$ rule to give the desired conclusion. From the original premises, the induction hypothesis on height (inserting $\Box B$) gives the derivability of $\Gamma', \Box\Gamma' \vdash \varphi_i, \vec{\Phi}_{-i}, \Box B$. A different application of the S4.3$\Box$ rule, bringing in empty contexts, on the original premises also gives the derivability of $\Box\Gamma' \vdash \Box\vec{\Phi}$. Applying the induction on formula rank then shows that $\Box\Gamma' \vdash B, \Box\vec{\Phi}$ is derivable.

At this point, the derivability of all necessary premises for a new S4.3$\Box$ rule has been proven. These are sequents of the form $\Gamma', \Box\Gamma' \vdash \varphi'_i, \Box\vec{\Phi}'_{-i}$ where $\vec{\Phi}' = \vec{\Phi}, B$. The final rule application is then:

$$\text{S4.3}\Box\ \frac{\Gamma', \Box\Gamma' \vdash \varphi'_i, \Box\vec{\Phi}'_{-i}}{\Sigma, \Box\Gamma' \vdash \Box\vec{\Phi}, \Box B, \Delta'}$$

Combining the results for single formula on both sides of a sequent, followed by a simple induction on multisets gives the main result.

Theorem 5.2.1 (`gs43_wk_adm`)**.** *Weakening is admissible for the calculus* `gs43_rls`*.*

Proof. Pen and paper proof described above. Like the pen and paper proof, the Isabelle proof requires an induction principle on both rule applications and formula rank, this is `gen_step_lem` (Lemma 4.2.3). See the appendix for the full proof script. ⊣

5.2.3 Invertibility and Contraction

Like S4, inversion lemmas are proved for the G3cp and Refl rules within the overall calculus. The only notable case occurs for the G3cp rules, where the last rule applied in the original derivation is S4.3$\Box$. If the original derivation is given by

$$\text{S4.3}\Box\ \frac{\begin{matrix}\Pi\\ \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}\end{matrix}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

then proving invertibility for G3cp requires showing the derivability of all premises after applying a G3cp rule backwards from the endsequent of the S4.3$\Box$ rule. As the classical rules do not operate on boxed formulae, the rule can only modify Σ or Δ:

$$\text{G3cp rule}\ \frac{\Sigma', \Box\Gamma \vdash \Box\vec{\Phi}, \Delta'}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

Clarifying again, invertibility of G3cp requires deriving $\Sigma', \Box\Gamma \vdash \Box\vec{\Phi}, \Delta'$. The usual tactic would apply another instance of the S4.3$\Box$ rule to the original premises, but bringing in a different context. However, this does not admit a proof if there are boxed formula in Σ' or Δ'. For example, if the G3cp rule is L$\wedge$ and the principal formula is $A \wedge \Box B$ then Σ' contains a boxed formula, $\Box B$, which cannot be introduced within the (box-free) context of a new S4.3$\Box$ rule application.

To accommodate this case, the premises of the modal rule are used to derive the conclusion without any context. Then weakening-admissibility is used to bring the remaining formulae in the premise of the G3cp rule:

$$\text{Weakening-admissibility}\ \dfrac{\dfrac{\Pi}{\Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}\ \text{S4.3}\Box}{\Sigma', \Box\Gamma \vdash \Box\vec{\Phi}, \Delta'}$$

Theorem 5.2.2 (`inv_rl_gs43_refl` and `inv_rl_gs43_lks`)**.** *`lkrefl` (Refl) and `lksss` (G3cp) are invertible within the calculus `gs43_rls`*

Proof. As above. The formal proofs are given in the Appendix. ⊣

Proving contraction-admissibility, for the cases where the last rule is from G3cp or Refl, is done in the same way as in S4. Difficulties arise within the S4.3□ case, however, and serve as an example of how a formalisation may find errors in rough pen and paper proof.

If the last rule used in the derivation was the S4.3□ rule, there are a two cases to consider. The case where the contraction-formula is parametric is handled by simply re-applying another instance of the S4.3□ rule like the S4 case. Similarly, when the contraction-formula is principal in the antecedent, then the proof proceeds like in S4. Specifically, one copy of $\Box A$ from $\Sigma, \Box A, \Box A, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta$ must be removed. The original derivation is:

$$\text{S4.3}\Box\ \dfrac{\dfrac{\Pi}{A, A, \Box A, \Box A, \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}}{\Sigma, \Box A, \Box A, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

By contracting twice using first the induction hypothesis on height, then the induction hypothesis on rank, on all premises followed by an application of the S4.3□ rule, the desired endsequent is obtained:

$$\text{S4.3}\Box\ \dfrac{\text{IH on rank}\ \dfrac{\text{IH on height}\ \dfrac{\dfrac{\Pi}{A, A, \Box A, \Box A, \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}}{A, A, \Box A, \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}}{A, \Box A, \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}}}{\Sigma, \Box A, \Box\Gamma \vdash \Box\vec{\Phi}, \Delta}$$

When the contraction-formula is principal in the succedent, then there are two cases of the premises to consider. Either a premise "un-boxes" one of the contraction-formulae[2], or it leaves both boxed. The original deduction is given by:

$$\dfrac{\dfrac{\Pi_1}{\Gamma, \Box\Gamma \vdash \Box\vec{\Phi}, A, \Box A} \qquad \dfrac{\Pi_2}{\Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A, \Box A}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Box A, \Box A, \Delta}$$

In the latter case, the induction hypothesis can be directly applied, removing one copy of the boxed formulae:

[2]Technically, there are two syntactically identical premises which individually un-box one of the two copies of $\Box A$

$$\text{IH on height}\ \frac{\begin{array}{c}\Pi_2\\ \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A, \Box A\end{array}}{\Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A}$$

For the former case however, the induction hypothesis cannot be applied, as the contraction-formulae are no longer copies of another. One attempt to resolve this begins by attempting another nested induction within the overall Contraction proof, essentially stating that the following rule is admissible:

$$\frac{\Gamma \vdash \Delta, A, \Box A}{\Gamma \vdash \Delta, A}$$

However, when attempting to formalise an inductive proof for this lemma, a problem was encountered when the last rule applied above the premise is principal on A, which ends up modifying the antecedent. One such example is if A is a negation $\neg B$, and the last rule is R$\neg$. In this case, there is no obvious way to apply some inductive hypothesis to derive the conclusion. Following a suggestion from Jeremy Dawson, the pen and paper proof was then modified to prove the following lemma (which drops the un-boxed A in the premise) *before* proving ctra.

Lemma 5.2.3 (`gs43_refl`)**.** *The rule R-refl is admissible in* `gs43_rls`*.*

$$\textit{R-refl}\ \frac{\Gamma \vdash \Delta, \Box A}{\Gamma \vdash \Delta, A}$$

The corresponding statement of the lemma in Isabelle states that if a sequent `seq` *is derivable in* `gs43_rls` *and the sequent is equivalent to $X \vdash Y, \Box A$ for any X and Y, then $X \vdash Y, A$ is also derivable.*

Proof. By an induction on height, for all Γ and Δ. The analysis is on the last rule applied in deriving $\Gamma \vdash \Delta, \Box A$.

Case 1. The last rule applied was id. If $\Box A$ is parametric then $\Gamma \vdash \Delta$ is an axiom, and the conclusion will be also. If $\Box A$ is principal, then $\Gamma = \{\Box A\} \cup \Gamma'$ and the following transformation is applied:

$$\text{Refl}\ \frac{\text{id}\ \dfrac{}{A, \Box A, \Gamma' \vdash \Delta, A}}{\Box A, \Gamma' \vdash \Delta, A}$$

Case 2. The last rule applied was from G3cp. No rules in G3cp operate on a boxed formula, so $\Box A$ must be parametric. The induction hypothesis on height can thus be applied to the premise of the G3cp rule. Applying the original G3cp on the resulting sequent gives the desired conclusion.

Case 3. The last rule applied was Refl. Like Case 2, $\Box A$ must be parametric, as Refl only operates on boxed formula in the antecedent.

Case 4. The last rule applied was S4.3$\Box$. Then one premise of the original deduction un-boxes $\Box A$. Using Refl followed by weakening-admissibility on this premise is enough to produce the conclusion. For clarify, here we express $\Gamma = \Sigma \cup \Box\Gamma'$ and $\Delta = \Box\vec{\Phi} \cup \Delta'$. The original derivation is:

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma', \Box\Gamma' \vdash \Box\vec{\Phi}, A & \Gamma', \Box\Gamma' \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A\end{array}}{\Sigma, \Box\Gamma' \vdash \Box\vec{\Phi}, \Delta', \Box A}$$

This is transformed into:

$$\text{Weakening-admissibility}\ \dfrac{\text{Refl}\ \dfrac{\Pi_1}{\Gamma', \Box\Gamma' \vdash \Box\vec{\Phi}, A}}{\Sigma, \Box\Gamma' \vdash \Box\vec{\Phi}, \Delta', A}$$

Once again, the Isabelle proof script can be found in the Appendix. ⊣

Returning back to the proof of contraction-admissibility, recall that the original derivation is:

$$\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2\\ \Gamma, \Box\Gamma \vdash \Box\vec{\Phi}, A, \Box A & \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A, \Box A\end{array}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Box A, \Box A, \Delta}$$

To contract one copy of $\Box A$, the result of the lemma is applied to transform the premise resulting from Π_1 into a sequent which contains two copies of the un-boxed contraction-formula. The induction hypothesis on rank can then be applied. The final step is an application of the S4.3$\Box$ rule on the newly generated premises to produce the required sequent.

$$\text{S4.3}\Box\ \dfrac{\begin{array}{cc}\text{IH on rank}\ \dfrac{\text{R-refl}\ \dfrac{\Pi_1}{\Gamma, \Box\Gamma \vdash \Box\vec{\Phi}, A, \Box A}}{\Gamma, \Box\Gamma \vdash \Box\vec{\Phi}, A, A} & \text{IH on height}\ \dfrac{\Pi_2}{\Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A, \Box A}\\ \Gamma, \Box\Gamma \vdash \Box\vec{\Phi}, A & \Gamma, \Box\Gamma \vdash \varphi_i, \Box\vec{\Phi}_{-i}, \Box A\end{array}}{\Sigma, \Box\Gamma \vdash \Box\vec{\Phi}, \Box A, \Delta}$$

Theorem 5.2.4 (`gs43_ctr_adm`)**.** *Contraction is admissible in* `gs43_rls`*.*

Proof. Pen and paper proof given above. Formalised proof given in the Appendix. ⊣

5.2.4 Cut

Due the modularity of the calculus, the cut-admissibility proof is the same as in S4, with the exception of the cases where S4.3$\Box$ is the last rule applied above the cut (on either side). Although, there is a slight difference for the cases where S4.3$\Box$ is used in combination with or a `lksss` rule will always be parametric. Unlike in S4, where Lemma 4.2.12 applies, the restriction on the S4.3$\Box$ rule's context requires using the second parametric lemma, see Lemma 4.2.13.

When S4.3$\Box$ leads to the left cut-sequent, and the Refl rule is used on the right, the transformation mimics the corresponding case for S4. However, the case where S4.3$\Box$ is principal on both sides requires a new transformation. For clarity, the premises

above the S4.3$\Box$ rule on the left are given as two cases, depending on whether the cut-formula is un-boxed or not. The boxed formula in the succedents of the premises are also distinguished by the superscripts L and R for left and right cut premises respectively. Explicitly, these are $\vec{\Phi}^L = \{\varphi_1^L \dots \varphi_i^L \dots \varphi_n^L\}$ and $\vec{\Phi}^R = \{\psi_1^R \dots \psi_k^R \dots \psi_m^R\}$. The original cut thus has form:

$$\dfrac{\dfrac{\dfrac{\Pi_L^a}{\Gamma_L, \Box\Gamma_L \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box A}\text{S4.3}\Box \quad \dfrac{\Pi_L^b}{\Gamma_L, \Box\Gamma_L \vdash A, \Box\vec{\Phi}^L}}{\Sigma_L, \Box\Gamma_L \vdash \Box\vec{\Phi}^L, \Delta_L, \Box A}\text{Cut on } \Box A \quad \dfrac{\dfrac{\Pi_R}{A, \Box A, \Gamma_R, \Box\Gamma_R \vdash \psi_k^R, \Box\vec{\Phi}_{-k}^R}\text{S4.3}\Box}{\Box A, \Sigma_R, \Box\Gamma_R \vdash \Box\vec{\Phi}^R, \Delta_R}}{\Sigma_L, \Sigma_R, \Box\Gamma_L, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \Box\vec{\Phi}^R, \Delta_L, \Delta_R}$$

To remove this cut, the derivation is transformed into one where the principal rule (S4.3$\Box$) is applied last to produce the desired endsequent. The problem is then proving that the premises of the following S4.3$\Box$ rule application are derivable. This in itself requires two different transformations of the original derivation, depending on the two forms that the premises can take; either the un-boxed formula in the succedent originated from the left cut premise, that is from $\Box\vec{\Phi}^L$, or from the right, within $\Box\vec{\Phi}^R$. These cases are named $\mathcal{P}_L$ and $\mathcal{P}_R$ respectively. The final S4.3$\Box$ rule used in our new transformation is then:

$$\text{S4.3}\Box\ \dfrac{\dfrac{\mathcal{P}_L}{\Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box\vec{\Phi}^R} \quad \dfrac{\mathcal{P}_R}{\Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \psi_k^R, \Box\vec{\Phi}_{-k}^R}}{\Sigma_L, \Sigma_R, \Box\Gamma_L, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \Box\vec{\Phi}^R, \Delta_L, \Delta_R}$$

For both transformations, the same idea behind the principal S4$\Box$ rule case is used. We first derive the original cut-sequents but without their original contexts. These new sequents will be called $\mathcal{D}_L$ and $\mathcal{D}_R$ respectively. These are derived using the derivations in the original cut, but applying new instances of the S4.3$\Box$ rule. Importantly, the new sequents $\mathcal{D}_L$ and $\mathcal{D}_R$ have the same height as the original cut-sequents.

$$\text{S4.3}\Box\ \dfrac{\dfrac{\Pi_L^a}{\Gamma_L, \Box\Gamma_L \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box A} \quad \dfrac{\Pi_L^b}{\Gamma_L, \Box\Gamma_L \vdash A, \Box\vec{\Phi}^L}}{\mathcal{D}_L = \Box\Gamma_L \vdash \Box\vec{\Phi}^L, \Box A}$$

$$\text{S4.3}\Box\ \dfrac{\dfrac{\Pi_R}{A, \Box A, \Gamma_R, \Box\Gamma_R \vdash \psi_k^R, \Box\vec{\Phi}_{-k}^R}}{\mathcal{D}_R = \Box A, \Box\Gamma_R \vdash \Box\vec{\Phi}^R}$$

Having introduced all the necessary notation and pre-requisites, the first actual case to consider is deriving $\mathcal{P}_L$. The induction on level allows $\mathcal{D}_R$ to be cut, on cut-formula $\Box A$, with all of the sequents given by the derivation Π_L^a above the original left S4.3$\Box$ rule. The transformation performs n cuts, for all premises corresponding to the formulae in $\vec{\Phi}^L$. The results of this cut then match exactly with $\mathcal{P}_L$ after using the admissibility of Weakening to introduce the formulae of Γ_R in the antecedent.

$$\dfrac{\dfrac{\dfrac{\Pi_L^a \qquad \mathcal{D}_R}{\Gamma_L, \Box\Gamma_L \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box A \qquad \Box A, \Box\Gamma_R \vdash \Box\vec{\Phi}^R}\ \text{Cut on } \Box A}{\Gamma_L, \Box\Gamma_L, \Box\Gamma_R \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box\vec{\Phi}^R}\ \text{Weakening-admissibility}}{\mathcal{P}_L = \Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash \varphi_i^L, \Box\vec{\Phi}_{-i}^L, \Box\vec{\Phi}^R}$$

To derive the sequents in $\mathcal{P}_R$, the induction hypothesis on level is used to cut $\mathcal{D}_L$ with all of the premises above the right S4.3$\Box$ in the original cut, with cut-formula $\Box A$. The induction on formula rank on A is then used to cut the sequent resulting from Π_L^b with all these new sequents. Finally, contraction-admissibility allows the removal of the extra copies of $\Box\Gamma$ and $\Box\vec{\Phi}^L$, and concludes the case.

$$\dfrac{\dfrac{\dfrac{\Pi_L^b \qquad \dfrac{\mathcal{D}_L \qquad \Pi_R}{\Box\Gamma_L \vdash \Box\vec{\Phi}^L, \Box A \qquad A, \Box A, \Gamma_R, \Box\Gamma_R \vdash \psi_k^R, \Box\vec{\Phi}_{-k}^R}\ \text{Cut on } \Box A}{\Gamma_L, \Box\Gamma_L \vdash A, \Box\vec{\Phi}^L \qquad \Box\Gamma_L, A, \Gamma_R, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \psi_k^R, \Box\vec{\Phi}_{-k}^R}\ \text{Cut on } A}{\Gamma_L, \Box\Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \Box\vec{\Phi}^L, \psi_k^R, \Box\vec{\Phi}_{-k}^R}\ \text{Contraction-admissibility}}{\mathcal{P}_R = \Gamma_L, \Box\Gamma_L, \Gamma_R, \Box\Gamma_R \vdash \Box\vec{\Phi}^L, \psi_k^R, \Box\vec{\Phi}_{-k}^R}$$

To conclude, the transformations above derive $\mathcal{P}_L$ and $\mathcal{P}_R$ while reducing cut-level or cut-rank. These are the premises of an instance of the S4.3$\Box$ rule which results in the conclusion of the original cut. This completes the cut-admissibility proof.

Theorem 5.2.5 (`gs43_cas`)**.** *Cut is admissible in the calculus* `gs43_Rls`*.*

Proof. As described above. In Isabelle, `gs43_cas` corresponds to

```
(?Xl |- mins ?A ?Yl, mins ?A ?Xr |- ?Yr) : cas gs43_Rls ?A
```

See the Appendix for the full formalised proof. ⊣

Chapter 6

Automated Theorem Proving: IntLSJGC

While the main focus of this thesis is on the formalisation of cut-admissibility proofs in ITPs, the motivation for such proofs have been approached from the perspective of automated theorem proving. For ATPs based on sequent style backward proof search, the necessity of having structural-free calculi calculi for efficient automated reasoning cannot be overstated. However, many other optimisations are generally required in order to produce an implementation which can potentially used in practice.

This chapter showcases a new prover for Intuitionistic Propositional Logic (**Int**), named IntLSJGC, and a number of the optimisations which are used in order to speed up search. The automated reasoners mentioned within this Chapter all perform object level reasoning, as opposed to the meta-theoretic proofs serving as the main focus of this work. Rather than manipulating given derivation trees as in cut-elimination, these provers can be considered to create a derivation[1] for each particular input sequent – in order to prove its validity or otherwise.

Parts of this chapter have been previously published for the prover IntHistGC, see the paper [Goré, Thomson and Wu 2014] . In fact, the optimisation techniques of IntHistGC are used as a basis for the new prover. The difference between the two provers consists of the underlying calculus and backward proof search strategy. IntLSJGC itself is implemented in OCaml.

6.1 Related Work

Although ATPs are mostly based on Classical Logics, the development of such provers for **Int** are also important for a number of applications. Research into logic programming and verification [Osorio et al. 2003; Fairtlough and Mendler 1994] are examples of areas which are particularly suited to the use of such provers. In the literature, the techniques used to improve search can be broadly generalised into two categories, either by creating more efficient calculi, or by optimising the implementation of the

[1]Note however that some provers do not produce an actual derivation of any sort, serving only as a decision procedure. Furthermore, provers generally perform a significant number of modifications on sequents, and so those which do produce derivations may not produce a tree for the exact original input.

search procedure. The best theorem provers generally combine optimisations in both areas. To serve as points of comparison and representatives of the state of the art, IntLSJGC is benchmarked against four provers for **Int**. This section briefly describes the main techniques used by each of these provers.

BDDIntKt [Goré and Thomson 2012]
This prover does not use a specific calculus, but instead attempts to find a (counter) model which makes an input formula φ_0 false. This is done by constructing the closure of φ_0, equivalent to the set of all subformulae of φ_0 for **Int**, and checking if the subsets of this closure lead to a falsification of φ_0. By using Binary Decision Diagrams (BDDs) as the core underlying data structure, many of the otherwise exponentially many subsets do not have to be explicitly created.

The impact of variable ordering is known to have a significant impact on the efficiency of BDD representations. The performance of BDDIntKt similarly varies depending on the variable reordering approaches specified at runtime. Various options have been implemented, allowing for various different combinations of ordering heuristics at various points of the program execution. The implementation also considers structure sharing, to allow multiple formula to be represented using one BDD. There are two versions of this ATP, written in OCaml and C++ respectively, the results in the section below are for the C++ configuration.

fCube [Ferrari et al. 2010]
fCube is a Prolog prover based on a tableaux calculus for **Int**, which relies on a variety of simplification techniques. The actual calculus, **Tab**, is based on a tableaux version of Dychkoff's calculus for **Int** but has extended rules for negation and implication in order to improve performance.

The simplification rules are crucial in reducing search space, doing so by replacing formulae with logically equivalent but simpler formulae. By reducing the complexity of formulae, fCube explores a significantly reduced search space. The actual simplification is done by analysing the polarity and sign of formula, within certain contexts and stages of a deduction, and often allows the prover to replace formulae with $\top$ or $\bot$. Polarity and sign of formulae are determined recursively; from a very simplified sequent perspective positive and negative polarities are based on the position of the given formulae both in the side of the sequent as well as within implications.

Imogen [McLaughlin and Pfenning 2008]
Imogen uses the polarity of formulae to restrict applicable inference rules. The prover begins by converting input formulae into a polarised form, as defined below, using heuristics to do so with as few shifts ($\uparrow, \downarrow$) as possible.

$$\text{Positive formulae } A^+ ::= P^+ \mid A^+ \vee A^+ \mid \bot \mid A^+ \wedge A^+ \mid \top \mid \downarrow A^-$$
$$\text{Negative formulae } A^- ::= P^- \mid A^+ \rightarrow A^- \mid A^- \bar{\wedge} A^- \mid \bar{\top} \mid \uparrow A^+$$

Proof search itself is applied in a forward manner, by using the focused inverse method, unlike the backward strategy used by many other provers. Two databases of

sequents are used, a *kept* and an *active* database. During inference, all possible rules are applied to a sequent from the kept database. The results are added to the same database, while the selected sequent is moved to the active database. The process repeats until the input sequent is found, or the kept database is exhausted.

IntHistGC [Goré et al. 2014]
The fundamental optimisation of this prover is the use of global caching. This technique uses the results of previous branches in a backward proof strategy to avoid the exploration of other very similar branches. Our new prover IntLSJGC is based on our previous work on IntHistGC, the most significant optimisations used by both provers are expanded upon in Section 6.3.

6.2 LSJ

IntLSJGC was developed in an attempt to maximise the effect of global caching. As stated before, there are two primary approaches in improving efficiency, either in choosing a better calculus, or by implementing better data structures or heuristics. IntLSJGC concentrates on the former. **Int** is known to be PSPACE-complete [Statman 1979], so derivation trees will be inevitably exponential in size. Nonetheless, the impact of caching can be improved by using calculi which reduce tree depth.

The key observation is that global caching (see Section 6.3) becomes useful when branches in the search space are similar to sequents which have been previously examined and stored. Ideally, sequents should be added to the cache as quickly and as early as possible during proof search, and furthermore it would be beneficial if branches were likely to contain the same core formulae.

The first property requires derivations with small depth, so that initial branches can be exhausted and added to the cache quickly. Conditions benefiting the second property are more difficult to articulate. Improved bounds on depth must be traded for worse breadth, ideally this increase in breadth will produce derivation branches which are very similar, if not identical, in many situations. Even if not identical, if many of these newly introduced branches can be solved by cache hits in less time than fully exploring the original deeper search trees, performance should improve.

The original prover IntHistGC uses the sequent calculus IG [Corsi and Tassi 2007], which gives proofs with an upper bound of n^2 depth in the size of the input. Meanwhile, the LSJ calculus, developed by the authors of fCube [Ferrari et al. 2013], has a significantly better guarantee – proofs are linear (n) in depth. The calculus itself is shown in Figure 6.1. The trade-off is that LSJ has greater branching compared to IG, as the L$\rightarrow$ in LSJ rule has three premises, while all other rules have the same number of premises as the corresponding rules in IG.

Linear depth is achieved using an extra "compartment" alongside each sequent. This compartment is distinguished from the rest of a standard sequent using ";". The formulae within such a compartment are generally represented using Θ.

The semantic meaning behind these formulae can be understood in terms of the

Axioms

$$\mathrm{L}\bot\ \frac{}{\Theta; \bot, \Gamma \vdash \Delta} \qquad \mathrm{id}\ \frac{}{\Theta; \Gamma, \varphi \vdash \varphi, \Delta}$$

Static Rules

$$\mathrm{L}\wedge\ \frac{\Theta; \Gamma, \varphi, \psi \vdash \Delta}{\Theta; \Gamma, \varphi \wedge \psi \vdash \Delta} \qquad \mathrm{R}\wedge\ \frac{\Theta; \Gamma \vdash \varphi, \Delta \qquad \Theta; \Gamma \vdash \psi, \Delta}{\Theta; \Gamma \vdash \varphi \wedge \psi, \Delta}$$

$$\mathrm{L}\vee\ \frac{\Theta; \Gamma, \varphi \vdash \Delta \qquad \Theta; \Gamma, \psi \vdash \Delta}{\Theta; \Gamma, \varphi \vee \psi \vdash \Delta} \qquad \mathrm{R}\vee\ \frac{\Theta; \Gamma \vdash \varphi, \psi, \Delta}{\Theta; \Gamma \vdash \varphi \vee \psi, \Delta}$$

Transitional Rules

$$\mathrm{L}\to\ \frac{\Theta; \Gamma, \psi \vdash \Delta \qquad \psi, \Theta; \Gamma \vdash \varphi, \Delta \qquad \psi; \Theta, \Gamma \vdash \varphi}{\Theta; \Gamma, \varphi \to \psi \vdash \Delta}$$

$$\mathrm{R}\to\ \frac{\Theta; \Gamma, \varphi \vdash \psi, \Delta \qquad \emptyset; \psi, \Theta, \Gamma \vdash \varphi}{\Theta; \Gamma \vdash \varphi \to \psi, \Delta}$$

Figure 6.1: The LSJ calculus

Kripke semantics for **Int**. Recall the semantics given in Section 2.1.1, where the $\to$ operator is the only one which requires the notion of multiple worlds. The L$\to$ and R$\to$ rules are then the only rules which may "jump" to some new world (reachable using the relation R). Due to persistence, the truth of a particular formula can only change from "not true" to become "true" in a reachable world. The compartment can be thought to contain formulae which are not yet true, but become true in an immediate successor of the current world.

The L$\to$ and R$\to$ rules are the only ones which modify this compartment. The premises of these rules which move formulae from the compartment to the antecedent of a sequent can be viewed as jumping to a (the first) world where the compartment formulae become true. The soundness of this calculus is based on the fact that **Int** has the finite model property [Ferrari et al. 2013] (**Int** can be expressed using finite Kripke models) and that if a formula is not true, there is a last world where it is not true.

Unfortunately, the third premise of the L$\to$ and the second premise of the R$\to$ rule are not invertible. Thus, these premises require backtracking. That is, given an application of either rule on a particular formula $\varphi \to \psi$ where the non-invertible premise does not lead to a sequent proof, the search must return to this application and try the rule on all other implication formulae in the sequent. The endsequent is not valid only if *all* possible applications of either rule do not lead to a sequent proof.

The general backward search strategy for the calculus is given by the following:

`while` some rule is applicable to a leaf sequent **`do`**

`stop:` apply any applicable Axiom rule to that leaf

`saturate:` apply any applicable static rule to that leaf

`trans:` apply any applicable transitional rule to that leaf

While the choice of leaf nodes to explore can be done in various ways, depth-first search is the most appropriate for caching. This is made obvious by the need to fully explore branches before caching entries becomes possible. With this in mind, further details for the `trans` phase, particularly the handling of backtracking, can be provided. This part of the overall search is implemented using loops to potentially apply transitional rules to all possible formulae. Given a sequent $\Theta; \Gamma \vdash \Delta$ the search performs the following:

define: `leftimps` to be the set of implications in Γ
while: `leftimps` is non-empty
    select an implication $\varphi \to \psi$ from `leftimps` and apply the L$\to$ rule
        if the first premise is unprovable **then return** unprovable
        else if the second premise is unprovable **then return** unprovable
        else if the third premise is provable **then return** provable
        else `leftimps` := `leftimps` $\setminus \{\varphi \to \psi\}$
define: `rightimps` to be the set of implications in Δ
while: `rightimps` is non-empty
    select an implication $\varphi \to \psi$ from `rightimps` and apply the R$\to$ rule
        if the first premise is unprovable **then return** unprovable
        else if the second premise is provable **then return** provable
        else `rightimps` := `rightimps` $\setminus \{\varphi \to \psi\}$
return unprovable

For each premise, depth-first search is used to recursively apply the overall backward search strategy. The results can potentially be used for early while loop termination if the invertible premises are found to be unprovable. Note that if all premises are provable, then a sequent proof has been found, and so the loop is similarly escaped by the corresponding return statement. If there are no implications, the procedure returns unprovable, as this means there are no more possible rules to apply (**trans** is applied after attempting to terminate via axioms, as well as after saturating).

6.3 Optimisations

This section describes the main optimisations behind IntLSJGC. We omit details for some of the heuristics used, such as lazy bi-implication expansion and rule choice, deferring instead to the paper [Goré et al. 2014].

Pre-processing - A simple processing function which replaces inputs with equivalent formulae can significantly reduce the complexity of a proof. By imposing a standard ordering on all formulae, one can completely remove any complications due to variable ordering at commutative operators. For example, the formula $(\varphi \wedge \psi) \to (\psi \wedge \varphi)$ can be rewritten to $(\varphi \wedge \psi) \to (\varphi \wedge \psi)$ which simplifies to $\top$.

Both IntHistGC and IntLSJGC also make use of standard logical identities for **Int**, such as $\varphi \wedge \top = \top \wedge \varphi = \varphi \wedge \varphi = \varphi$, and removes nested implications using the identity

$$\varphi \wedge \top = \top \wedge \varphi = \varphi \wedge \varphi = \varphi \qquad \varphi \wedge \bot = \bot \wedge \varphi = \bot$$
$$\varphi \vee \bot = \bot \vee \varphi = \varphi \vee \varphi = \varphi \qquad \varphi \vee \top = \top \vee \varphi = \top$$
$$\varphi \to \top = \bot \to \varphi = \varphi \to \varphi = \top \qquad \top \to \varphi = \varphi$$
$$\varphi_0 \to (\varphi_1 \to \varphi_2) = (\varphi_0 \wedge \varphi_1) \to \varphi_2 \qquad \top \to \bot = \bot$$

Figure 6.2: Basic logical identities in **Int**

$$\mathrm{L}\vee \dfrac{\mathrm{id}\dfrac{}{A, C \vdash C, D} \quad \mathrm{id}\dfrac{}{A, D \vdash C, D}}{\mathrm{L}\vee \dfrac{A, C \vee D \vdash C, D}{A \vee B, C \vee D \vdash C, D}\times \quad B, C \vee D \vdash C, D}$$

Figure 6.3: An example of backjumping.

$\varphi_0 \to (\varphi_1 \to \varphi_2) = (\varphi_0 \wedge \varphi_1) \to \varphi_2$ before applying backwards proof search. A full list of such identities are provided in Figure 6.2

Backjumping and Global Caching For calculi where Weakening is admissible, a proof of validity for a particular sequent is sufficient to prove the validity of any of its super-sequents. If the crucial formulae within a sequent can be extracted during proof search, one can then construct a minimal derivable sequent. The validity of other sequents, which are simply extensions of this minimal example by arbitrary contexts, follows immediately from the statement of weakening-admissibility.

Backjumping is a technique from the literature which uses this property at branching points within a search tree, to reduce search space [Hustadt and Schmidt 1998]. In backward proof search, if a rule splits a sequent into multiple premises, and these premises only differ by formulae which are not crucial for a derivation, then proving one premise is sufficient for all of them. In the example of Figure 6.3, the minimal sequent required for a proof is $C \vee D \vdash C, D$. Both premises created from the branching $\mathrm{L}\vee$ rule contain this minimal sequent, and so the right premise does not have to be explored once the left has been proven.

The actual process of determining such a minimal sequent is done by tracing down the applications of rules to formula which are used in an axiom. In our implementation, this is done by storing the principal formula modified at every rule instance, and using these to reconstruct the important formula in the conclusion after a recursive call has returned "provable" for the premises.

To begin with, global caching can be considered as essentially an extension of backjumping by storing minimal sequents and using these throughout the entire search tree, rather than only at a specific branching point. While potentially allowing for enormous improvements in search efficiency – in an ideal situation this would mean that effort will never be wasted in producing two instances of the same proof – in practice the optimisation is limited by memory and lookup considerations. Nevertheless, the optimisation proves to be very effective when implemented with appropriate heuristics and data structures, as shown by [Goré et al. 2014] and Section 6.4.

Unlike in proof theory, and backjumping, where sequent proofs are the only ob-

jects of interest, global caching places the same importance on failed derivations. If some sequent cannot be proven, then neither can any of its sub-sequents. At best, all the rule applications of the original sequent can be applied, which are known from previous experience to be insufficient for a sequent proof. Such unprovable sequents can also be stored, and used throughout the search procedure to terminate early when there is no chance of a proof. In contrast to the minimal examples for provable sequents, here the optimal addition to the cache is a maximal sequent.

6.3.1 Data Structures

From an entirely theoretical perspective, global caching only has benefits. The technique allows search space to be potentially reduced across all branches, by substituting previous proofs. In practice, searching for cache hits, as well as storing cache entries pose significant challenges. As **Int** is PSPACE-complete, caching sequents after all rule applications will result in an exponential growth in cache size. In addition to memory limits, performing sub-sequent and super-sequent checks for cache entries is also time-consuming.

A set-trie is a data structure that can be used to address both of these issues. Tries are already reasonably efficient at avoiding wasted storage when the objects involved can be tokenised and share similarities. In addition, the representation of sets[2] as ordered lists, gives rise to set containment queries which have been empirically found to be of linear time complexity [Savnik 2013]. A modification of the algorithms presented in [Savnik 2013] are implemented for IntLSJGC.

To allow for such an ordered representation of sequents, formulae are represented as integers. After pre-processing, every formula (and all subformulae) are given a unique positive integer identifier, with tables to keep track of the formula type as well as integers corresponding to any subformulae. A (hash) table is also used to map integers to their original formula (as a tree) representation. To produce a single ordered list for a sequent, these positive identifiers can be distinguished to have originated from the compartment, antecedent or the succedent by any appropriate trichotomy[3].

To further mitigate the memory usage of our global caches, additions are only considered at applications of the transitional rules or where backjumping has occurred. While essentially arbitrary choices, sequents at transitional points are comparatively structured due to the necessary previous saturation phase. This structure can aid in cache lookups[4]. Similarly, if backjumping has been applied, then some unnecessary rule applications must have been introduced by the search procedure. The same search procedure is then likely to introduce similar redundancies elsewhere.

[2]While sequents are defined to consist of multisets of formulae for the majority of this work, sets are implemented in IntLSJGC. This is possible as the calculus involved never requires multiple copies of a single formulae. Indeed, the original presentation of the calculus in [Ferrari et al. 2013] uses sets.

[3]The implementation of IntLSJGC uses ordered lists of integers. An identifier i is mapped to $2i + 1$, $2i$, or $-2i - 1$ for the compartment, antecedent or succedent respectively.

[4]Lookups themselves do not consider subformulae, so without saturation a hit for say $A \wedge (C \vee D) \vdash C, D$ given a minimal sequent $C \vee D \vdash C, D$ will not occur. The former is a not a super-sequent of the latter, but can become so after an application of the L$\wedge$ rule.

6.4 Results and Discussion

The Intuitionistic Logic Theorem Proving (ILTP) Library [Raths et al. 2007] is the standard benchmark used in the literature for **Int** provers. The performance of our prover is evaluated on this library, which allows a maximum of 600 seconds for each individual problem instance. Results are presented in Figure 6.4 for varying levels of optimisations, along with `BDDIntKt -autoreorder -nary -assumeimp` (BDD), `fCube` (no options), `Imogen -h optimize` and `IntHistGC` (option naming identical to ILSJ). We also include experiments on an unofficial extended set of the ILTP problem classes, kindly provided by the authors of the library, Jens Otten and Thomas Raths[5] in Figure 6.5. All tests were conducted on a machine with an Intel i7-3770 CPU @3.40GHz and 8GB of memory. The configurations for IntLSJGC are as follows:

Configuration	Optimisations involved
n	The naïve/baseline prover
b	Backjumping
c	Global caching with additions made at transitional rules
c2	Global caching with additions also at backjumping points
c3	Heuristic to perform cache lookups only every second node

The most interesting result from Figure 6.4 is the improvement of the naïve version of the IntLSJGC over IntHistGC. The two provers are very apt for comparison, given that their implementations are essentially equal apart from the calculus used. The basic LSJ calculus seems to offer an advantage over IG in a number of classes, only performing worse in SYJ205. However, from the ILTP problems, it still seems that the use of backjumping and global caching is still the greatest factor in prover efficiency. With these optimisations, both provers solve the same number of problems, even on the extended ILTP library.

For a more fine grained analysis of prover performance, rather than only instances solved, Figure 6.5 includes a measure of total time taken (with a maximum of 10 minutes added for timeouts). Here it becomes evident that while the best configurations of IntLSJGC and IntHistGC solve the same problems, IntLSJGC does so faster. Interestingly, the best result in terms of time for IntHistGC (IHbcc3) uses a heuristic which only performs cache lookups every second node. When the same heuristic was used for LSJ (not shown), performance was significantly worse than without (ILSJbc2).

It should be noted that fCube remains the best prover for the ILTP formulae families, both in terms of instances solved, and total time. However, it is also clear that a new set of benchmarks are required for **Int**. Most state of the art provers have little problem with the standard ILTP benchmark (with the exception of the pigeonhole class SYJ202). fCube, IntHistGC and IntLSJGC are also able to solve all problems on the extended problem set, once again with the exception of the SYJ202. This issue is recognised in [Goré et al. 2014], where three other families of formulae are used: Portia, which are encodings of a "real world puzzle"; Nishimura [Fiorentini 2000], previously used by the authors of fCube; and randomly generated formulae.

[5] Available at: `http://www.iltp.de/download/SYJ2xx-50/SYJ2xx-50.tar.gz`

Class	ILSJn	ILSJbc2	IHn	IHbc2	BDD	fCube	Imogen	Out of
LCL	2	2	2	2	2	2	2	2
SYN	20	20	20	20	20	20	20	20
SYJ10*	12	12	12	12	12	12	12	12
SYJ201	4	20	2	20	20	20	20	20
SYJ202	3	8	3	8	13	9	8	20
SYJ203	15	20	10	20	20	20	20	20
SYJ204	20	20	20	20	20	20	20	20
SYJ205	9	20	10	20	20	20	20	20
SYJ206	20	20	20	20	20	20	20	20
SYJ207	5	20	3	20	20	20	20	20
SYJ208	12	20	10	20	11	20	20	20
SYJ209	10	20	9	20	20	20	20	20
SYJ210	20	20	20	20	20	20	20	20
SYJ211	8	20	4	20	20	20	20	20
SYJ212	20	20	20	20	20	20	20	20
Total	180	262	165	262	258	263	262	274

Figure 6.4: Performance on the ILTP benchmark.

Class	ILSJbc	ILSJbc2	IHbcc3	IHbc2	BDD	fCube	Imogen	Out of
SYJ201	50	50	50	50	50	50	36	50
SYJ202	5	8	6	8	13	9	8	38
SYJ203	50	50	50	50	50	50	50	50
SYJ204	50	50	50	50	50	50	50	50
SYJ205	50	50	50	50	50	50	50	50
SYJ206	50	50	50	50	50	50	50	50
SYJ207	50	50	50	50	49	50	26	50
SYJ208	38	38	38	38	11	38	37	38
SYJ209	50	50	50	50	50	50	50	50
SYJ210	50	50	50	50	50	50	50	50
SYJ211	50	50	50	50	45	50	50	50
SYJ212	50	50	50	50	50	50	46	50
Total	543	546	544	546	518	547	503	576
Time	355	329	365	382	654	305	850	5760

Figure 6.5: Performance on extended ILTP formulae.

Time is the total time taken given in minutes, with a maximum of 10 minutes added for each problem instance.

Results on random formulae are shown in Figure 6.6. These show average time taken for 1000 instances of randomly generated formulae of increasing size[6]. Timing results for difficult instances of Portia and Nishimura, as well some from ILTP are displayed in Figure 6.7.

While performing slightly better than IntHistGC on the ILTP problems, IntLSJGC does significantly worse on the Portia problems. The problems themselves contain a very large number of implications (translated from negations), and it seems that the trade-off for extra branching of the LSJ calculus is detrimental for these formulae.

For random formulae, which tend to be unprovable with growing size, the new prover has its best showing yet. IntLSJGC has the best performance out of all provers in Figure 6.6, with a particularly noteworthy improvement in scaling compared to the results of IntHistGC.

Although IntLSJGC has poor performance on Portia overall, the problem class still serves as a meaningful point of interest. There are two variants of the unprovable Portia problems, differing only in the order of atomic propositions when given as input formulae. IntLSJGC only manages to solve one ordering, timing out on the V2 instance. Similarly IntHistGC and fCube have significantly different times depending on the version of the problem used, highlighting the effects of input ordering within search procedures. Roughly speaking, the problems of the class contain many provable branches, increasing in number with problem size, but always one unprovable branch. Slow times for a prover are then explained by the search procedure exploring the provable branches first, in contrast to fast times where the unprovable branch is examined early on. Heuristics to potentially identify such branches are then one future avenue for improving performance.

[6]Where size is approximately the sum of the number of atomic propositions and logical operators. One new atomic proposition is introduced for roughly every increase of 10 in size.

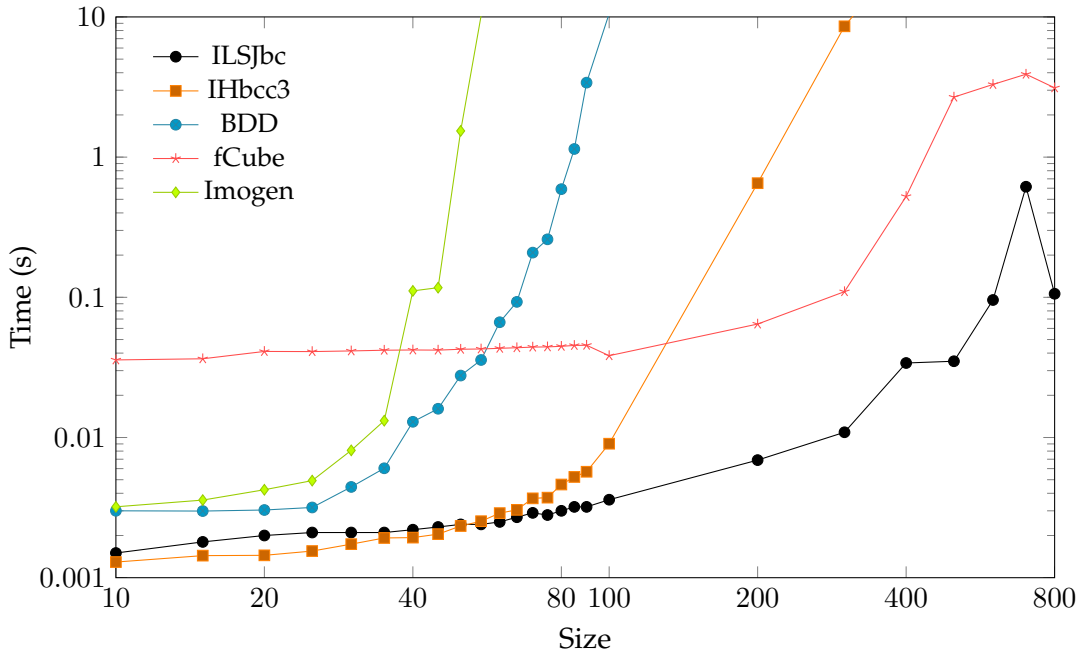

Figure 6.6: Average times for random formula of increasing size.
The spike for ILSJbc at size 700 is due to one instance of a randomly generated formulae being particularly difficult for IntLSJGC.

Problem	ILSJbc	ILSJbc2	IHbc2	IHbcc3	BDD	fCube	Imogen
SYJ201+1.050	23.45	23.94	10.13	4.58	159.92	31.79	–
SYJ202+1.008	–	192.53	190.16	–	0.33	27.86	70.76
SYJ207+1.050	3.75	3.84	454.69	161.73	188.36	17.90	–
SYJ208+1.038	186.82	185.86	185.99	113.32	–	1.76	–
SYJ211+1.050	12.23	12.16	1.37	10.15	–	0.02	0.27
SYJ212+1.050	0.01	0.01	0.26	0.22	50.91	0.22	–
p-portia.100	–	–	–	238.00	–	334.02	–
n-portia.100	42.85	42.21	295.84	4.52	–	385.12	–
n-portiaV2.100	–	–	–	239.08	–	97.62	–
nishimura.035	0.51	0.51	0.51	0.51	7.05	–	–

Figure 6.7: Timing comparisons (all times in seconds, – indicates a timeout)
'p' means provable, 'n' is non-provable and V2 is a version with reordered formulae. Shading indicates the best time(s) for each problem.

Chapter 7

Conclusion

The main contributions of this work are admissibility proofs for structural-free sequent calculi describing S4 and S4.3. For S4, we have provided the first formalised proofs of weakening, contraction and cut-admissibility, as well as a new pen and paper proof for the modal case in the cut-elimination proof. Our proof of cut-elimination has the advantage of being significantly simpler than existing proofs. For S4.3, we describe a logical sequent calculus for the logic, and give both the first formalised and the first syntactic pen and paper proofs of weakening, contraction and cut-admissibility.

The formalisations of both calculi are implemented in Isabelle, providing a machine checked proof of correctness for all admissibility proofs. The code itself is available at: `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/`. The necessary directories required to run the proofs are "gen", "pmgms" and "seqms". The files for S4 and S4.3 are within the "seqms" directory, named "gs4" and "gs43" respectively.

We have also implemented a new ATP for **Int**. The prover utilises a linear depth calculus, LSJ, in combination with a number of search reduction techniques, the most crucial of which is global caching. Results show that IntLSJGC is competitive with the state of the art on many classes of formulae in the ILTP library, and is world leading on randomly generated formulae.

7.1 Future work

The most obvious extension of this work is to continue research into new calculi and logics. The definitions and theorems of the Isabelle formalisation provide a solid foundation from which machine checked proofs of admissibility for more complex calculi can be developed. One potential calculus for future work is SKM.LC for the logic KM.LC [Clouston and Goré 2014]. The calculus itself includes elements of provability logic GL, by transferring formulae from the succedent to the antecedent, as well as S4.3, in the use of an indeterminate number of premises. Given that formalisations and admissibility results have been shown for both GL (in the base formalisation) and S4.3 (described in this work), the next step would apply these ideas to SKM.LC.

Another avenue for future work is formally proving cut-elimination for calculi satisfying the sufficient conditions of [Rasga 2005]. Alternatively, a program could be implemented using Rasga's framework to generate a cut-elimination proof given a

calculus satisfying the sufficient conditions.

The Isabelle formalisations could also potentially be improved. Porting the code to the latest versions of Isabelle would make the code base more accessible to other researchers. Alternatively, cut-admissibility proofs could perhaps be constructed in a manner conducive to Isabelle's program extraction capabilities. Ideally Isabelle could be used to automatically generate programs checking validity, following formalisations of cut-admissibility and completeness. This in turn could significantly aid the development of ATPs, essentially providing a machine-checked base implementation.

Additionally, the techniques of the theorem proving aspect of this work can be extended to the logics discussed for the proof theory sections. In Chapter 5 we note a lack of efficient ATPs for S4.3; a new prover utilising the logical sequent calculus presented in the Chapter, and the global caching techniques of IntHistGC and IntLSJGC is one clear possibility for future development.

Appendix

A.1 Code

Note that the following Isabelle proof scripts cannot be understood fully without inspecting the entirety of the Isabelle files for the formalisation. They are included here to demonstrate their complexity, and general structure.

A.1.1 Core proofs for S4

```
(* Weakening-admissibility *)
val _ = qed_goal "gs4_ext_concl" gs4.thy "ext_concl gs4_rls"
  (fn _ => [ (simp_tac (esimps [gs4_def]) 1),
    (REPEAT_ALL_NEW (rtac ext_concl_Un) 1),
    (ALLGOALS (simp_tac (esimps [lksss_extrs]))) ]) ;

val _ = bind_thm ("gs4_wk_adm", gs4_ext_concl RS wk_adm_ext_concl) ;

(* Invertibility of most rules *)

(* lkrefl invertible as principal formula copied to premise,
   and as wk_adm holds *)
val _ = qed_goal "inv_rl_gs4_refl" gs4.thy
  "Ball (extrs lkrefl) (inv_rl gs4_rls)"
  (fn _ => [ (clarsimp_tac (cesimps [inv_rl.simps, drs.all]) 1),
    (etac (gs4_wk_adm RS wk_admD_alt) 1),
    (clarsimp_tac (ces [extrs_E, lkrefl.elims], esimps [extend_def]) 1) ]) ;

(* Use inv_step for the lksss rules *)
val _ = qed_goal "inv_rl_gs4_lks" gs4.thy "Ball lksss (inv_rl gs4_rls)"
  (fn _ => [ Clarify_tac 1,
    (rtac gen_inv_by_step 1), atac 2, Clarify_tac 1,
    (etac thin_rl 1), (etac gs4_rls.elims 1),
    (rewtac lksss_def), (ALLGOALS Clarsimp_tac),
  (* rule in lksss *)
    (dtac (extrs_UN_single RS equalityD2') 1),
    Clarify_tac 1,
    (etac lksne.elims 1), Safe_tac,
  (* extrs of axiom *)
    (stac (extrs_UN_single RS sym) 1),
    (rtac inv_step_UN 1),
    Clarify_tac 1,
    (etac extrs_E2 1), Clarsimp_tac 1,
    (etac lksne.elims 1), Safe_tac,
    (force_tac (ces [extrs.elims], esimps [inv_step.simps, invs_of_def]) 1),
```

```
(* symmetric cases for lksil and lksir *)
  (REPEAT_DETERM_N 2 (EVERY [
    (rtac (lks_id_decomp RS id_decomp_step1e'') 1),
    (rtac idrls.intros 4), (rtac lksne.axiom 3),
    (eresolve_tac [lksne.ilI, lksne.irI] 1),
    (eresolve_tac [lksil_scrls, lksir_scrls] 1),
    (Force_tac 1),
    (Clarify_tac 1), (rtac gs4_rls.lksI 1),
    (clarsimp_tac (cesimps [lksss_def]) 1) ] )),
(* lksil and lksir *)
  (REPEAT_DETERM_N 2 (EVERY [
    (stac (extrs_UN_single RS sym) 1),
    (rtac inv_step_UN 1),
    Clarify_tac 1,
    (etac lksne.elims 1), Safe_tac,
    (force_tac (ces [extrs.elims], esimps [inv_step.simps, invs_of_def]) 1),
    (TRYALL (EVERY' [ (rtac inv_step_scrls),
      (rtac refl), atac,
      (eresolve_tac [lksil_scrls, lksir_scrls] o incr),
      (eresolve_tac [lksil_scrls, lksir_scrls] o incr),
      Clarsimp_tac o incr,
      (eresolve_tac lksucs o incr), atac o incr ])),
    (TRYALL (EVERY' [
      Clarify_tac, (etac thin_rl), (etac extrs_E),
      (dresolve_tac [lksne.ilI, lksne.irI]),
      (dtac lksss.extI), (dtac gs4_rls.lksI), Force_tac ])) ] )),
(* rule in lkrefl *)
  (stac (extrs_UN_single RS sym) 1),
  (rtac inv_step_UN 1),
  Clarify_tac 1,
  (etac lksne.elims 1), Safe_tac,
  (force_tac (ces [extrs.elims], esimps [inv_step.simps, invs_of_def]) 1),
  (REPEAT_DETERM_N 2 (EVERY [
    (rtac inv_step_scrls 1),
    (rtac refl 1),
    (res_inst_tac [("flr", "flr"), ("psc", "(a, b)")] extrs.intros 1),
    (rtac singletonI 1), (Clarsimp_tac 1),
    (Clarify_tac 1), (etac extrs_E 1), (dtac gs4_rls.reflI 1), Force_tac 1,
    (etac lkrefl_scrls 1),
    (eresolve_tac [lksil_scrls, lksir_scrls] 1),
    (etac lkrefl.elims 1),
    (eresolve_tac [lksil.elims, lksir.elims] 1),
    (auto_tac (cesimps [Btimes, Bplus, Bimp, Bneg, Box])) ] )),
(* rule in lkbox *)
  (clarsimp_tac (ces [lkbox.elims, extrs.elims],
    esimps [inv_step.simps, invs_of_def]) 1),
  (etac lksne.elims 1),
  (Auto_tac),
  (REPEAT (EVERY [
    Clarify_tac 1,
    (forward_tac [lksil_atomic, lksir_atomic] 1),
    (dtac (atomic_le_sum RS iffD1) 1),
    (etac allE2 1), (etac impE 1),
    (rtac xt3 1),
    (full_simp_tac (esimps [extend_def]) 2),
    (rtac le_plus1 2),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (etac sym 1), (etac disjE 1),
    (eresolve_tac [UnI1 RS lks_not_Box, UnI2 RS lks_not_Box] 1), atac 1,
    (clarsimp_tac (cesimps [extend_def]) 1),
    (fatac (sum_seq_le_lem RS iffD1) 1 1),
    (ftac (thin_rl RS (lkbox.I RS gs4_rls.boxI RS drs.derI')) 1),
```

```
      Simp_tac 1,
      (subgoal_tac
        "xa + flra = extend (xa + flra - (mset_map Box gamma |- {#Box A#})) \
        \ (mset_map Box gamma |- {#Box A#})" 1),
      (clarsimp_tac (cesimps [extend_def]) 2),
      (rtac trans 2), (rtac sym 3),
      (rtac le_pm_assoc 3),
      (rtac xt6 3), atac 4, Simp_tac 3, Simp_tac 2,
      (rtac mem_same 1), (etac sym 2), atac 1]))
    ]) ;


(* Contraction-admissibility *)
val _ = qed_goal "gs4_ctr_adm" gs4.thy "ctr_adm gs4_rls A"
  (fn _ => [
    (rtac gen_ctr_adm_step 1), (rtac wfp_ipsubfml 1), (Clarify_tac 1),
    (etac gs4_rls.elims 1),
    (rewtac lksss_def), (etac extrs_E 1), (etac lksne.elims 1),
    (ALLGOALS Clarsimp_tac),
  (* axiom *)
    (rtac id_ctr_adm_step 1),
    (rtac idrls.intros 1),
    (Force_tac 1),
    (Clarsimp_tac 1), (rtac gs4_rls.lksI 1), (rewtac lksss_def),
    (etac extrs_E 1), (rtac extrs.intros 1), (atac 2), (Clarify_tac 1),
  (* lksil and lksir *)
    (REPEAT_DETERM_N 2 (EVERY [
      (rtac sc_ctr_adm_step 1),
      (rtac refl 1),
      (resolve_tac [lksil_scrls, lksir_scrls] 1), (atac 1),
      (Force_tac 1),
      (subgoal_tac "Ball lksne (subfml_cp_prop ipsubfml)" 1),
      (rtac lks_subfml_cp_prop 2),
      (dresolve_tac [lksne.ilI, lksne.irI] 1), (Clarsimp_tac 1),
      (subgoal_tac "Ball lksss (inv_rl gs4_rls)" 1),
      (rtac inv_rl_gs4_lks 2), (rewtac lksss_def),
      (dresolve_tac [extrs_il_lksne, extrs_ir_lksne] 1), (Force_tac 1),
      (Clarsimp_tac 1), (rtac gs4_rls.lksI 1), (rewtac lksss_def),
      (etac extrs_E 1), (rtac extrs.intros 1), (atac 2), (Clarify_tac 1) ] )),
  (* lkrefl *)
    (rtac sc_ctr_adm_step 1),
    (rtac refl 1),
    (rtac lkrefl_scrls 1), (atac 1),
    (Force_tac 1),
    (clarsimp_tac (cesimps [subfml_cp_prop.simps]) 1),
    (etac lkrefl.elims 1), (Force_tac 1),
    (subgoal_tac "Ball (extrs lkrefl) (inv_rl gs4_rls)" 1),
    (rtac inv_rl_gs4_refl 2), (dtac extrs_lkrefl 1), (Force_tac 1),
    (Clarify_tac 1), (etac extrs_E 1), (dtac gs4_rls.reflI 1), (Force_tac 1),
  (* lkbox *)
    (clarsimp_tac (cesimps [ctr_adm_step_def, ctr_adm_step1.simps,
        ctr_admd_def, ctr_adm_seq_def, extend_def]) 1),
    (subgoal_tac "(As + As <= c) | (As <= flr)" 1),
    (rtac atoms_le_sum_ctr 2), (rtac seq_atomic_ms 2), (atac 2), (atac 2),
    (etac disjE 1), (thin_tac "All ?dtn" 2),
    (* case - contraction formula in context of S4 rule *)
    (subgoal_tac "(p, extend (flr - As) c) : gs4_rls" 2),
    (rtac gs4_rls.boxI 3), (atac 3),
    (rotate_tac -2 2), (dres_inst_tac [("C", "c")] le_pm_assoc 2),
    (asm_full_simp_tac (esimps [extend_def]) 2),
    (etac drs.derI' 2), (Clarsimp_tac 2),
    (* case - contraction formulae "As" both in conclusion of S4 rule *)
```

```
    (* show As can only be in antecedent, and hence As = Box ba *)
    (etac lkbox.elims 1), (Clarsimp_tac 1),
    (thin_tac "?x <= ?y" 1),
    (dtac ms_of_single 1), (etac disjE 1), (Clarsimp_tac 2),
    (dtac le_size_le 2), (Force_tac 2),
    (clarsimp_tac (cesimps [seq_plus_def]) 1),
    (dtac le_mset_map 1), (Clarify_tac 1),
    (dtac (sym RS mset_map_split) 1), (Clarify_tac 1),
    (REPEAT (dtac mset_map_is_single 1)),
    (clarsimp_tac (cesimps [Box_eq]) 1),
    (etac impE 1), (dtac mset_map_mono 1) ,
    (clarsimp_tac (cesimps [mset_map_union]) 1),
    (rtac xt6 1), (atac 2), (Simp_tac 1),
    (* break down IH on formula size for ba *)
    (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
    (eres_inst_tac [("x", "{#Ba#} |- 0")] allE 1), (etac impE 1),
    (Simp_tac 1), (etac thin_rl 1), (etac allE 1), (mp_tac 1),
    (* apply IH on size *)
    (ftac sum_leD1 1),
    (etac impE 1), (Clarsimp_tac 1), (rtac xt3 1),
    (rtac (le_pm_assoc RS sym) 1),
    (rtac (single_le RS iffD2) 1), (rtac mset_mapI 1),
    (rtac (single_le RS iffD1) 1), (atac 1),
    (rtac xt6 1), (atac 2), (Simp_tac 1),
    (* apply IH on premises (depth) *)
    (thin_tac "?x : ?s" 1), (Full_simp_tac 1),
    (rtac drs.derI' 1),
    (res_inst_tac [("flr1", "flr")] (gs4_rls.boxI RS mem_same) 1),
    (res_inst_tac [("gamma", "gamma - {#Ba#}")] lkbox.I 1),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (rtac conjI 1), (rtac refl 1),
    (simp_tac (esimps ([seq_eq_iff, antec_sum, succ_sum] @ add_ac)) 1),
    (rtac conjI 1), (rtac refl 2),
    (rtac trans 1), (rtac le_pm_assoc 2),
    (rtac (single_le RS iffD2) 2), (rtac mset_mapI 2),
    (rtac (single_le RS iffD1) 2), (atac 2), (Simp_tac 1),
    (rtac trans 1), (rtac mset_map_diff_le 1),
    (atac 1), (Simp_tac 1),
    (Simp_tac 1), (etac mem_same 1),
    (simp_tac (esimps (add_ac)) 1),
    (rtac trans 1), (rtac mm_comm 1),
    (rtac trans 1), (rtac th5 1), (atac 1),
    (rtac trans 1), (rtac (le_pm_assoc RS sym) 1),
    (rtac xt6 1), (res_inst_tac [("A", "{#Ba#}")] mset_map_mono 1),
    (atac 1), (Simp_tac 1), (Simp_tac 1),
    (rtac trans 1), (rtac (mset_map_diff_le RS sym) 2),
    (Simp_tac 1), (atac 1)
  ]) ;


(* Cut-elimination *)
val _ = qed_goal "gs4_cas" gs4.thy
  "(Xl |- mins A Yl, mins A Xr |- Yr) : cas gs4_rls A"
  (fn _ => [ (clarify_tac (cis [cas.caI] addSDs [derrec_valid]) 1),
    (rtac (casdtD RS mem_same) 1), (Fast_tac 4), (TRYALL atac),
    (REPEAT (etac thin_rl 1)),
    (rtac (wfp_ipsubfml RS sumhs2_tr_lem_rls) 1), (Clarify_tac 1),
    (rtac sumhs2_tr_casdt_valid 1), (Clarify_tac 1),
    (case_tac "dtaa" 1), (case_tac "dtb" 1), (Safe_tac),
    (Full_simp_tac 3), (Full_simp_tac 2),
    (ftac validD_rls 1), (chop_last (ftac validD_rls 1)),
    (etac thin_rl 1), (etac thin_rl 1), (Full_simp_tac 1),
```

```
  (safe_tac (ces [gs4_rls.elims])),
(* lks rules on both sides *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (rtac gs2sr_alle 1), (rtac gs4_wk_adm 1),
  (full_simp_tac (esimps [lksss_def]) 5),
  (full_simp_tac (esimps [lksss_def]) 5),
  (rtac c8sg_ger 1), (rtac gs4_lksne_c8sg 1),
  (rtac lks_iscrls 1), (rtac xt6 1), (rtac lksne_idrls 2),
  (Clarify_tac 1), (etac (lks_simpI RS gs4_rls.lksI) 1), (Clarify_tac 1),
  (clarsimp_tac (ces [gs4_rls.lksI], esimps [lksss_extrs RS sym]) 1),
(* lks on left, lkrefl on right *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (etac lksss.elims 1), (Clarify_tac 1),
  (case_tac "A :# succ bb" 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps) 2), (atac 3), (atac 3),
  (etac gs4_extrs_lksne 2),
  (etac lksne.elims 1), (* A in succedent of lksne rule *)
  (* lksil succedent is empty *)
  (force_tac (ces [lksil.elims], esimps []) 2),
  (* axiom case *)
  (rtac (gs4_wk_adm RS gs2_axls) 1),
  (Clarify_tac 2), (atac 2), (rtac gs4_id 1),
  (* lksir does not contain box *)
  (case_tac "A :# antec ba" 1),
  (rtac (gs4_wk_adm RS rcg_gen_steps) 2), (atac 3), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extrs.defs]) 2),
  (etac lkrefl.elims 1),
  (force_tac (ces [lksir.elims], esimps (in_single_count :: fc_dist')) 1),
(* lks on left, lkbox on right *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (etac lksss.elims 1), (Clarify_tac 1),
  (case_tac "A :# succ ba" 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps) 2), (atac 3), (atac 3),
  (etac gs4_extrs_lksne 2), (etac lksne.elims 1),
  (force_tac (ces [lksil.elims], esimps []) 2),
  (rtac gs2_axls 1), (rtac gs4_wk_adm 1),
  (Clarify_tac 2), (atac 2), (rtac gs4_id 1),
  (case_tac "A :# antec c" 1), (* S4 rule *)
  (rtac (gs4_wk_adm RS rcg_gen_steps_extcs) 2), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extcs.defs]) 2),
  (etac lkbox.elims 1), (Clarsimp_tac 1), (etac mset_mapE 1),
  (force_tac (ces [lksir.elims], esimps (in_single_count :: fc_dist')) 1),
(* lks on right, lkrefl on left *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (etac lksss.elims 1), (Clarify_tac 1),
  (case_tac "A :# antec bb" 1),
  (rtac (gs4_wk_adm RS rcg_gen_steps) 2), (atac 3), (atac 3),
  (etac gs4_extrs_lksne 2), (etac lksne.elims 1),
  (force_tac (ces [lksir.elims], esimps []) 3),
  (rtac (gs4_wk_adm RS gs2_axrs) 1),
  (Clarify_tac 2), (atac 2), (rtac gs4_id 1),
  (case_tac "A :# succ b" 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps) 2), (atac 3), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extrs.defs]) 2),
  (etac lkrefl.elims 1),
  (force_tac (ces [lksil.elims], esimps (in_single_count :: fc_dist')) 1),
(* lkrefl both sides *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps) 1), (atac 3),
  (etac lkrefl.elims 2), (Clarsimp_tac 2),
  (rtac xt6 1), (etac extrs_lkrefl 2),
  (force_tac (claset (), esimps [gs4_rls.defs, extrs.defs]) 1),
```

```
(* lkrefl on left, lkbox on right *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (case_tac "A :# succ b" 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps) 2), (atac 3), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extrs.defs]) 2),
  (etac lkrefl.elims 1), (Clarsimp_tac 1),
(* lkbox on left, lks on right *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (etac lksss.elims 1), (Clarify_tac 1),
  (case_tac "A :# antec ba" 1),
  (rtac (gs4_wk_adm RS rcg_gen_steps) 2), (atac 3), (atac 3),
  (etac gs4_extrs_lksne 2), (etac lksne.elims 1),
  (force_tac (ces [lksir.elims], esimps []) 3),
  (rtac gs2_axrs 1), (rtac gs4_wk_adm 1),
  (Clarify_tac 2), (atac 2), (rtac gs4_id 1),
  (case_tac "A :# succ c" 1), (* S4 rule *)
  (rtac (gs4_wk_adm RS lcg_gen_steps_extcs) 2), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extcs.defs]) 2),
  (etac lkbox.elims 1),
  (force_tac (ces [lksil.elims], esimps (in_single_count :: fc_dist')) 1),
(* lkbox on left, lkrefl on right *)
  (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
  (case_tac "A :# succ c" 1),
  (rtac (gs4_wk_adm RS lcg_gen_steps_extcs) 2), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extcs.defs]) 2),
  (case_tac "A :# antec b" 1),
  (rtac (gs4_wk_adm RS rcg_gen_steps) 2), (atac 3), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extrs.defs]) 2),
  (* cut formula principal in both rules *)
  (etac lkbox.elims 1), (etac lkrefl.elims 1),
  (clarsimp_tac (cesimps
    [prop2_def, gen_step2sr_def', in_single_count, Box_eq]) 1),
  (REPEAT (thin_tac "conclDT ?z = ?s" 1)),
  (thin_tac "?s : car ?r ?f" 1), (* use cut on S4 conc and refl prem *)
  (thin_tac "?s : derrec ?r ?p" 1), (thin_tac "?s : derrec ?r ?p" 1),
  (REPEAT (chop_tac 1 (thin_tac "?s : derrec ?r ?p" 1))),
  (case_tac "flr" 1), (case_tac "flra" 1),
  (asm_full_simp_tac (esimps [car_eq, extend_def]) 1),
  (* cut on subformula *)
  (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
  (datac bspec 1 1), (chop_tac 1 (datac bspec 1 1)),
  (asm_full_simp_tac (esimps [car_eq]) 1),
  (* use contraction on duplicate Box gamma *)
  (thin_tac "?s : derrec ?r ?p" 1), (thin_tac "?s : derrec ?r ?p" 1),
  (Clarify_tac 1),
  (subgoal_tac "(gamma + mset_map Box gamma + multiset1 + multiset1a \
    \ |- multiset2 + multiset2a) : derrec gs4_rls {}" 1),
  (thin_tac "?s : derrec ?r ?p" 1),
  (rtac (double_eqv RS iffD1) 2), (rtac gs4_wk_adm 3),
  (Clarify_tac 2), (rtac gs4_ctr_adm 2),
  (rtac wk_admD_alt 2), (rtac gs4_wk_adm 2), (atac 2),
  (etac thin_rl 2), (Clarsimp_tac 2),
  (rtac ms_leI 2), (Force_tac 2),
  (* apply lkrefl (multiple times) *)
  (rtac (drs.dlNil RSN (2, drs.dlCons) RSN (2, derl_derrec_trans)) 1),
  (atac 2), (rtac drl_mono 1),
  (rtac (mk_mono gs4_rls_lkrefl) 2),
  (res_inst_tac [("gamma1", "gamma"), ("X1", "multiset1 + multiset1a")]
    (multi_refl RS mem_same) 1),
  (Simp_tac 1), (Safe_tac),
  (TRYALL (rtac refl)),
  (simp_tac spsimps 1), (simp_tac spsimps 1),
```

```
(* lkbox both sides *) (* level 197 *)
  (case_tac "A :# succ c" 1),
  (rtac gs2_car_sumhs_tr 2), (Simp_tac 2),
  (rtac (gs4_wk_adm RS lcg_gen_steps_extcs) 2), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extcs.defs]) 2),
  (case_tac "A :# antec ca" 1),
  (rtac gs2_car_sumhs_tr 2), (Simp_tac 2),
  (rtac (gs4_wk_adm RS rcg_gen_steps_extcs) 2), (atac 3),
  (force_tac (claset (), esimps [gs4_rls.defs, extcs.defs]) 2),
  (* case where A is principal in both sides *)
  (* want to cut on new conclusion on left, without context *)
  (rtac sumhs2_tr_casdt_valid 1), (Clarify_tac 1),
  (clarsimp_tac (cesimps [sumh_step2_tr_eq, prop2_def, casdt_eq]) 1),
  (subgoal_tac "valid gs4_rls (Der c list)" 1),
  (rtac validRule 2), (atac 2),
  (dres_inst_tac [("flr", "0")] gs4_rls.boxI 2), (Clarsimp_tac 2),
  (* get premises of both sides as valid dertrees *)
  (REPEAT (etac lkbox.elims 1)), (Clarify_tac 1),
  (subgoal_tac "valid gs4_rls z" 1),
  (dtac validUpD 2), (atac 3), (Clarsimp_tac 2),
  (subgoal_tac "valid gs4_rls za" 1),
  (chop_tac 1 (dtac validUpD 2)), (atac 3), (Clarsimp_tac 2),
  (* now cut based on level *)
  (eres_inst_tac [("xa", "Der (mset_map Box gamma |- {#Box Aa#}) [z]"),
      ("x", "za")] allE2 1),
  (clarsimp_tac (cesimps [in_single_count, car_eq]) 1),
  (* show that there exists a dertree for this new sequent *)
  (dtac derrec_valid 1), (etac exE 1), (etac conjE 1),
  (* cut on formula size *)
  (eres_inst_tac [("x", "Aa")] allE 1),
  (etac impE 1), (rtac Box_ipsubfml 1),
  (eres_inst_tac [("xa", "z"), ("x", "dt")] allE2 1),
  (clarsimp_tac (cesimps [car_eq]) 1),
  (REPEAT (thin_tac "valid ?rls ?dt" 1)),
  (REPEAT (thin_tac "conclDT ?dt = ?s" 1)),
  (* contract extra copy of Box gamma *)
  (subgoal_tac "Aa :# gammaa" 1),
  (clarsimp_tac (ces [mset_mapE], esimps []) 2),
  (subgoal_tac "(gamma + mset_map Box gamma + \
      \ (gammaa - {#Aa#}) + (mset_map Box gammaa  - {#Box Aa#}) |- \
      \ {#Ab#}) : derrec gs4_rls {}" 1),
  (rtac ctr_wk_le_double 2), (atac 4),
  (rtac gs4_wk_adm 3), (Clarify_tac 2),
  (rtac gs4_ctr_adm 2),
  (simp_tac spsimps 2),
  (REPEAT (dtac (single_le RS iffD2) 2)),
  (asm_simp_tac (esimps [le_pm_assoc RS sym]) 2),
  (stac (le_pm_assoc RS sym) 2),
  (rtac xt6 2), (rtac le_plus1 2), (atac 2),
  (asm_simp_tac (esimps [le_pm_assocc RS sym]) 2),
  (simp_tac spsimps 2),
  (* apply S4 rule *)
  (thin_tac "?s : ?d" 1),
  (case_tac "flr" 1), (case_tac "flra" 1),
  (asm_simp_tac (esimps [car_eq]) 1),
  (etac (reop rev drs.derIsingle) 1),
  (res_inst_tac
    [("flr1", "multiset1 + multiset1a |- multiset2 + multiset2a"),
    ("gamma2", "gamma + (gammaa - {#Aa#})")]
    (lkbox.I RS gs4_rls.boxI RS mem_same) 1),
  (simp_tac spsimps 1), (Safe_tac), (rtac refl 2),
  (simp_tac (esimps [mset_map_union, mset_map_diff_inj]) 1),
```

```
    (simp_tac (spsimps addsimps
      [mset_map_union, mset_map_diff_inj, inj_Box]) 1),
    (etac (transfer gs4.thy single_le RS iffD2 RS le_pm_assoc) 1),
    (rtac refl 1)
  ]) ;
```

A.1.2 Core proofs for S4.3

Note that some cases, which are exact analogues to those for S4, in the following proofs have been omitted for the sake of space. These are indicated by dots surrounding a comment, e.g.

```
.
.
(* similar to some case *)
.
.
```

The main proofs:

```
(* Prove Weakening-admissibility for antec and succ sides separately *)
(* Left side *)
val _ = qed_goalw "gs43_wk_adm_santec" gs43.thy [wk_adm_single_antec_def]
  "wk_adm_single_antec gs43_rls"
  (fn _ => [ (Clarify_tac 1),
    (res_inst_tac [("P","%A x. x + ({#A#} |- {#}) : derrec gs43_rls {}")]
      gen_step_lem 1),
    (rtac wfp_ipsubfml 1), (atac 2),
    (clarsimp_tac (cesimps [gen_step.simps]) 1),
    (etac gs43_rls.elims 1),
  (* lksss *)
    (etac lksss.elims 1),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (rtac drs.derI'' 1), (rtac gs43_rls.lksI 1),
    (dres_inst_tac [("flr","flr +  ({#A#} |- 0)")] lksss.extI 1),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (chop_last (etac mem_same 1)),
    (Clarify_tac 1), (rtac conjI 1), (rtac refl 1),
    (simp_tac spsimps 1),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (datac bspec 1 1), (dtac conjunct2 1),
    (chop_last (etac mem_same 1)), (simp_tac spsimps 1),
  (* lkrefl *)
    (etac lkrefl.elims 1), (clarsimp_tac (cesimps [extend_def]) 1),
    (res_inst_tac [("p","({#Aa#} + {#Box Aa#} |- 0) + flr + ({#A#} |- 0)")]
      drs.derIsingle 1), (atac 2),
    (res_inst_tac [("A2","Aa"), ("flr1", "flr + ({#A#} |- 0)")]
      (lkrefl.I RS gs43_rls.reflI RS mem_same) 1),
    (simp_tac spsimps 1),
  (* s43box *)
    (Clarify_tac 1), (case_tac "EX Aw. A = Box Aw" 1),
  (* boxed case *)
    (Clarify_tac 1), (subgoal_tac
      "extend (nboxseq flr) (c + ({#Box Aw#} |- 0)) : derrec gs43_rls {}" 1),
    (chop_last (etac mem_same 1)),
    (simp_tac spsimps 1),
    (etac s43box.elims 1), (Clarsimp_tac 1),
    (rtac drs.derI'' 1), (rtac gs43_rls.boxI 1),
    (rtac mem_same 1), (rtac s43box.I 1), (Clarsimp_tac 1),
    (rtac conjI 1), (rtac conjI 2), (rtac refl 3),
```

```
    (rtac trans 2), (rtac mset_map_union 2),
    (rtac (add_left_canc_eq RS iffD2) 2),
    (rtac mset_map_single 2), (rtac refl 1),
    (clarsimp_tac (cesimps [nprems_def]) 1),
    (datac bspec 1 1), (dtac conjunct2 1),
    (clarsimp_tac (cesimps [ithprem_def, mset_map_union]) 1),
    (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
    (dtac bspec 1), (chop_last (atac 1)),
    (chop_last (etac mem_same 1)),
    (simp_tac spsimps 1),
  (* non-boxed case *)
    (res_inst_tac [("ps","p")] drs.derI'' 1),
    (Clarify_tac 2),
    (datac bspec 1 2), (dtac conjunct1 2), (atac 2),
    (dres_inst_tac [("flr","flr + ({#Abs_nboxfml A#} |- 0)")]
      gs43_rls.boxI 1),
    (REPEAT_DETERM_N 4 (etac thin_rl 1)),
    (etac mem_same 1),
    (simp_tac (spsimps addsimps
      [nboxseq_def, antec_sum, succ_sum, mset_map_union]) 1),
    (rtac Abs_nboxfml_inverse 1),
    (clarsimp_tac (cesimps [nboxfml_def, Box]) 1)
  ]) ;

val _ = bind_thm ("gs43_wk_adm_antec", gs43_wk_adm_santec RS wk_adm_santec);

(* Right side *)
val _ = qed_goalw "gs43_wk_adm_ssucc" gs43.thy [wk_adm_single_succ_def]
  "wk_adm_single_succ gs43_rls"
  (fn _ => [
  .
  .
  (** same as left side up until the s43box case *)
  .
  .
  (* s43box *)
    (Clarify_tac 1), (case_tac "EX Aw. A = Box Aw" 1),
  (* non-boxed case *)
    (res_inst_tac [("ps","p")] drs.derI'' 2),
    (Clarify_tac 3),
    (datac bspec 1 3), (dtac conjunct1 3), (atac 3),
    (dres_inst_tac [("flr","flr + (0 |- {#Abs_nboxfml A#})")]
      gs43_rls.boxI 2),
    (REPEAT_DETERM_N 4 (etac thin_rl 2)),
    (etac mem_same 2),
    (simp_tac (spsimps addsimps
      [nboxseq_def, antec_sum, succ_sum, mset_map_union]) 2),
    (rtac Abs_nboxfml_inverse 2),
    (clarsimp_tac (cesimps [nboxfml_def, Box]) 2),
  (* boxed case *)
    (Clarify_tac 1), (subgoal_tac
      "extend (nboxseq flr) (c + (0 |- {#Box Aw#})) : derrec gs43_rls {}" 1),
    (chop_last (etac mem_same 1)),
    (simp_tac spsimps 1),
    (etac thin_rl 1),
    (* first show new premise for Box Aw is derivable:
       (gamma + mset_map Box gamma |- {#Aw#} + mset_map Box (ms_of_list As))
       start from original rule, but remove context *)
    (subgoal_tac "c : derrec gs43_rls {}" 1),
    (res_inst_tac [("ps","p")] drs.derI'' 2),
    (dres_inst_tac [("flr", "0")] gs43_rls.boxI 2),
    (REPEAT_DETERM_N 3 (etac thin_rl 2)),
```

```
    (full_simp_tac (spsimps addsimps [nboxseq_def, seq_zero_def]) 2),
    (Clarify_tac 2), (datac bspec 1 2), (dtac conjunct1 2), (atac 2),
    (etac s43box.elims 1), (Clarsimp_tac 1),
    (thin_tac "?x : ?derivs" 1),
    (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
    (chop_last (datac bspec 1 1)),
    (thin_tac "?x : ?derivs" 1),
    (subgoal_tac "wk_adm_antec gs43_rls" 1),
    (rtac gs43_wk_adm_antec 2),
    (full_simp_tac (spsimps addsimps [wk_adm_antec_def]) 1),
    (etac allE 1), (eatac impE 1 1),
    (eres_inst_tac [("x","(gamma |- 0)")] allE 1),
    (thin_tac "?x : ?derivs" 1), (full_simp_tac spsimps 1),
    (* apply S4.3 rule *)
    (rtac mem_same 1), (rtac add_commute 2), (rtac drs.derI'' 1),
    (rtac (simplify (esimps [extend_def]) gs43_rls.boxI) 1),
    (res_inst_tac [("a","(nprems gamma (Aw # As), \
      \ (mset_map Box gamma |- mset_map Box (ms_of_list (Aw # As))))")]
      mem_same 1),
    (rtac s43box.I 1), (Clarify_tac 1),
    (rtac conjI 1), (rtac refl 1),
    (simp_tac (spsimps addsimps [mins_def, mset_map_union]) 1),
    (Clarify_tac 1),
    (clarsimp_tac (cesimps [nprems_def, ithprem_def]) 1),
    (etac disjE 1),
    (* extra premise *)
    (Clarify_tac 1), (etac mem_same 1), (Force_tac 1),
    (clarsimp_tac (cesimps [ithprem_def]) 1), (Safe_tac),
    (etac mem_same 1), (Force_tac 1),
    (* Weaken in Box Aw to old premises *)
    (datac bspec 1 1), (dtac conjunct2 1),
    (simp_tac (spsimps addsimps [mins_def, mset_map_union]) 1),
    (chop_last (etac mem_same 1)),
    (simp_tac spsimps 1)
  ]) ;

val _ = bind_thm ("gs43_wk_adm_succ", gs43_wk_adm_ssucc RS wk_adm_ssucc);

val _ = bind_thm ("gs43_wk_adm",
        [gs43_wk_adm_antec, gs43_wk_adm_succ] MRS wk_adm_sides);

(* invertibility of most rules *)
(* lkrefl invertible as principal formula copied to premise *)
val _ = qed_goal "inv_rl_gs43_refl" gs43.thy
  "Ball (extrs lkrefl) (inv_rl gs43_rls)"
  (fn _ => [ (clarsimp_tac (cesimps [inv_rl.simps, drs.all]) 1),
    (etac (gs43_wk_adm RS wk_admD_alt) 1),
    (clarsimp_tac (ces [extrs_E, lkrefl.elims], esimps [extend_def]) 1) ]) ;

val _ = qed_goal "inv_rl_gs43_lks" gs43.thy "Ball lksss (inv_rl gs43_rls)"
  (fn _ => [
  .
  .
  (** same as the corresponding proof for S4 (replacing gs4 with gs43),
      up until the s43box rule *)
  .
  .
  (* rule in s43box *)
    (clarsimp_tac (ces [extrs.elims], esimps [inv_step.simps, invs_of_def]) 1),
    (* use conclusion of modal rule, without context *)
    (subgoal_tac "c : derrec gs43_rls {}" 1),
    (rtac drs.derI' 2), (atac 3),
```

```
    (dres_inst_tac [("flr", "0")] gs43_rls.boxI 2),
    (chop_last (etac mem_same 2)),
    (simp_tac (spsimps addsimps [nboxseq_def, seq_zero_def]) 2),
    (etac s43box.elims 1), (etac lksne.elims 1),
    (Auto_tac),
    (* show that "c" must be a subsequent of premise xa, then use wk_adm *)
    (REPEAT (EVERY [
      (forward_tac [lksil_atomic, lksir_atomic] 1),
      (dtac (atomic_le_sum RS iffD1) 1),
      (etac allE2 1), (etac impE 1), (rtac xt3 1),
      (full_simp_tac (esimps [extend_def]) 2),
      (rtac le_plus1 2),
      (clarsimp_tac (cesimps [extend_def]) 1),
      (etac sym 1), (etac disjE 1),
      (eresolve_tac [UnI1 RS lks_not_Box', UnI2 RS lks_not_Box'] 1), atac 1,
      (clarsimp_tac (cesimps [extend_def]) 1),
      (fatac (sum_seq_le_lem RS iffD1) 1 1),
      (REPEAT_DETERM_N 4 (etac thin_rl 1)),
      (thin_tac "?a <= nboxseq ?flr" 1),
      (dtac le_add_diff_ex 1), (etac exE 1),
      (subgoal_tac "wk_adm gs43_rls" 1),
      (rtac gs43_wk_adm 2), (full_simp_tac (esimps [wk_adm_def]) 1),
      (etac allE 1), (eatac impE 1 1),
      (eres_inst_tac [("x","xa + D")] allE 1),
      (chop_last (etac mem_same 1)),
      (Clarify_tac 1), (simp_tac spsimps 1)] ))
  ]) ;

(* contraction admissibility *)
(* first need to prove following lemma *)
val _ = qed_goal_spec_mp "gs43_refl" gs43.thy
  "seq : derrec gs43_rls {} --> \
    \ (ALL X Y. seq = extend (X |- Y) (0 |- {#Box A#}) --> \
    \ extend (X |- Y) (0 |- {#A#}) : derrec gs43_rls {})"
  (fn _ => [ (rtac impI 1), (etac drs.inductr 1),
    (ALLGOALS (Clarify_tac)),
    (etac gs43_rls.elims 1),
  (* lkrefl, use IH on prems then re-apply rule *)
    (etac lkrefl.elims 2), (case_tac "flr" 2),
    (clarsimp_tac (cesimps [extend_def]) 2),
    (rtac drs.derIsingle 2), (chop_last (atac 3)),
    (subgoal_tac
        "([{#Aa#} + {#Box Aa#} |- {#}], {#Box Aa#} |- {#}) : lkrefl" 2),
    (rtac lkrefl.I 3),
    (dres_inst_tac [("flr", "(multiset1 |- {#A#} + Y)")] gs43_rls.reflI 2),
    (Clarsimp_tac 2),
    (etac lksss.elims 1), (etac lksne.elims 1),
  (* axiom *)
    (Clarify_tac 1), (case_tac "flr" 1),
    (clarsimp_tac (cesimps [extend_def]) 1),
    (case_tac "Aa = Box A" 1),
    (* case: create new axiom *)
    (subgoal_tac "extend ({#Box A#} + multiset1 |- Y) \
        \ ({#A#} |- {#A#}) : derrec gs43_rls {}" 1),
    (rtac drs.derI'' 2), (rtac (simplify (simpset())
        (lksne.axiom RS lksss.extI RS gs43_rls.lksI)) 2),
    (Clarsimp_tac 2), (clarsimp_tac (cesimps [extend_def]) 1),
    (rtac drs.derIsingle 1), (atac 2), (subgoal_tac
        "([{#A#} + {#Box A#} |- {#}], {#Box A#} |- {#}) : lkrefl" 1),
    (rtac lkrefl.I 2),
    (dres_inst_tac [("flr", "(multiset1 |- {#A#} + multiset2)")]
        gs43_rls.reflI 1),
```

```
  (full_simp_tac spsimps 1),
  (* case: use old axiom *)
  (subgoal_tac "Aa :# Y" 1),
  (dtac (sym RS le_plus_eq1) 2),
  (asm_full_simp_tac (esimps [in_single_count, single_le]) 2),
  (Clarify_tac 2),
  (res_inst_tac  [("A", "Aa")] id_admD 1), (rtac gs43_id_adm 1),
  (Simp_tac 1), (Clarsimp_tac 1),
(* lksil and lksir do not affect Box A*)
  (Clarsimp_tac 1), (case_tac "Box A :# succ bb" 1),
  (force_tac (ces [lksil.elims], simpset()) 1),
  (Clarsimp_tac 2), (case_tac "Box A :# succ bb" 2),
  (case_tac "bb" 2), (Clarsimp_tac 2),
  (eatac (UnI2 RS lks_no_Box_succ) 1 2),
  (* case of parametric rule application *) (*level 46*)
  (REPEAT_DETERM_N 2 (EVERY [
      (case_tac "bb" 1), (case_tac "flr" 1),
      (Clarsimp_tac 1),
      (chop_last (ftac (sym RS xt3) 1) ), (rtac le_plus1 1),
      (dtac (atomic_single RS (atomic_le_sum RS iffD1)
          RS spec RS spec RS mp) 1),
      (etac disjE 1), (clarsimp_tac (cesimps [single_le]) 1),
      (dtac le_add_diff_ex 1), (clarsimp_tac (claset (), spsimps) 1),
      (rtac drs.derI'' 1),
      (rtac (lksss_extI2 RS gs43_rls.lksI RS mem_same) 1),
      (eresolve_tac [lksne.ilI, lksne.irI] 1),
      (Clarsimp_tac 1), (Fast_tac 1),
      (Clarsimp_tac 1), (case_tac "x" 1), (datac bspec 1 1),
      (etac allE2 1), (etac impE 1), (chop_last (etac mem_same 2)),
      (clarsimp_tac (cesimps [add_assoc RS sym]) 2), (Fast_tac 2),
      (asm_simp_tac spsimps 1) ])),
(* S4.3 rule, use premise with A unboxed *)
  (chop_tac 1 (etac thin_rl 1)),
  (etac s43box.elims 1), (case_tac "flr" 1),
  (clarsimp_tac (cesimps [extend_def, nboxseq_def]) 1),
  (ftac le_plus_eq1 1),
  (dtac (atomic_single RS (atomic_le_sum RS iffD1)
      RS spec RS spec RS mp) 1),
  (full_simp_tac (esimps [single_le]) 1),
  (etac disjE 1),
  (dtac (sum_diffc RS sym) 1),
  (clarsimp_tac (cesimps [in_ms_map_list, nprems_def]) 1),
  (subgoal_tac "(gamma + mset_map Box gamma |- {#A#} + \
      \ mset_map Box (ms_of_list (remove1 A As))) : derrec gs43_rls {}" 1),
  (* use weakening admissibility *)
  (subgoal_tac "wk_adm gs43_rls" 1), (rtac gs43_wk_adm 2),
  (full_simp_tac (esimps [wk_adm_def]) 1),
  (etac allE 1), (eatac impE 1 1),
  (eres_inst_tac [("x","(mset_map Rep_nboxfml multiset1 |- \
      \ mset_map Rep_nboxfml multiset2)")] allE 1),
  (etac thin_rl 1), (thin_tac "?x : derrec gs43_rls {}" 1),
  (* apply refl rule multiple times *)
  (rtac (drs.dlNil RSN (2, drs.dlCons) RSN (2, derl_derrec_trans)) 1),
  (atac 2), (rtac drl_mono 1),
  (rtac (mk_mono gs43_rls_lkrefl) 2),
  (res_inst_tac [("gamma1", "gamma"),
      ("X1", "mset_map Rep_nboxfml multiset1")]
      (multi_refl RS mem_same) 1),
  (Simp_tac 1), (rtac conjI 1), (rtac refl 1),
(* clean up subgoals *)
  (simp_tac spsimps 1),
  (rtac trans 1), (rtac le_pm_assoc 2),
```

```
    (simp_tac spsimps 1),
    (simp_tac (esimps [in_ms_map_list, single_le]) 2),
    (etac imageI 2),
    (simp_tac (esimps [mset_remove1,
      transfer gs43.thy inj_Box RS mset_map_diff_inj]) 1),
    (clarsimp_tac (cesimps [drs.all]) 1),
    (dtac subsetD 1), (etac imageI 1),
    (clarsimp_tac (cesimps [ithprem_def]) 1),
    (etac mem_same 1), (Simp_tac 1),
    (dtac in_mset_map 1), (Clarify_tac 1),
    (dtac ([Rep_nboxfml, sym] MRS mem_same) 1),
    (force_tac (cesimps [nboxfml_def, Box]) 1)
  ]) ;

(* The actual Contraction proof *)
val _ = qed_goal "gs43_ctr_adm" gs43.thy "ctr_adm gs43_rls A"
  (fn _ => [
  .
  .
  (** same as the corresponding proof for S4 (replacing gs4 with gs43),
      up until the s43box rule *)
  .
  .
  (* s43box *)
    (clarsimp_tac (cesimps [ctr_adm_step_def, ctr_adm_step1.simps,
        ctr_admd_def, ctr_adm_seq_def, extend_def]) 1),
    (subgoal_tac "(As + As <= c) | (As <= nboxseq flr)" 1),
    (rtac atoms_le_sum_ctr 2), (rtac seq_atomic_ms 2), (atac 2), (atac 2),
    (etac disjE 1), (thin_tac "All ?dtn" 2),
    (* case - contraction formula in context of S4.3 rule *)
    (rtac drs.derI'' 2), (Clarify_tac 3),
    (datac bspec 1 3), (datac conjunct1 1 3),
    (dres_inst_tac [("flr","flr - seqnbox As")] gs43_rls.boxI 2),
    (rtac mem_same 2), (atac 2),
    (simp_tac (spsimps addsimps [nboxseq_diff]) 2),
    (stac Abs_nboxseq_inv 2), (etac le_pm_assoc 3),
    (case_tac "As" 2),
    (clarsimp_tac (cesimps [nboxseq_def]) 2),
    (subgoal_tac "atomic {#dt#}" 2), (rtac atomic_single 3),
    (dtac (atomic_le_sum RS iffD1) 2),
    (eres_inst_tac [("x", "multiset1"), ("xa", "multiset2")] allE2 2),
    (etac impE 2), (rtac order_eq_refl 2), (simp_tac spsimps 2),
    (REPEAT_DETERM_N 4 (etac thin_rl 2)), (etac disjE 2),
    (chop_tac 1 (dtac xt6 2)), (atac 2), (dtac xt6 3), (atac 3),
    (REPEAT (EVERY [(clarsimp_tac (cesimps [single_le]) 2),
        (dtac in_mset_map 2), (Clarify_tac 2), (rtac Rep_nboxfml 2)])),
    (* case - contraction formula principal in antecedent of S4.3 rule *)
    (thin_tac "?a <= ?b" 1), (case_tac "As = ({#dt#} |- 0)" 1),
    (* set up for IH on premises *)
    (etac s43box.elims 1), (Clarsimp_tac 1),
    (ftac le_mset_map 1), (Clarify_tac 1),
    (dtac (sym RS mset_map_split) 1), (Clarify_tac 1),
    (REPEAT (dtac mset_map_is_single 1)), (Clarsimp_tac 1),
    (rtac mem_same 1), (rtac le_pm_assocc 2),
    (ftac sum_leD1 2), (full_simp_tac spsimps 2),
    (simp_tac (esimps [seq_minus_def]) 1),
    (* now use S4.3 rule *)
    (res_inst_tac [("ps","nprems (gamma - {#Ba#}) Asa")] drs.derI'' 1),
    (rtac mem_same 1),
    (res_inst_tac [("gamma1","gamma - {#Ba#}"), ("flr","flr"), ("As1", "Asa")]
        (s43box.I RS gs43_rls.boxI) 1), (simp_tac spsimps 1),
    (simp_tac (esimps [transfer gs43.thy inj_Box RS mset_map_diff_inj]) 1),
```

```
(clarsimp_tac (cesimps [nprems_def, ithprem_def]) 1),
(* apply IH on height *)
(datac bspec 1 1), (dtac conjunct2 1), (etac impE 1),
(rtac xt6 1), (rtac le_plus1 1), (atac 1),
(* apply IH on subformula *)
(etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
(eres_inst_tac [("x", "{#Ba#} |- 0")] allE 1),
(chop_last (ftac sum_leD1 1)), (etac impE 1),
(Simp_tac 1), (etac allE 1), (mp_tac 1), (etac impE 1),
(Clarsimp_tac 1), (rtac xt3 1),
(rtac (le_pm_assocc RS sym) 1),
(rtac (single_le RS iffD2) 1), (rtac mset_mapI 1),
(rtac (single_le RS iffD1) 1), (atac 1),
(rtac xt6 1), (atac 2), (Simp_tac 1),
(thin_tac "?x : derrec gs43_rls {}" 1),
(Clarsimp_tac 1), (rtac mem_same 1), (atac 1),
(Simp_tac 1), (rtac trans 1), (rtac th5 1),
(rtac (single_le RS iffD2) 1), (rtac mset_mapI 1),
(rtac (single_le RS iffD1) 1), (atac 1),
(stac mset_map_diff_inj 1), (rtac inj_Box 1),
(simp_tac spsimps 1), (rtac (le_pm_assocc RS sym) 1), (atac 1),
(* case - contraction formula principal in succedent of S4.3 rule *)
(subgoal_tac "As = (0 |- {#dt#})" 1),
(ftac seq_atomic_ms 2), (case_tac "As" 2), (Clarsimp_tac 2),
(dtac (atomic_le_sum RS iffD1) 2),
(eres_inst_tac [("x", "multiset1 |- 0"),
    ("xa", "0 |- multiset2")] allE2 2),
(chop_last (etac impE 2)), (MSSG Simp_tac 2),
(Clarsimp_tac 2), (etac disjE 2),
(REPEAT (clarsimp_tac (cesimps [le0_eq]) 2)),
(* set up for IH on premises *)
(etac s43box.elims 1), (Clarsimp_tac 1),
(ftac le_mset_map 1), (Clarify_tac 1),
(dtac (sym RS mset_map_split) 1), (Clarify_tac 1),
(REPEAT (dtac mset_map_is_single 1)), (Clarsimp_tac 1),
(rtac mem_same 1), (rtac le_pm_assocc 2),
(ftac sum_leD1 2), (full_simp_tac spsimps 2),
(simp_tac (esimps [seq_minus_def]) 1),
(* now use S4.3 rule *)
(res_inst_tac [("ps","nprems gamma (remove1 Ba Asa)")] drs.derI'' 1),
(rtac mem_same 1),
(res_inst_tac [("gamma1","gamma"), ("flr","flr"),
    ("As1", "remove1 Ba Asa")] (s43box.I RS gs43_rls.boxI) 1),
(simp_tac spsimps 1),
(simp_tac (esimps [mset_remove1,
    transfer gs43.thy inj_Box RS mset_map_diff_inj]) 1),
(clarsimp_tac (cesimps [nprems_def, ithprem_def]) 1),
(* two cases, contraction formula both boxed/one unboxed in premise *)
(ftac (set_remove1_subset RS subsetD) 1),
(* deal with unboxed case first, needs lemma *)
(case_tac "x = Ba" 1), (Clarsimp_tac 1),
(datac bspec 1 1), (dtac conjunct1 1),
(subgoal_tac "Box Ba :# mset_map Box (ms_of_list (remove1 Ba Asa))" 1),
(rtac mset_mapI 2), (simp_tac (esimps [mset_remove1]) 2),
(chop_last (dtac (ms_le_def RS defD1) 2)),
(chop_last (etac allE 2)), (etac xt8 2), (Force_tac 2),
(* use lemma *)
(dtac (transfer gs43.thy single_le RS iffD2 RS le_add_diff_ex) 1),
(etac exE 1), (dtac sym 1),
(dres_inst_tac [("A", "Ba"), ("Y", "{#Ba#} + D")] gs43_refl 1),
(MSSG (asm_simp_tac spsimps) 1),
(etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
```

```
    (eres_inst_tac [("x", "0 |- {#Ba#}")] allE 1), (etac impE 1),
    (Force_tac 1), (etac allE 1), mp_tac 1, (etac impE 1), (Force_tac 1),
    (chop_last (etac mem_same 1)),
    (clarsimp_tac (cesimps [mset_remove1]) 1),
    (dtac (sym RS (add_commute RS trans RS sum_diff)) 1),
    (clarsimp_tac (cesimps
        [transfer gs43.thy inj_Box RS mset_map_diff_inj]) 1),
    (* both contraction formula boxed, use IH on height *)
    (datac bspec 1 1), (dtac conjunct2 1), (etac impE 1),
    (simp_tac (esimps [transfer gs43.thy
        inj_Box RS mset_map_diff_inj, mset_remove1]) 1),
    (rtac xt6 1), (rtac le_plus2 1), (rtac sum_le 1), (etac thin_rl 1),
    (subgoal_tac "{#x#} + ({#Ba#} + {#Ba#}) <= (ms_of_list Asa)" 1),
    (chop_tac 1 (dtac le_add_diff_eq 2)), (etac (sym RS xt3) 2),
    (simp_tac spsimps 2),
    (rtac ms_leI 2), (Simp_tac 2),
    (SELECT_GOAL Safe_tac 2),
    (rtac Suc_leI 2), (etac (in_ms_of RS iffD2) 2),
    (chop_last (dtac mset_map_mono 1)),
    (chop_last (etac xt4 1)),
    (simp_tac (esimps [mset_map_union]) 1),
    (etac (gs43_wk_adm RS wk_admD_alt) 1),
    (Simp_tac 1), (rtac xt6 1), (rtac pm_assoc_le 2),
    (simp_tac (spsimps addsimps [transfer gs43.thy
        inj_Box RS mset_map_diff_inj, mm_comm, mset_remove1]) 1)
  ]) ;

(* Cut-elimination *)
val _ = qed_goal "gs43_cas" gs43.thy
  "(Xl |- mins A Yl, mins A Xr |- Yr) : cas gs43_rls A"
  (fn _ => [
  .
  .
  (* initial 7 cases similar to S4 *)
  .
  .
  (* lkbox on left, lkrefl on right *)
    (rtac gs2_car_sumhs_tr 1), (Simp_tac 1),
    (case_tac "A :# succ c" 1),
    (rtac (gs43_wk_adm RS lcg_gen_step_extconc) 2), (atac 3),
    (etac gs43_s43box 2),
    (case_tac "A :# antec b" 1),
    (rtac (gs43_wk_adm RS rcg_gen_steps) 2), (atac 3), (atac 3),
    (force_tac (claset (), esimps [gs43_rls.defs, extrs.defs]) 2),
    (* cut formula principal in both rules *)
    (etac s43box.elims 1), (etac lkrefl.elims 1),
    (clarsimp_tac (cesimps
        [prop2_def, gen_step2sr_def', in_single_count]) 1),
    (REPEAT (thin_tac "?f conclDT ?z = ?s" 1)),
    (REPEAT( thin_tac "?s : derrec ?r ?p" 1)),
    (* use cut on S4.3 conc and refl prem *)
    (case_tac "flr" 1), (case_tac "flra" 1),
    (clarsimp_tac (cesimps [car_eq, extend_def, nboxseq_def]) 1),
    (* cut on subformula, first show premise which unboxes A is derivable *)
    (etac mset_mapE 1), (Clarsimp_tac 1),
    (subgoal_tac "ithprem gamma As x : set (nprems gamma As)" 1),
    (simp_tac (esimps [nprems_def]) 2), (rtac imageI 2),
    (clarsimp_tac (cesimps [in_ms_of]) 2),
    (* now the actual cut *)
    (datac bspec 1 1), (dtac conjunct1 1),
    (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
    (dtac bspec 1), (chop_last (atac 1)),
```

```
  (dtac bspec 1), (atac 1),
  (REPEAT (thin_tac "?x : derrec gs43_rls {}" 1)),
  (thin_tac "?x : set ?s" 1),
  (clarsimp_tac (cesimps [ithprem_def, car_eq]) 1),
  (* contract duplicates of Box gamma, and Box (As - x) *)
  (subgoal_tac "(gamma + mset_map Box gamma + \
      \ mset_map Rep_nboxfml multiset1 + multiset1a |- \
      \ mset_map Box (ms_of_list As) + mset_map Rep_nboxfml multiset2 - \
      \ {#Box x#} + multiset2a) : derrec gs43_rls {}" 1),
  (rtac ctr_wk_le_double 2), (atac 4),
  (rtac gs43_wk_adm 3), (Clarify_tac 2), (rtac gs43_ctr_adm 2),
  (simp_tac spsimps 2), (rtac xt6 2), (rtac le_plus2 2),
  (simp_tac (esimps [mset_remove1,
      transfer gs43.thy inj_Box RS mset_map_diff_inj]) 2),
  (stac (le_pm_assoc RS sym) 2), (rtac le_plus2 3),
  (simp_tac (esimps [single_le]) 2),
  (rtac mset_mapI 2), (atac 2),
  (thin_tac "?x : derrec gs43_rls {}" 1),
  (* apply the refl rule multiple times *)
  (rtac (drs.dlNil RSN (2, drs.dlCons) RSN (2, derl_derrec_trans)) 1),
  (atac 2), (rtac drl_mono 1),
  (rtac (mk_mono gs43_rls_lkrefl) 2),
  (res_inst_tac [("gamma1", "gamma"),
      ("X1", "mset_map Rep_nboxfml multiset1 + multiset1a")]
      (multi_refl RS mem_same) 1),
  (Simp_tac 1), (Safe_tac),
  (TRYALL (rtac refl)),
  (simp_tac spsimps 1), (simp_tac spsimps 1),
(* s43box both sides *)
  (case_tac "A :# succ c" 1),
  (rtac gs2_car_sumhs_tr 2), (Simp_tac 2),
  (rtac (gs43_wk_adm RS lcg_gen_step_extconc) 2), (atac 3),
  (rtac gs43_s43box 2), (atac 2),
  (case_tac "A :# antec ca" 1),
  (rtac gs2_car_sumhs_tr 2), (Simp_tac 2),
  (rtac (gs43_wk_adm RS rcg_gen_step_extconc) 2), (atac 3),
  (rtac gs43_s43box 2), (atac 2),
  (* case where A is principal in both sides *)
  (rtac sumhs2_tr_casdt_valid 1), (Clarify_tac 1),
  (clarsimp_tac (cesimps [sumh_step2_tr_eq, prop2_def, casdt_eq]) 1),
  (* will need to cut on new conclusion on left, without context *)
  (subgoal_tac "valid gs43_rls (Der c list)" 1),
  (rtac validRule 2), (atac 2),
  (rtac gs43_s43box 2), (atac 2),
  (* also need another new conclusion on right, without context *)
  (subgoal_tac "valid gs43_rls (Der ca lista)" 1),
  (rtac validRule 2), (atac 2),
  (rtac gs43_s43box 2), (atac 2),
  (* show premises are valid dertrees *)
  (subgoal_tac "ALL pr : set lista. valid gs43_rls pr" 1),
  (Clarify_tac 2), (eatac validUpD 1 2),
  (subgoal_tac "ALL pl : set list. valid gs43_rls pl" 1),
  (Clarify_tac 2), (eatac validUpD 1 2),
  (* cut new left conclusion with all right premises, using IH on level *)
  (subgoal_tac "ALL pr : set lista. \
      \ (conclDT (Der c list), conclDT pr) : car gs43_rls A" 1),
  (Clarify_tac 2),
  (eres_inst_tac [("xa", "Der c list"), ("x", "pr")] allE2 2),
  (etac impE 2), (simp_tac (esimps [less_Suc_eq_le]) 2),
  (etac height_s 2), (Clarsimp_tac 2),
  (* cut new right conclusion with all left premises, using IH on level *)
  (subgoal_tac "ALL pl : set list. \
```

```
    \ (conclDT pl, conclDT (Der ca lista)) : car gs43_rls A" 1),
(Clarify_tac 2),
(eres_inst_tac [("xa", "pl"), ("x", "Der ca lista")] allE2 2),
(etac impE 2), (simp_tac (esimps [less_Suc_eq_le]) 2),
(etac height_s 2), (Clarsimp_tac 2),
(REPEAT_DETERM_N 4 (thin_tac "valid ?r ?dt" 1)),
(chop_tac 5 (etac thin_rl 1)),
(* Set up endsequent for backward application of s43box *)
(etac s43box.elims 1), (etac s43box.elims 1),
(case_tac "flr" 1), (case_tac "flra" 1),
(clarsimp_tac (cesimps [extend_def, car_eq, nboxseq_def]) 1),
(etac mset_mapE 1), (etac mset_mapE 1), (Clarsimp_tac 1),
(res_inst_tac [("a","extend (nboxseq \
    \ (multiset1 + multiset1a |- multiset2 + multiset2a)) \
    \ (mset_map Box (gamma + gammaa - {#x#}) |- \
    \ mset_map Box (ms_of_list (remove1 x (As @ Asa))))")] mem_same 1),
(simp_tac (spsimps addsimps [nboxseq_def, mset_remove1,
    transfer gs43.thy inj_Box RS mset_map_diff_inj]) 2),
(rtac conjI 2),
(simp_tac (spsimps addsimps [mset_map_union, ms_of_append]) 3),
(rtac ([asm_rl, le_pm_assoc RS sym, le_pm_assoc RS sym] MRS box_equals) 3),
(rtac xt6 4), (rtac le_plus1 4),
(etac (mset_mapI RS (single_le RS iffD2)) 4),
(rtac xt6 4), (rtac le_plus2 4),
(etac (mset_mapI RS (single_le RS iffD2)) 4),
(simp_tac spsimps 3),
(simp_tac (spsimps addsimps [mset_map_union]) 2),
(rtac ([asm_rl, le_pm_assoc RS sym, le_pm_assoc RS sym] MRS box_equals) 2),
(rtac xt6 3), (rtac le_plus2 3),
(etac (mset_mapI RS (single_le RS iffD2)) 3),
(rtac xt6 3), (rtac le_plus1 3),
(etac (mset_mapI RS (single_le RS iffD2)) 3),
(simp_tac spsimps 2),
(* apply s43box rule *)
(rtac drs.derI'' 1), (rtac gs43_rls.boxI 1), (rtac s43box.I 1),
(clarsimp_tac (cesimps [nprems_def, ithprem_def]) 1),
(* two cases for premises, in (As - x) or Asa *)
(case_tac "xa : set (remove1 x As)" 1),
(ftac (set_remove1_app RS subsetD) 2),
(Clarsimp_tac 2),
(* cut on left premises, and new right *)
(chop_last (ftac (set_remove1_subset RS subsetD) 1)),
(fatac (sym RS maps_eq) 1 1), (Clarify_tac 1),
(chop_last (datac bspec 1 1)),
(chop_last (dtac sym 1)),
(clarsimp_tac (cesimps [ithprem_def, car_eq]) 1),
(* use wk_adm *)
(etac (gs43_wk_adm RS wk_admD_alt) 1),
(simp_tac (spsimps addsimps [mset_remove1]) 1),
(rtac conjI 1), (stac (le_pm_assoc RS sym) 1),
(chop_last (dtac (single_le RS iffD2) 1)), (atac 1),
(simp_tac (esimps [mset_map_union, transfer gs43.thy
    inj_Box RS mset_map_diff_inj]) 1),
(simp_tac (esimps [ms_of_append, mset_map_union, transfer gs43.thy
    inj_Box RS mset_map_diff_inj]) 1),
(stac (le_pm_assocc RS sym) 1), (rtac (single_le RS iffD2) 1),
(rtac mset_mapI 1), (atac 1),
(stac (le_pm_assocc RS sym) 1),
(chop_tac 1 (dtac (in_ms_of RS iffD2) 1)),
(chop_last (dres_inst_tac [("f", "Box")] mset_mapI 1)),
(chop_last (dtac (single_le RS iffD2) 1)),
(full_simp_tac (esimps [mset_remove1, transfer gs43.thy
```

```
      inj_Box RS mset_map_diff_inj]) 1),
  (simp_tac spsimps 1),
  (rtac xt6 1), (rtac pm_assoc_le 2),
  (simp_tac (esimps [mm_comm]) 1),
  (* second case, cut on right premises, and new left *)
  (chop_last (fatac (sym RS maps_eq) 1 1)), (Clarify_tac 1),
  (chop_tac 2 (dtac bspec 1)), (atac 1),
  (chop_last (dtac sym 1)),
  (clarsimp_tac (cesimps [ithprem_def, car_eq]) 1),
  (* need to cut on subformula, using left premise with x unboxed *)
  (ftac (in_ms_of RS iffD1) 1),
  (fatac (sym RS maps_eq) 1 1), (Clarify_tac 1),
  (chop_tac 1 (dtac bspec 1)), (atac 1),
  (dtac derrec_valid 1), (Clarify_tac 1),
  (etac allE 1), (etac impE 1), (rtac Box_ipsubfml 1),
  (etac allE2 1), (etac impE 1), (rtac conjI 1),
  (atac 1), (chop_last (atac 1)),
  (chop_tac 3 (dtac sym 1)),
  (clarsimp_tac (cesimps [car_eq, ithprem_def]) 1),
  (REPEAT_DETERM_N 4 (etac thin_rl 1)),
  (REPEAT (thin_tac "conclDT ?dt = ?x" 1)),
  (* contract extra copy of Box gamma and Box (As - x) *)
  (rtac ctr_wk_le_double 1), (Clarify_tac 1),
  (rtac gs43_ctr_adm 1), (rtac gs43_wk_adm 1), (atac 1),
  (thin_tac "?s : derrec gs43_rls {}" 1),
  (simp_tac (spsimps addsimps [mset_remove1, ms_of_append, mset_map_union,
       transfer gs43.thy inj_Box RS mset_map_diff_inj]) 1),
  (rtac conjI 1),
  (chop_last (ftac (single_le RS iffD2) 1)),
  (chop_last (dres_inst_tac [("f","Box")] mset_mapI 1)),
  (chop_last (dtac (single_le RS iffD2) 1)),
  (stac (le_pm_assoc RS sym) 1), (atac 1),
  (stac (le_pm_assoc RS sym) 1), (atac 1),
  (simp_tac spsimps 1),
  (rtac xt6 1), (rtac pm_assoc_le 2),
  (stac (le_pm_assoc RS sym) 1), (atac 1),
  (stac (le_pm_assocc RS sym) 1), (atac 1),
  (simp_tac spsimps 1),
  (chop_tac 1 (dtac (in_ms_of RS iffD2) 1)),
  (chop_tac 2 (dtac (in_ms_of RS iffD2) 1)),
  (REPEAT_DETERM_N 8 (etac thin_rl 1)),
  (dres_inst_tac [("f","Box")] mset_mapI 1),
  (dres_inst_tac [("f","Box")] mset_mapI 1),
  (REPEAT (dtac (single_le RS iffD2) 1)),
  (stac (le_pm_assocc RS sym) 1), (atac 1),
  (stac (le_pm_assoc RS sym) 1), (atac 1),
  (simp_tac spsimps 1)
]) ;
```

Glossary

Int Intuitionistic Propositional Logic. vii, 5, 57–64, 68

admissible A rule r is admissible for a calculus $\mathbf{S}$ if for all instances of r, whenever the premises of r are derivable, then so is the conclusion. Also, r should not be a rule in $\mathbf{S}$. 13, 14, 16, 32

ATP Automated Theorem Prover. 1, 9, 38, 47, 57, 58, 68, 69

BDD Binary Decision Diagram. 58

cut-free In the context of a sequent calculus, "cut-free" means that the cut rule is not one of the rules of the calculus. In a derivation, it means the derivation does not contain an instance of the Cut rule. 2, 16

ITP Interactive Theorem Prover. vii, 1, 9, 10, 22, 23, 26, 47, 57

logical In the context of a sequent rule, logical rules act on a logical operator. The names of such rules include the respective operator, for example R$\wedge$, L$\Box$. 1, 6, 11, 14, 40

meta-level Reasoning about proofs, typically proofs produced by systems which in turn perform object-level reasoning.. 8, 9

object-level Reasoning which proves properties about objects within some theory. Generally used in this work to refer to sequent proofs.. 8, 9

structural In the context of sequent rules, a structural rule is one that does not apply to a logical operator. In this work, the term generally refers specifically to the rules: Weakening, Contraction, and Cut. vii, 1, 6, 9, 11, 14, 29, 37, 38

structural-free calculi A sequent calculus which does not contain structural rules. 57

sub-sequent A sequent $\Gamma_1 \vdash \Delta_1$ is a sub-sequent of $\Gamma_2 \vdash \Delta_2$ if $\Gamma_1 \leq \Gamma_2$ and $\Delta_1 \leq \Delta_2$. 26, 63

super-sequent A sequent $\Gamma_1 \vdash \Delta_1$ is a super-sequent of $\Gamma_2 \vdash \Delta_2$ if $\Gamma_2 \leq \Gamma_1$ and $\Delta_2 \leq \Delta_1$. 26, 62, 63

References

BELNAP, N. D. 1982. Display logic. *Journal of philosophical logic 11*, 4, 375–417. (p. 24)

BIERMAN, G. AND DE PAIVA, V. 1996. Intuitionistic necessity revisited. In *PROCEEDINGS OF THE LOGIC AT WORK CONFERENCE* (1996). (p. 38)

BUSS, S. R. 2012. Sharpened lower bounds for cut elimination. *J. Symb. Log. 77*, 2, 656–668. (p. 19)

CASTELLINI, C. 2006. Automated reasoning in quantified modal and temporal logics. *AI Communications 19*, 2, 183–185. (p. 47)

CLOUSTON, R. AND GORÉ, R. 2014. Sequent calculus in the topos of trees. To appear. (pp. 16, 68)

CORSI, G. AND TASSI, G. 2007. Intuitionistic logic freed of all metarules. *J. Symb. Log. 72*, 4, 1204–1218. (p. 59)

DAWSON, J. 2014a. Cut-elimination with explicit contraction. `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/seqms/calkxc.ML`. Included in Isabelle code base. (p. 22)

DAWSON, J. 2014b. Mix-elimination for S4. `http://users.cecs.anu.edu.au/~jeremy/isabelle/2005/seqms/S4ca.ML`. Included in Isabelle code base. (p. 37)

DAWSON, J. E. AND GORÉ, R. 2006. Machine-checked cut-elimination for display logic. (pp. 2, 10, 25)

DAWSON, J. E. AND GORÉ, R. 2010. Generic methods for formalising sequent calculi applied to provability logic. In *Proceedings of the 17th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, LPAR'10 (Berlin, Heidelberg, 2010), pp. 263–277. Springer-Verlag. (pp. 1, 2, 10, 23, 25)

DYCKHOFF, R. 1992. Contraction-free sequent calculi for intuitionistic logic. *J. Symb. Log. 57*, 3, 795–807. (p. 15)

ENGEL, A. 2010. *Verification, Validation and Testing of Engineered Systems* (1 ed.). Wiley Series in Systems Engineering and Management. Wiley. Wertvoll wegen den verschiedenen Black-Box-Testing Methoden für komplexe Systeme. (p. 1)

FAIRTLOUGH, M. AND MENDLER, M. 1994. An intuitionistic modal logic with applications to the formal verification of hardware. In L. PACHOLSKI AND J. TIURYN

Eds., *CSL*, Volume 933 of *Lecture Notes in Computer Science* (1994), pp. 354–368. Springer. (p. 57)

FELTY, A. AND THÉRY, L. 1997. Interactive theorem proving with temporal logic. *Journal of Symbolic Computation 23*, 4, 367–397. (p. 47)

FERRARI, M., FIORENTINI, C., AND FIORINO, G. 2010. fcube: An efficient prover for intuitionistic propositional logic. In *LPAR*, Volume 6397 of *Lecture Notes in Computer Science* (2010), pp. 294–301. Springer. (p. 58)

FERRARI, M., FIORENTINI, C., AND FIORINO, G. 2013. Contraction-free linear depth sequent calculi for intuitionistic propositional logic with the subformula property and minimal depth counter-models. *Journal of automated reasoning 51*, 2, 129–149. (pp. 59, 60, 63)

FIORENTINI, C. 2000. All intermediate logics with extra axioms in one variable, except eight, are not strongly ω-complete. *The Journal of Symbolic Logic 65*, 04, 1576–1604. (p. 64)

GENTZEN, G. 1935. Untersuchungen über das logische schlieSSen. i. *Mathematische Zeitschrift 39*, 1, 176–210. (pp. 11, 21)

GONTHIER, G. 2007. The four colour theorem: Engineering of a formal proof. In D. KAPUR Ed., *ASCM*, Volume 5081 of *Lecture Notes in Computer Science* (2007), pp. 333. Springer. (p. 1)

GORÉ, R. 1994. Cut-free sequent and tableau systems for propositional diodorean modal logics. *Studia Logica 53*, 3, 433–457. (pp. 5, 29, 47)

GORÉ, R. 1998. Substructural logics on display. *Logic Journal of the IGPL 6*, 3, 451–504. (p. 24)

GORÉ, R. 2009. Machine checking proof theory: An application of logic to logic. In R. RAMANUJAM AND S. SARUKKAI Eds., *ICLA*, Volume 5378 of *Lecture Notes in Computer Science* (2009), pp. 23–35. Springer. (pp. 1, 22, 23)

GORÉ, R. AND RAMANAYAKE, R. 2008. Valentini's cut-elimination for provability logic resolved. In *Advances in Modal Logic*, Volume 7, pp. 67–86. College Publications, London. (p. 23)

GORÉ, R. AND THOMSON, J. 2012. Bdd-based automated reasoning for propositional bi-intuitionistic tense logics. In *IJCAR*, Volume 7364 of *Lecture Notes in Computer Science* (2012), pp. 301–315. Springer. (p. 58)

GORÉ, R., THOMSON, J., AND WU, J. 2014. A history-based theorem prover for intuitionistic propositional logic using global caching: Inthistgc system description. In S. DEMRI, D. KAPUR, AND C. WEIDENBACH Eds., *IJCAR*, Volume 8562 of *Lecture Notes in Computer Science* (2014), pp. 262–268. Springer. (pp. vii, 59, 61, 62, 64)

GOUBAULT-LARRECQ, J. 1996. On computational interpretations of the modal logic s4 i. cut elimination. Technical report, Institut fur Logik, Komplexitat und Deduktionssysteme, Universitat. (pp. 38, 45)

HARLAND, J., LUTOVAC, T., AND WINIKOFF, M. 2000. Goal-directed proof search in multiple-conclusioned intuitionistic logic. In *In Proceedings of the First International Conference on Computational Logic, volume LNAI 1861* (2000), pp. 254–268. Springer. (p. 28)

HERBELIN, H. AND LEE, G. 2009. Forcing-based cut-elimination for gentzen-style intuitionistic sequent calculus. In *Logic, Language, Information and Computation*, pp. 209–217. Springer. (p. 23)

HERMANT, O. 2005. Semantic cut elimination in the intuitionistic sequent calculus. In P. URZYCZYN Ed., *TLCA*, Volume 3461 of *Lecture Notes in Computer Science* (2005), pp. 221–233. Springer. (p. 16)

HUSTADT, U. AND SCHMIDT, R. A. 1998. Simplification and backjumping in modal tableau. In *TABLEAUX*, Volume 1397 of *Lecture Notes in Computer Science* (1998), pp. 187–201. Springer. (p. 62)

INDRZEJCZAK, A. 2012. Cut-free hypersequent calculus for s4. 3. *Bulletin of the Section of Logic 41*, 1-2, 89–104. (p. 47)

LEIVANT, D. 1981. On the proof theory of the modal logic for arithmetic provability. *J. Symb. Log. 46*, 3, 531–538. (p. 23)

MCLAUGHLIN, S. AND PFENNING, F. 2008. Imogen: Focusing the polarized inverse method for intuitionistic propositional logic. In *LPAR*, Volume 5330 of *Lecture Notes in Computer Science* (2008), pp. 174–181. Springer. (p. 58)

MINTS, G. 2005. Cut elimination for provability logic. *Collegium Logicum*. (p. 23)

MOEN, A. 2001. The proposed algorithms for eliminating cuts in the provability calculus gls do not terminate. In *NWPT 2001, Norwegian Computer Center (2001-12-10)* (2001). (p. 23)

NEGRI, S. 2005. Proof analysis in modal logic. *Journal of Philosophical Logic 34*, 5-6, 507–544. (pp. 23, 38, 47)

NEGRI, S. 2011. Proof theory for modal logic. *Philosophy Compass 6*, 8, 523–538. (p. 15)

NEGRI, S. AND VON PLATO, J. 2001. *Structural proof theory*. Cambridge University Press. (pp. 17, 21, 43)

OHNISHI, M. AND MATSUMOTO, K. 1957. Gentzen method in modal calculi. *Osaka Mathematical Journal 9*, 2, 113–130. (p. 37)

OKADA, M. 2002. A uniform semantic proof for cut-elimination and completeness of various first and higher order logics. *Theor. Comput. Sci. 281*, 1-2, 471–498. (p. 16)

OSORIO, M. AND PÉREZ, J. A. N. 2004. Answer set programming and s4. In C. LEMAÎTRE, C. A. R. GARCÍA, AND J. A. GONZÁLEZ Eds., *IBERAMIA*, Volume 3315 of *Lecture Notes in Computer Science* (2004), pp. 353. Springer. (p. 37)

OSORIO, M., PÉREZ, J. A. N., AND ARRAZOLA, J. 2003. Applications of intuitionistic logic in answer set programming. *CoRR cs.LO/0305046*, 325–354. (pp. 1, 57)

PATTINSON, D. AND SCHRÖDER, L. 2009. Generic modal cut elimination applied to conditional logics. In *Automated Reasoning with Analytic Tableaux and Related Methods*, pp. 280–294. Springer. (p. 24)

PATTINSON, D. AND SCHRÖDER, L. 2010. Cut elimination in coalgebraic logics. *Information and Computation 208*, 12, 1447 – 1468. Special Issue: International Workshop on Coalgebraic Methods in Computer Science (CMCS 2008). (p. 23)

PAULSON, L. C. 2003. *The Isabelle Reference Manual*. University of Cambridge. (pp. 2, 10)

PAULSON, L. C. ET AL. 2013. Isabelle's logics. (p. 47)

PAULSON, L. C. AND BLANCHETTE, J. C. 2010. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. In *PAAR@ IJCAR* (2010), pp. 1–10. (p. 9)

POGGIOLESI, F. 2011. *Gentzen Calculi for Modal Propositional Logic*, Volume 32 of *Trends in Logic*. Springer Netherlands. (pp. 11, 14)

RASGA, J. 2005. Cut elimination for a class of propositional based logics. Technical report, Instituto Superior Técnico. (pp. 24, 32, 68)

RATHS, T., OTTEN, J., AND KREITZ, C. 2007. The ILTP problem library for intuitionistic logic. *J. Autom. Reasoning 38*, 1-3, 261–271. (p. 64)

SAVNIK, I. 2013. Index data structure for fast subset and superset queries. In *Availability, Reliability, and Security in Information Systems and HCI*, Volume 8127 of *Lecture Notes in Computer Science*, pp. 134–148. Springer. (p. 63)

SHIMURA, T. 1991. Cut-free systems for the modal logic S4.3 and S4.3Grz. *Reports on Mathematical Logic 25*, 57–72. (p. 47)

STATMAN, R. 1979. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci. 9*, 67–72. (p. 59)

TEWS, H. 2013. Formalizing cut elimination of coalgebraic logics in coq. In D. GALMICHE AND D. LARCHEY-WENDLING Eds., *Automated Reasoning with Analytic Tableaux and Related Methods - 22th International Conference, TABLEAUX 2013, Nancy, France, September 16-19, 2013. Proceedings*, Volume 8123 of *Lecture Notes in Computer Science* (2013), pp. 257–272. Springer. (p. 23)

TROELSTRA, A. S. AND SCHWICHTENBERG, H. 2000. *Basic Proof Theory (2nd Ed.)*. Cambridge University Press, New York, NY, USA. (pp. 3, 11, 14, 15, 16, 17, 21, 22, 37, 43, 45)

VALENTINI, S. 1983. The modal logic of provability: Cut-elimination. *Journal of Philosophical Logic 12*, 4, pp. 471–476. (p. 23)

VON PLATO, J. 2014. The development of proof theory. In E. N. ZALTA Ed., *The Stanford Encyclopedia of Philosophy* (Winter 2014 ed.). Stanford University. (p. 11)

WANSING, H. 1995. Strong cut-elimination in display logic. *Reports on Mathematical Logic 29*, 117–131. (p. 24)