FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Applications of Higher-Order Cut-Elimination

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

within the

Vienna PhD School of Informatics

by

Martin Riener
Registration Number 9927068

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.-Prof. Dr. Alexander Leitsch

External reviewers:
PD Dr. Christoph Benzmüller
Prof. Dr. Michael Kohlhase

Wien, 20.07.2017

(Signature of Author) (Signature of Advisor)

Declaration of Authorship

Martin Riener
Ernst Melchior Gasse 11/3/5, 1020 Wien

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

(Place, Date) (Signature of Author)

Acknowledgements

Out of the long list of people making this thesis possible, my deepest gratitude goes to my advisor Alex Leitsch. His wide knowledge, guidance and patience has been the foundation on which I could explore the winded paths of logic where, as he once said in a little different context, “everything else is just a practical problem”.

I am equally grateful to my wonderful colleagues from Vienna with whom I shared the experience: Cvetan Dunchev, David Cerna, Gabriel Ebner, Stefan Hetzl, Tomer Libal, Giselle Reis, Christoph Roschger, Mikheil Rukhaia and Bruno Woltzenlogel-Paleo. Also without my incredible current team in Nancy, this thesis would not have happend. The discussions with Jasmin Blanchette, Damien Doligez, Pascal Fontaine, Stephan Merz and Simon Cruanes were invaluable and gave me essential insights from completely different points of view. A thousand thanks go also to my reviewers Christoph Benzmüller and Michael Kohlhase for fighting through this highly technical and exhausting material.

Finally, my thanks go to my family and friends, especially to my parents Krista and Alfons, my sister Ingrid, to Tamás Schmidt and Mani Esmaeili. A special place in my heart is occupied by Sylwia Polberg, with whom I am happy to share my feelings as well as my thoughts, even when they are as obscure as proof theory.

Abstract

The method of Cut-Elimination by Resolution (CERES) [7,8] bridges the fields of proof theory and automated theorem proving. Its proof theoretic value lies in the extraction of information from a theorem beyond its mere truth while its contribution to the automated theorem proving community lies in the generation of hard problems for which, at least in theory, a proof can always be found. An example for such a successful analysis in first-order logic was the formalization of Fürstenberg's proof of the infinity of primes [5].

So far, CERES$^{\omega}$, the extension of CERES to higher-order logic was missing an application of equivalent complexity. This thesis develops the CERES$^{\omega}$ method further, improving the interplay with automated higher-order theorem provers. Furthermore we explore the use of expansion proofs as a concise representation of witness terms for weak quantifiers, the essential information generated by CERES. Moreover, we introduce CERES$^{\omega}_{=}$, a variant which targets the fragment of functional quantification with first-order equality and definition rules. Finally, we introduce LLK, a LaTeX inspired proof input language and Sunburst trees, a global visualization of sequent calculus proofs and the implementation of these proof analysis techniques in the GAPT system.

As a case study, we formalize the infinite pigeon hole principle and extract functions enumerating arbitrary many pigeons in the infinite-sized hole. We compare our results to those of Ratiu and Trifonov which were found via A-translation and modified Realizability [79]. Simultaneously, we explore the gradient of solvable problems for higher-order provers, identifying possible directions for future development.

Kurzfassung

Die Methode der Schnittelimination mittels Resolution (CERES) verbindet die Gebiete der Beweistheorie und des automatischen Beweisens. Einerseits erlaubt sie, Aussagen zu einem Beweis zu treffen, die über die reine Gültigkeit des Satzes hinausgehen, andererseits liegen die dabei entstehenden Teilprobleme an der Grenze der Kapazitäten aktueller Theorembeweiser. Da sie aufgrund ihrer Konstruktion lösbar sind, tragen sie so zu deren Weiterentwicklung bei. In der Prädikatenlogik erster Stufe ließ sich bereits mittels CERES zeigen, dass sich die topologischen Argumente in einer Formalisierung von Fürstenbergs Beweis der Existenz unendlich vieler Primzahlen direkt auf die zahlentheoretischen Argumente Euklids zurückführen lassen. Für CERES$^{\omega}$, die Erweiterung von CERES für die Prädikatenlogik höherer Stufe, fehlt solch eine experimentelle Anwendung noch.

Diese Dissertation behandelt die Anpassung von CERES$^{\omega}$ an Fragmente der Logik höherer Stufe, die an die Fähigkeiten entsprechender Theorembeweiser angepasst sind. Die CERES$^{\omega}_{=}$ genannte Variante der Methode beschränkt sich zwar auf Quantoren über Funktionen anstatt auch solche über Prädikate zuzulassen. Dafür erlauben zusätzliche Regeln für Definitionen und Gleichheitsersetzungen von Termen erster Ordnung eine natürlichere Formulierung mathematischer Beweise. Ein weiteres Thema ist die Erkenntnis, dass Millers Expansionsbeweise die für unsere Analyse notwendigen Informationen in Form von Quantoreninstantiierungen enthalten. Ihre Verwendung erlaubt es, mehrere aufwändige Transformationen von Beweisen im Sequenzenkalkül auszulassen. Alle genannten Methoden sind im System GAPT implementiert, das nun auch neben der klassischen Baumdarstellung eine kreisförmige Gesamtübersicht über Sequenzenkalkül-artige Beweise erlaubt.

Als Anwendungsbeispiel dient eine Formalisierung des unendlichen Schubfachprinzips, welches in ähnlicher Form schon von Ratiu und Trifonov für deren Experimente zur Programmextraktion mittels A-Übersetzung und modifizierter Realisierbarkeit herangezogen wurde. Die mittels CERES$^{\omega}_{=}$ gewonnenen Funktionsterme werden den extrahierten Programmen gegenübergestellt und erlauben einen Einblick, wie aus klassischen Beweisen stammende Programme aussehen können. Die bei der Analyse entstandenen Probleme für Theorembeweiser gaben den Anlass, die TPTP Bibliothek um einfache Syntheseaufgaben für Funktionen zu erweitern und könnten Impulse für die Weiterentwicklung von automatischen Beweisern geben.

Contents

CHAPTER 1

Introduction

A fascinating field of proof theory is proof mining, where we extract information from a formal proof which goes beyond the truth of the theorem. For instance, proving that there exists an infinite series of (different) prime numbers is already a reward by itself. However when we investigate Euclid's famous proof of this theorem, we can actually obtain more information from each prefix $p_1, \ldots, p_n$ of the series: we know that the term $p_1 \cdot \ldots \cdot p_n + 1$ used in the proof is either a prime itself or has a divisor which is not contained in the prefix. Therefore the term is a rough upper bound for prime gaps i.e. the distance between consecutive primes.

One way to obtain this term is by formalizing this argumentation as a proof of $\forall x \exists y(Prime(x) \rightarrow (x < y \wedge Prime(y)))$ in Sequent Calculus. After cut-elimination, the rules introducing the quantifier $\exists y$ contain the term above as witness. In general, different proofs of the same statement have different witnesses. Therefore the instantiation terms can be seen as characteristic for the mathematical argumentation behind the proof. Herbrand's theorem justifies this procedure in a general setting. In a simplified form, it states that the prenex formula $\exists x\, F[x]$ is true exactly when there exists a finite number of terms $t_1, \ldots, t_n$ such that $F[t_1] \vee \ldots \vee F[t_n]$ is true. Since a Herbrand disjunction is purely propositional and checking its validity is decidable, the quantifier instantiations contribute significantly to the truth of the theorem.

In the context of cut-elimination by resolution (CERES), the analysis via Herbrand sequents – the generalization of Herbrand disjunctions to non-prenex formulas and sequents – was applied [49] to a proof showing the equivalence of three axiomatizations L1-L3 of lattices by proving L3 from L1 and L2 from L3. After cut-elimination, the Herbrand sequent of the proof of L2 from L1 alone provided the instantiations necessary to construct an independent direct proof which is even simple enough to allow a formulation in mathematical prose.

Since whenever a method works well, it is applied to harder problems, the same holds for proof analysis via cut-elimination. In this case this means investigating proofs in higher-order logic. There matters become more complicated: the most obvious obstacle is the loss of completeness and compactness. Likewise, important properties for automated reasoning are impaired: first-order style skolemization is unsound and the unification is of infinitary type. Furthermore the high expressivity often surpasses human intuition. For example, Girard showed that the inductive definition of the natural numbers in second-order arithmetic makes the induction axiom superfluous by relativizing each quantified variable to be in the natural numbers. [40, p178-180]. Even the notion of cut becomes fuzzy, since the formula $\forall X(X \rightarrow X)$ can be used to simulate the cut-rule. Benzmüller et. al. [10] showed that the same holds for Leibniz' axiomatization of equality and the axioms for extensionality.

When we want to interpret instantiation terms, the formula language becomes even more

important. Simply typed lambda calculus is an excellent choice: on one hand its computation model is easy to treat, since β-reduction terminates. On the other hand, it is sufficiently expressive to form the foundation of Simple Type Theory, when it is extended with the rules of generalization and modus ponens and Hilbert-style axioms. On the other hand, the choice of the axiomatization and restrictions of the proof calculus like eigenvariables constrain the expressivity of functions written in simply typed lambda calculus.

Nonetheless the relation of lambda terms to functional programs is so close, that it is tempting to interpret them this way. Since we can prove the existence of uncomputable functions it is obvious that this interpretation is impossible in the general case. Still, it is unclear where this additional expressiveness is hidden.

Therefore the first group of research questions we seek to answer is:

How do instantiation terms look like when we transfer the methods we developed for first-order logic to higher-order logic? How do these terms interact with the logical layer? Can we interpret them as functional programs - if not why?

Naturally, we are convinced that the methods we developed over the last years are helpful for proof mining. Foremost, this involves the development of extraction of Herbrand disjunctions for CERES^{ω} [48, 111], the higher-order counterpart to CERES [7, 8]. The proposed data-structure to represent Herbrand-instances are the more general expansion trees. An expansion tree can hold the instantiations of higher-order quantifiers and is not restricted to prenex formulas. There is also a variant with skolem terms instead of eigenvariables. Since CERES^{ω} contains the refutation of a higher-order formula as its central step, it is directly bound to the capabilities of higher-order automated theorem provers like Leo II and Satallax or alternatively to a human found refutation. The previous experiments with Fürstenberg's topological proof of the infinity of primes exceeded the limits of both. Therefore we were looking for possible restrictions which ease the burden of the ATPs, in particular by removing higher-order substitutions from the logical layer. We also believe that the adaption of first-order theorem provers as back-ends provides an efficient handling of first-order equality in higher-order provers and want to integrate equational reasoning on individuals into CERES^{ω}.

This leads us the second set of questions for this thesis:

Are expansion trees useful for proof mining via cut-elimination by CERES^{ω}*? Can we find an interesting theorem as a case study for the method? Does a restriction to function quantifiers and first-order equality create problem sets which ATPs handle well?*

1.1 Methodology

The thesis has two parts: one is purely deductive and develops the CERES^{ω} method into the direction of better integration with theorem provers. Another focus lies on methods for proof formalization: extending the calculi with rules for first-order equality, definitions and macros necessitates the introduction of proof transformations for their elimination. These transformations are shown to be sound and, if possible, complete.

The second part of the thesis concerns itself with the experimental evaluation of the extended method. The variations of mathematical proofs - in fact even the varying formalizations of a sin-

gle proof - are so numerous that the use of quantitative investigation is severely limited. Proof collections like the TSTP Solution Library [100, 102], the Mizar Mathematical Library [107] or the Archive of Formal Proofs [58] exist, but these proofs require considerable effort to translate to Sequent Calculus without introducing additional cuts. For this reason we investigate the infinite pigeon hole principle as a case study. It has already been studied using program extraction via the refined A-Translation and Gödel's Dialectica interpretation [79] which allows a comparison with the results obtained by cut-elimination. To make matters more challenging, we use a purely classical proof[1] of the finite pigeon hole principle as a lemma without an embedding in intuitionistic logic.

Experiments also call for an implementation. Since the General Architecture for Proof Theory (GAPT) [33] was designed for proof mining, it is natural to extend it with the methods described here. On one hand, this is centered around classical software-engineering problems like interfacing with various external software. This includes writing parsers and converters of the output of theorem provers, designing an input language for the new calculi and continue Daniel Weller's implementation of CERES^{ω}. On the other hand, the proof objects we are investigating can easily have thousands of inferences. Therefore the implementation also contains work on graphical user interfaces for proof exploration.

The structure of the thesis is as follows: chapter 2 introduces the logic and calculi used in the thesis and recapitulates the results and algorithms for expansion proofs, reductive cut-elimination and CERES^{ω} which will be adapted later on. Chapter 3 shows how expansion proofs can significantly shorten the analysis process of CERES^{ω} and introduces $\mathrm{CERES}^{\omega}_{=}$ for the fragment of functional quantification extended by definitions and first-order equality. It also discusses the methods used to formalize a proof, to make the extracted sequents sets tractable for automated theorem provers, global visualization methods and the implementation in GAPT. Chapter 4 presents the case study of our formulation of the infinite pigeon hole principle and compares our results with those obtained by modified Realizability. The conlusion in chapter 5 wraps the thesis up, investigates some of the lessons learned and identifies future directions.

[1] The proof inherently relies on the law of excluded middle and cannot be proved in intutitionistic logic without a double-negation translation.

2.1 Simply Typed Lambda Calculus and Elementary Type Theory

Even though the major proof assistants Isabelle and Coq use more expressive systems, elementary type theory is already sufficient to naturally express mathematical problems. Since we also rely on the communication with external theorem provers, staying in a a subset of simple type theory and therefore within THF0 [14], the core TPTP fragment for higher-order logic and the best supported is also desirable.

Following the presentation of elementary type theory [24] given by Benzmüller and Miller [12] and Miller [71, 72], we will start with Church's simply typed lambda calculus. We call the primitive type of booleans o and the primitive type of individuals ι. Complex types are inductively constructed from simple types and basic types via the type application $>$ which is considered as right-associative.

We denote the type α of constants and variables as subscript to their name and call the corresponding (infinite) sets Σ_α and $\mathcal{V}_\alpha$. The sets Σ and $\mathcal{V}$ then denote the union of all sets of constants and variables for each simple type. If the type is not relevant, it will be omitted. If not noted otherwise, the sets $\{a, b, c, d\}$ and $\{i, \ldots, m\} \cup \{x, \ldots, z\}$ will be used for constants and variables of type ι, the sets $\{P, \ldots, S\}$ and $\{X, Y, Z\}$ will be used for constants and variables of type $\alpha_1 > \ldots > \alpha_n > o$ (predicates) and the sets $\{f, g\}$ and $\{h\}$ will be used for constants and variables of other types; if more names are necessary, we will add indices to the names. Terms are built from constants and variables by lambda abstraction and application:

Definition 2.1.1 (Terms of Simply Typed Lambda Calculus)**.**

- A constant c_α and a variable x_α are terms of type α.
- If x_α is a variable and t_β is a term, then $\lambda x.t$ is a term of type $\alpha > \beta$.
- If $s_{\alpha>\beta}$ and t_α are terms, then $s\ t$ is a term of type β.

For readability purposes we will write $P(f(x), x)$ for $P\ \ (f\ \ x)\ \ x$, i.e. an application $s(t_1, \ldots, t_n)$ stands for $(\ldots(st_1)\ldots t_n)$.

We consider a variable x as free in a term t if it occurs outside the context of a corresponding binder $\lambda x\ \ldots$ in t. We consider x as bound in t if it occurs inside such a binder. The set $FV(t)$ denotes all free variables occurring in t. A variable may occur bound and free in the same term (e.g. in $(\lambda x\ x)\ x$) and it may be overbound (e.g. in $\lambda x(\lambda x\ x)$ where the variable x is bound by the inner λx but not by the outer one).

A substitution is a mapping of finitely many variables x_α to terms t_α. Applying a substitution $\sigma = \{x \leftarrow t\}$ to a term s is written in post-fix notation $s\sigma$. It is performed by replacing all free occurrences of x in s by t, provided that no variable $y \in FV(t)$ becomes bound (is captured).

Simply typed lambda calculus then is an equational theory over simply typed terms with the reflexive-transitive closure of the following rules:

Definition 2.1.2 (α-conversion, β-reduction and β-expansion)**.**

- α-conversion: if the term t results from the term s by replacing all occurrences of a bound variable x_γ by a a bound variable y_γ, then s alpha-converts to t.
- β-reduction: the term $(\lambda xs)\ t$ beta-reduces to the term $s\{x \leftarrow t\}$, where all free occurrences of x in s are substituted by the term t.
- β-expansion: conversely, the term $s\{x \leftarrow t\}$ beta-expands to the term $(\lambda xs)\ t$.

Alpha-conversion is implicitly performed whenever necessary. In particular it may be required during substitution to prevent overbinding: evaluating $(\lambda x\ y)\{y \leftarrow x\}$ requires a renaming of x in the binder. Therefore it is equal to $\lambda z\ x$ but not to $\lambda x\ x$.

Beta-reduction is a strongly normalizing rewrite rule which produces unique normal-forms. To decide the β-equality of two terms s and t, it is therefore sufficient to compare their normal forms $s{\downarrow_\beta}$ and $t{\downarrow_\beta}$.

To prepare for the introduction of logical operators $\neg, \vee, \wedge, \rightarrow, \forall, \exists$, we define an atom formula as a variable of type o, a constant of type o or an application $s\ t$ of type o where s is a constant or variable different from the logical operators. A formula then is an atom formula or a term $\neg F$, $F \wedge G$, $F \vee G$, $F \rightarrow G$, $\forall x\ F$ or $\exists x\ F$ of type o where F and G are formulas and x is a variable of arbitrary type. Given this definition the term $(\lambda X\ X)X_o$ is not atomic but its β-normal form X_o is. For this reason we will often implicitly normalize a term, in particular when used in calculi. It is also noteworthy that atomicity is not closed under substitution. For instance $X\{X \leftarrow (Y \rightarrow Y)\}$ evaluates to $Y \rightarrow Y$ which is clearly not atomic.

We will regularly encounter the notion of polarity and weak / strong quantifiers. The intuition is that a strong quantifier becomes universal when the formula is transformed into prenex normal-form and a weak quantifier becomes existential after such a transformation. Since shifting negation over a quantifier inverts the polarity, the polarity of a formula propagates to its immediate sub-formulas as follows: if $\neg F$ is a positive (negative) occurrence then F is a negative (positive) occurrence. If $A \rightarrow B$ is a positive (negative) occurrence then A is a negative (positive) occurrence and B is a positive (negative) occurrence. If $A \wedge B$, $A \vee B$, $\forall F$ or $\exists F$ is a positive (negative) occurrence then A, B and F are positive (negative) occurrences. If the formula $\forall F$ ($\exists F$) occurs positively (negatively), then it is called a strong quantifier. If it occurs negatively (positively) then it is called a weak quantifier occurrence.

Elementary type theory (ETT) which is also called Church's $\mathcal{T}$[1] is a Hilbert style deduction system. We will use $\neg$, $\vee$ and $\forall$ as basic operators and have the usual abbreviations with $\forall x_\alpha F_o$

[1] We will usually write ETT since Gödel's system $\mathcal{T}$ plays a role in our comparison with alternative approaches in section 4.13.

for $\forall_{\alpha>o}(\lambda x F)$, $\exists x F$ for $\neg\forall x \neg F$, $P \wedge Q$ for $\neg(\neg P \vee \neg Q)$ and $P \to Q$ for $\neg P \vee Q$. The axioms of ETT are:

1. Propositional Axioms:
   - $P \vee P \to P$
   - $P \to P \vee Q$
   - $P \vee Q \to (Q \vee P)$
   - $P \to Q \to ((R \vee P) \to (R \vee Q))$
2. for each simple type α: $\forall_{(\alpha>o)>o} F_{\alpha>o} \to F_{\alpha>o}(x_\alpha)$
3. for each simple type α: $\forall X_\alpha (p_o \vee F_{\alpha>o}(X)) \to (p \vee \forall_{(\alpha>o)>o} F)$

The inference rules of ETT are:

1. One step of α-conversion, β-reduction or β-expansion
2. Substitution: from $F_{\alpha>o}(X_\alpha)$ infer $F(A_\alpha)$ if X is not free in A.
3. Modus Ponens: from P and $P \to Q$ infer Q
4. Generalization: from $F_{\alpha>o} X_o$ infer $\forall_{(\alpha>o)>o} F$ if X is not free in F.

Simple Type Theory (STT) is built on top of Elementary Type Theory. It adds an equality symbol for each type to the language and axioms for Leibniz Equality, Church Numerals, Infinity, Description, Functional and Boolean Extensionality, the Axiom of Choice and an axiom ensuring the existence of two individuals to the inference system.

There are multiple reasons for us to stick to ETT. Foremost, we are interested in the interplay of different proof calculi which share the same logic. The triplet of Sequent Calculus (as defined by Miller [71][2]), Expansion Proofs (also defined by Miller [71, 72]) and ETT was shown to be relatively equivalent by Miller [71, 72]. Also Andrews' Resolution Calculus $\mathcal{R}$ is refutationally complete to ETT in the sense that if a finite set of sentences S derives falsum in ETT, then there is also a refutation of S in $\mathcal{R}$ [2].

Furthermore, most of the additional axioms (Leibniz Equality, Choice, Extensionality) of STT are cut-strong [10], which means that they may simulate a cut-rule, effectively forcing an automated theorem prover to find the instantiation of a weak quantifier without any guidance. This effect is hard to control since the input proofs for the CERES method are usually interactively generated but during cut-elimination, an ATP is required to refute the characteristic sequent set generated from that input proof. Since the formulas in the sequent set come from the introduction rules of the proof, a cut-strong axiom in the end-sequent does not necessarily pose a problem. For instance, if the induction axiom is instantiated with a first-order invariant, its corresponding formulas in the CSS will also be first-order and therefore unable to simulate

[2] Similar presentations were done by Takeuti [104], Schütte [91]. According to Andrews [2], these are slightly weaker than ETT since in the latter "for all types α and β there is a type $\beta > \alpha$".

the cut rule. Nevertheless, the exact instantiations necessary become only visible during proof formalization. Some cut-strong formulas in the CSS can be avoided by reorganising the proof but naturally this cannot always work. The case study even shows an example (see section 4.12) where the characteristic sequent set becomes cut-strong by proving a weaker result[3].

2.1.1 Expressivity of Simply Typed Lambda Calculus

Since simply typed lambda-terms will be used to express instantiation terms of quantifiers, we are interested which kind of functions we can express. A foundational result [98] of Statman (later reproved by Mairson [66]) states that deciding the equality of two terms in simply typed lambda calculus is non-elementary. Using Knuth's up-arrow notation, the bound is $2 \uparrow\uparrow \mathcal{O}(n)$. The underlying reason is that computing the normal-form of a simply typed lambda term can cause a non-elementary blow-up in term size [95]. [4]

Using Church numerals, Schwichtenberg showed [88] that functions of simply typed lambda-calculus coincide with the extended polynomials: a numeral n is a term $\lambda\alpha.\alpha^n$ of type $num = (\iota > \iota) > (\iota > \iota)$ with the term $\alpha^n = \lambda x.\alpha(...(\alpha(x))$ having n iterations of α. Then addition is expressed as $plus = \lambda m_{num}\lambda n_{num}\lambda x_{\iota>\iota}\lambda u_\iota.(mx)((nx)u)$[5] and multiplication is expressed as $mul = \lambda m_{num}\lambda n_{num}\lambda u_{\iota>\iota}.m(nu)$. A constant function n is expressed as $\lambda x.\alpha^n$ and a case distinction $d(n, m, 0) = n, d(n, m, i)$ for $0 \neq i$ as $\lambda m_{num}\lambda n_{num}\lambda e\lambda u_{\iota>\iota}\lambda x_\iota.e(\lambda y_\iota.(mu)x)((nu)x)$.

2.1.2 Skolem Terms in Elementary Type Theory

Since both Skolem Expansion Proofs (see section 2.3) and LK_{skc} (see section 2.6.1) will introduce skolem terms we repeat the definitions of Miller [71] which introduce skolem functions, skolem terms and ETT extended by skolem terms.

Definition 2.1.3. The list $\sigma = \langle \alpha, \beta_1, \ldots, \beta_p \rangle$ where $\alpha, \beta_1, \ldots, \beta_p$ are simple types ($p \geqslant 0$) is called the signature of a skolem function. For each signature σ, let $\mathcal{K}_\sigma$ be a denumerable infinite set of function symbols of type $\beta_1 > \ldots > \beta_p > \alpha$ which are not in the formulation of ETT and such that for different signatures σ_1 and σ_2 the sets $\mathcal{K}_{\sigma_1}$ and $\mathcal{K}_{\sigma_2}$ are disjoint. If $f \in \mathcal{K}_\sigma$ then f is called a skolem function of signature σ with arity p. Let ETT * be the formulation of ETT with skolem functions added.

Now we define the Herbrand Universe $\mathcal{U}$ as union of universes $\mathcal{U}_\alpha$ for each type α of terms of ETT *. This also includes skolem terms, which are skolem functions that are applied to at least their arity number of arguments.

Definition 2.1.4. Let $\mathcal{U}$ be the smallest set of formulas of ETT * such that

- All variables and constants (which are no skolem functions) are in $\mathcal{U}$.

[3] The reason is that the if-then-else conditional is skolemized in the induction proof of the general statement. Proving a specific instance works without induction and chases the ATP off to find arbitrary conditionals.

[4] Examples for this behavior are contained in the book as exercises 3.20-3.22.

[5] The original formulation uses function composition instead of explicitly using u.

- If $p \geqslant 0$, $t_i \in \mathcal{U}_{\beta_i}$ for $1 \leqslant i < p$ and the skolem function f has signature $\langle \alpha, \beta_1, \ldots, \beta_p \rangle$, then $ft_1, \ldots, t_p \in \mathcal{U}_\alpha$. The term $ft_1, \ldots, t_p$ is called a skolem term. Since f may have more than p arguments, the terms $t_1, \ldots, t_p$ are called the necessary arguments of f.
- If $s \in \mathcal{U}_{\alpha>\beta}$ and $t \in \mathcal{U}_\alpha$ then $st \in \mathcal{U}_\beta$.
- If $s \in \mathcal{U}_\alpha$ and x is a variable of type β which is not free in any necessary arguments of any skolem function occurrence, then $(\lambda x\ s) \in \mathcal{U}_{\beta>\alpha}$.

2.2 Sequent Calculus LK

Sequent calculus was originally developed by Gentzen to prove the completeness of natural deduction via cut-elimination [39]. Even though there are sequent calculi for simple type theory, we will restrict ourselves to one for elementary type theory, given by Weller [48, 111]. It is essentially the same as Miller's LKH [71, 72] with the common modification that a sequent consists of multisets instead of lists of formulas, making the exchange rule[6] superfluous.

Also the λ-rule applying β-normalization is replaced by an implicit normalization of each formula. LKH again differs from Gentzen's LK only by using simply typed lambda calculus as term language which leads to more expressive quantifier rules and the addition of the λ-rules.

We use Gentzen's original definition of a sequent:

Definition 2.2.1 (Sequent)**.** A sequent S is a pair $(\{F_1, \ldots, F_n\}, \{G_1, \ldots, G_m\})$ of (possibly empty) lists of formulas written as $F_1, \ldots, F_n \vdash G_1, \ldots, G_m$. The formulas $F_1, \ldots, F_n$ are called the antecedent of the sequent and the formulas $G_1, \ldots, G_m$ are called the succedent of the sequent. The logical interpretation of S is the formula $F_1 \wedge \ldots \wedge F_n \to G_1 \vee \ldots \vee G_m$. An empty antecedent corresponds to $\top$, an empty succedent to $\bot$ (the respective unit elements of conjunction and disjunction).

A clause can be seen as a sequent of atom formulas where the negative literals occur in the antecedent and the positive literals occur in the succedent. Since the CERES method combines resolution and sequent calculus, will use sequent notation for clauses throughout this text.

Then the rules of LK used here are as follows:

Definition 2.2.2 (Rules of LK)**.**

Propositional rules:

$$\frac{}{F \vdash F}\ \text{ax} \qquad \frac{\Gamma \vdash \Delta, F}{\neg F, \Gamma \vdash \Delta}\ \neg : l \qquad \frac{F, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \neg F}\ \neg : r$$

$$\frac{F, \Gamma \vdash \Delta \quad G, \Pi \vdash \Lambda}{F \vee G, \Gamma, \Pi \vdash \Delta, \Lambda}\ \vee : l \qquad \frac{\Gamma \vdash \Delta, F}{\Gamma \vdash \Delta, F \vee G}\ \vee : r^1 \qquad \frac{\Gamma \vdash \Delta, G}{\Gamma \vdash \Delta, F \vee G}\ \vee : r^2$$

$$\frac{\Gamma \vdash \Delta, F \quad \Pi \vdash \Lambda, G}{\Gamma, \Pi \vdash \Delta, \Lambda, F \wedge G}\ \wedge : r \qquad \frac{F, \Gamma \vdash \Delta}{F \wedge G, \Gamma \vdash \Delta}\ \wedge : l^1 \qquad \frac{G, \Gamma \vdash \Delta}{F \wedge G, \Gamma \vdash \Delta}\ \wedge : l^2$$

$$\frac{\Gamma \vdash \Delta, F \quad G, \Pi \vdash \Lambda}{F \to G, \Gamma, \Pi \vdash \Delta, \Lambda}\ \to : l \qquad \frac{F, \Gamma \vdash \Delta, G}{\Gamma \vdash \Delta, F \to G}\ \to : r$$

[6] In Miller's terminology it is called interchange.

Structural rules:

$$\frac{\Gamma \vdash \Delta, F, F}{\Gamma \vdash \Delta, F}\ \text{contr}\colon r \qquad \frac{F, F, \Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ \text{contr}\colon l$$

$$\frac{\Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ \text{weak}\colon l \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, F}\ \text{weak}\colon r \qquad \frac{\Gamma \vdash \Delta, F \quad F, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda}\ \text{cut}$$

Quantifier rules:

$$\frac{RT, \Gamma \vdash \Delta}{\forall R, \Gamma \vdash \Delta}\ \forall\colon l \qquad \frac{\Gamma \vdash \Delta, RX}{\Gamma \vdash \Delta, \forall R}\ \forall\colon r \qquad \frac{\Gamma \vdash \Delta, RT}{\Gamma \vdash \Delta, \exists R}\ \exists\colon r \qquad \frac{RX, \Gamma \vdash \Delta}{\exists R, \Gamma \vdash \Delta}\ \exists\colon l$$

The eigenvariable X of the $\forall\colon r$ and $\exists\colon l$ rules does not occur in R, Γ and Δ. In contrast to Gentzen's version, the primary and auxiliary formulas need not occur at the end of the antecedent or the beginning of the succedent, removing the need for an exchange rule.

Miller showed the equivalence of LKH to Expansion Proofs[7] which are equivalent to ETT. Since we will formally introduce Expansion Proofs in section 2.3, we will state the corresponding theorems there.

We restrict this presentation of LK to tautological axioms $F \vdash F$ because CERES$^\omega$ relies on the presence of the partner formula in the axiom to infer the Skolemization context. In sections independent of CERES$^\omega$ we will allow arbitrary introduction rules.

Even though higher-order quantifier instantiations do not have the sub-formula property[8]. But even though, strictly speaking, this higher-order sequent calculus is not analytic, the empty sequent $\vdash$ (i.e. falsum) cannot be derived from tautological axioms without cut. If we allow non-tautological introduction rules, we can speak of a refutation as a cut-free derivation of the empty sequent. More precisely, if $\mathcal{S}$ is a set of sequents, then an LK-refutation of $\mathcal{S}$ is an LK-tree π where the end-sequent of π is the empty sequent, and the leaves of π are either tautological axioms $F \vdash F$ or sequents in $\mathcal{S}$. In section 3.10 we will transform first order derivations of such a proof into one with the end-sequent $S_1, \ldots, S_n \vdash$ where $S_1, \ldots, S_n$ are closed formulas corresponding to elements of $\mathcal{S}$. The CERES$^\omega$ method for cut-elimination (see section 2.6) investigated here is also closely related.

For the first-order case earlier work already added binary equality rules [4] to the sequent calculus for the CERES method.

Definition 2.2.3 (Binary Equality Rules (first-order))**.**

$$\frac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \Gamma_2 \vdash F[s], \Delta_2}{\Gamma \vdash F[t], \Delta} =\colon r \qquad \frac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \Gamma_2, F[s] \vdash \Delta_2}{\Gamma, F[t] \vdash \Delta} =\colon r$$

In section 3.2 will discuss the benefits of unary equality rules as an alternative and also add the binary rules to the calculi of the CERES$^\omega$ method (see section 3.1).

[7] Theorems 4.1 and 4.2 in the journal version [72].

[8] For instance, if we infer $\exists X(X(a) \wedge X(b))$ from $(P(a) \to Q) \wedge (P(b) \to Q)$, neither P nor Q appear in the conclusion. Already in first-order logic, we would need to speak of sub-formulas modulo term-instantiation, but in higher-order logic, not even the logical structure needs to be preserved in the conclusion. Nevertheless, the cut rule is the only inference where a premise formula does not have a successor in the conclusion.

2.2.1 Formula Occurrences and Paths

Sometimes we want to distinguish between multiple occurrences of the same formula in a sequent. For instance, when we trace the first occurrence of $\exists x P(x)$ in the end-sequent (marked with $*$) of the proof

$$\dfrac{\dfrac{\dfrac{P(s) \vdash P(s)}{P(s) \vdash \exists x P(x)}\,\exists : r}{P(s) \vdash \exists x P(x)}\,w : r \qquad \dfrac{\dfrac{\dfrac{P(t) \vdash P(t)}{P(t) \vdash \exists x P(x)}\,\exists : r}{P(t) \vdash \exists x P(x)}\,w : r}{P(s) \vee P(t) \vdash \exists x P(x)^*, \exists x P(x)}\,\vee : l$$

to its ancestors, the witness term for it is s. But when we trace the second occurrence, we reach the witness term t. Moreover, if the second occurrence were introduced by weakening, there would be no witness at all. In this text, we can mark an occurrence like above. For the implementation in GAPT we use the following definition:

Definition 2.2.4 (Formula Occurrence)**.** Let ρ be an inference in a proof φ with end-sequent S and let i be the index of a formula F occurring in the antecedent/succedent of S. Then a *formula occurrence* is defined as the quadruple (φ, ρ, ced, i) where $ced \in \{Ant, Succ\}$.

We also make the notion of tracing more explicit by defining ancestors and successors of a formula occurrence which then form upwards and downwards paths.

Definition 2.2.5 (Ancestors and Descendants)**.** In an inference rule, each auxiliary formula is an immediate ancestor of the primary formula. Likewise, each primary formula is the immediate descendant of its rule's auxiliary formulas. The ancestor and successor relations are the reflexive, transitive closure of the immediate ancestor and successor relations.

Definition 2.2.6 (Paths)**.** A sequence $\mu_1, \ldots, \mu_n$ of formula occurrences is called a downwards path, if each μ_i is an immediate ancestor of μ_{i+1} with $1 \leqslant i < n$. It is maximal, if μ_n is a formula in the end-sequent.
A sequence $\mu_1, \ldots, \mu_n$ of formula occurrences is called an upwards path, if each μ_i is an immediate descendant of μ_{i+1} with $1 \leqslant i < n$. It is maximal, if μ_n is an axiom formula.

2.3 Expansion Proofs

Expansion proofs were introduced by Miller [72] as a generalization of Herbrand disjunctions to Elementary Type Theory. Containing (at least) all the necessary quantifier instantiations of a valid formula, they are concise representations for proofs. At the same time, there is a close connection to higher-order sequent calculus defined in section 2.2 and consequently also LK_{skc}, the variant used for CERES$^\omega$2.6.1. The main observation here is that only the quantifier rules of such a calculus are not analytic. The rules for the other logical operators still introduce the conclusion from some of its sub-formulas. Therefore, when reasoning backwards from an

end-sequent formula towards the axioms, the outermost logical operator determines the rule(s)[9] applied until a quantifier instantiation happens. Even then, only the instantiation of a predicate quantifier introduces new logical content. Therefore, the branching structure of the proofs of two instances of a formula agrees up to the outermost quantifiers and may only differ afterwards. In contrast to a Herbrand disjunction, an expansion tree represents this shared structure only once and collects the instantiations as sub-trees of a weak quantifier node. This tree structure can be directly extracted from a sequent calculus proof.

We will first define expansion trees, which do not necessarily correspond to a sequent calculus proof and then introduce the conditions which make them expansion proofs. Since expansion trees are similar to term trees, it is possible to define two mappings from expansion trees to formulas. The shallow mapping leaves quantifiers intact, whereas the deep mapping uses the term annotations to expand quantifier nodes. Depending on the polarity and type of the quantifier, the quantifier is replaced by a conjunction or disjunction of instantiations of the deep formulas of the child nodes, where the annotated term is substituted for the bound variable. The relation between the shallow and deep formula of an expansion tree is similar to a formula and its Herbrand disjunction. In fact, the deep formula is a compressed form of a Herbrand disjunction, where the shared term structure is not repeated.

Definition 2.3.1 (Expansion Tree)**.** An expansion tree is a tree where inner nodes are labeled by logical operators, leaf nodes are labeled by formulas and edges may be labeled by terms. An expansion tree T is inductively defined by the following rules:

- Axiom node: if A is an atom formula or of the form $\forall F$, then T is an axiom node labeled with A. We define $\mathrm{Sh}(T) = \mathrm{Dp}(T) = A$.
- Unary Logical node: if T_1 is an expansion tree, then $T = \neg T_1$, is an expansion tree. We define $\mathrm{Sh}(T) = \neg\mathrm{Sh}(T_1)$ and $\mathrm{Dp}(T) = \neg\mathrm{Dp}(T)$.
- Binary Logical node: if T_1 and T_2 are expansion trees which do not share selected variables, then $T = T_1 \circ T_2$ is an expansion tree, where $\circ \in \{\wedge, \vee, \rightarrow\}$. We define $\mathrm{Sh}(T) = \mathrm{Sh}(T_1) \circ \mathrm{Sh}(T_2)$ and $\mathrm{Dp}(T) = \mathrm{Dp}(T_1) \circ \mathrm{Dp}(T_2)$.
- Strong Quantifier node: Let T_1 be an expansion tree where $\mathrm{Sh}(T_1)$ is equal to the β-normal form of the formula $F(x)$, F itself is also β-normal and x is not selected in T_1. Then $T = Q\ F +^x T_1$ is an expansion tree with $\mathrm{Sh}(T) = Q\ F$ where $Q \in \{\forall, \exists\}$ and $\mathrm{Dp}(T) = \mathrm{Dp}(T_1)$. We call x a *selected variable* in T.
- Weak Quantifier node: Let T_i be expansion trees with $1 \leqslant i \leqslant n$ where each t_i is a β-normal term for which there exists a formula $F(x)$, such that $\mathrm{Sh}(T_i)$ is the β-normal form of $F(t_i)$.

[9] In our version of LK, the $\vee : r$ and $\wedge : l$ are not invertible because they infer $A \mathring{B}$ from only one sub-formula. Therefore, during backwards reasoning we need to first apply contraction and apply the rule on each copy to obtain both possible contributors. If we were using the invertible version with two auxiliary formulas A and B, each operator would directly correspond to one rule application.

Then $T = Q\ F +^{t_1}\ T_1 +\ \ldots +^{t_n}\ T_n$ is an expansion tree with $Q \in \{\forall, \exists\}$. We define $\mathrm{Sh}(T) = Qx\ F(x)$. If $Q = \forall$, we define $\mathrm{Dp}(T) = \mathrm{Dp}(T_1) \wedge \ldots \wedge \mathrm{Dp}(T_n)$. Otherwise, $Q = \exists$ and we define $\mathrm{Dp}(T) = \mathrm{Dp}(T_1) \vee \ldots \vee \mathrm{Dp}(T_n)$. We call $t_1, \ldots, t_n$ the *expansion terms* of T.

The names for strong and weak quantifier nodes already mirror those of the LK rules they correspond to. Similarly to formulas, we can also define positive and negative occurrences of sub-trees. In fact, the polarity of a node T' an expansion tree T coincides with the polarity of the sub-formula $\mathrm{Dp}(T')$ in $\mathrm{Dp}(T)$.

Definition 2.3.2 (Polarity of Expansion Trees)**.** Let T_1, T_2 be expansion trees.

- An axiom tree of an atom A may have positive and negative polarity. An axiom tree of a quantified formula $\forall F$ only occurs in negative polarity.
- If the tree $\neg T_1$ has positive (negative) polarity, then T_1 has negative (positive) polarity.
- If the tree $T_1 \wedge T_2$ resp. $T_1 \vee T_2$ has positive (negative) polarity, then T_1 and T_2 have positive (negative) polarity.
- If the tree $T_1 \to T_2$ has positive (negative) polarity, then T_1 has negative (positive) and T_2 has positive (negative) polarity.
- The tree $Q\ F +^y T_1$ with $Q \in \{\forall, \exists\}$ only occurs in positive polarity. Its sub-tree T_1 also has positive polarity.
- The tree $Q\ F +^{t_1} T_1 + \ldots +^{t_n} T_n$ with $Q \in \{\forall, \exists\}$ only occurs in negative polarity. Its sub-trees $T_1, \ldots, T_n$ also have negative polarity.

Since not every expansion tree represents a proof we need to further analyze the interplay between selected quantifiers. We define $\mathbf{S}_T$ as the set of all selected variables in T and Θ_T as the set of all expansion terms in T. Furthermore we say that a node *dominates* each node of its sub-trees. Then, given an expansion tree T we define the relation $<^0_T$ between terms $s, t \in \Theta_T$ such that $t <^0_T s$ if there exists a variable which is selected for a node dominated by the (weak quantifier) node of t and which is free in s. We call the transitive closure of $<^0_T$ the *dependency relation* $<_T$ of T. Soundness (defined below) together with the dependency relation has a function similar to the eigenvariable condition in strong quantifier rules of LK. The first disallows the occurrence of an eigenvariable in the context and the second ensures that instantiation terms can be ordered in a way that does not invalidate the eigenvariable condition.

Since the definition of an axiom node also allows a formula having a weak quantifier as head we call an expansion tree *grounded* if it does not have an axiom node of this kind. Now we have all the means to define an expansion proof and substitutions for expansion trees:

Definition 2.3.3 (Expansion Proof)**.** An expansion tree T is *sound* if the free variables of $\mathrm{Sh}(T)$ are not selected in T. An expansion tree T is an expansion tree for the formula A if $\mathrm{Sh}(T)$ is the β-normal form of A and T is sound. An expansion tree T where its dependency relation $<_T$ is acyclic and where $\mathrm{Dp}(T)$ is a tautology is called an *expansion proof*.

Definition 2.3.4 (Expansion Tree Substitution)**.** By overloading the notation for terms, we write the substitution σ of an expansion tree T by $\{x_\alpha \leftarrow t_\alpha\}$ provided that x is not selected in T. Then $T\sigma$ is defined by replacing each term t of each node in T with the β-normal form of $t\sigma$. If T is an expansion tree and $\mathrm{Sh}(T\sigma)\downarrow_\beta = \mathrm{Sh}(T\sigma\downarrow_\beta)$ and $\mathrm{Dp}(T\sigma)\downarrow_\beta = \mathrm{Dp}(T\sigma\downarrow_\beta)$ then $T\sigma$ is an expansion tree.

When talking about equivalence of (cut-free) LK proofs to Expansion Proofs it is convenient to extend the notion of sequents to Expansion Trees. To be more descriptive, we use the name expansion sequent for what Miller calls a q-sequent.

Definition 2.3.5 (Expansion Sequent)**.** An *expansion sequent* E is a pair of (possibly empty) lists of expansion trees. We will write it as $S_1, \ldots, S_n \vdash_{ET} T_1, \ldots, T_m$ where $S_1, \ldots, S_n$ are expansion trees of negative polarity and $T_1, \ldots, T_m$ are expansion trees of positive polarity. An expansion sequent is equivalent to the expansion tree $S_1 \wedge \ldots \wedge S_n \to T_1 \vee \ldots \vee T_m$. The deep sequent of E is the sequent $\mathrm{Dp}S_1, \ldots, \mathrm{Dp}S_n \vdash \mathrm{Dp}T_1, \ldots, \mathrm{Dp}T_m$.

Through the interpretation as an expansion tree the notions of soundness, acyclicity, substitution and proof may also be used for expansion sequents. We can now state the equivalence results to LK [72]:

Theorem 2.3.6. If A is a formula with a grounded expansion proof, then there exists an LK proof of the sequent $\vdash A$.

Theorem 2.3.7. If the sequent $\Gamma \vdash \Delta$ has an LK proof, there is a grounded expansion sequent $\Lambda \vdash \Pi$ such that $\Gamma = \mathrm{Sh}(\Lambda)$ and $\Delta = Sh(\Pi)$. Thus, if A has an LK proof, it has a grounded expansion proof.

The reason for requiring a grounded expansion proof in theorem 2.3.6 is that a corresponding LK proof can be constructed without proof search. Since by definition the deep formula of an expansion proof is valid, it is always possible to find the necessary instantiations. Besides, theorem 2.3.7 is more relevant to our analysis process since our main interest lies in the extraction of (Skolem) expansion proofs from LK proofs. For example, the LK proof π

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(z,a) \to P(z,b) \vdash P(z,a) \to P(z,b)}{(\forall X(X(a) \to X(b))) \vdash (P(z,a) \to P(z,b))}\,\forall : l \quad \dfrac{P(a,z) \to P(b,z) \vdash P(a,z) \to P(b,z)}{(\forall X(X(a) \to X(b))) \vdash P(a,z) \to P(b,z)}\,\forall : l}{(\forall X(X(a) \to X(b))), (\forall X(X(a) \to X(b))) \vdash (P(z,a) \to P(z,b)) \wedge (P(a,z) \to P(b,z))}\,\wedge : r}{(\forall X(X(a) \to X(b))) \vdash (P(z,a) \to P(z,b)) \wedge (P(a,z) \to P(b,z))}\,c : l}{(\forall X(X(a) \to X(b))) \vdash (\forall y(P(z,a) \to P(y,b)) \wedge (P(a,y) \to P(b,y)))}\,\forall : r$$

$$(\pi)$$

is equivalent to the expansion sequent $T_1 \vdash T_2$ with

$$\begin{aligned} T_1 &= \forall X(X(a) \to X(b)) \\ &\quad +^{\lambda x P(x,z)}(P(a,z) \to P(b,z)) \\ &\quad +^{\lambda x P(z,x)}(P(z,a) \to P(z,b)) \end{aligned}$$

$$\begin{aligned} T_2 &= \forall y((P(y,a) \to P(y,b)) \wedge (P(a,y) \to P(b,y))) \\ &\quad +^{z}((P(z,a) \to P(z,b)) \to (P(a,z) \to P(b,z))) \end{aligned}$$

which has the deep formulas $\mathrm{Dp}(T_1) = \mathrm{Dp}(T_2) = (P(z,a) \to P(z,b)) \wedge (P(a,z) \to P(b,z))$, making $\mathrm{Dp}(T_1) \to \mathrm{Dp}(T_2)$ true. The instantiation terms $\lambda x P(x,z)$ and $\lambda x P(z,x)$ in T_1 have one free variable, but it is not selected in any of its sub-node. Therefore, the dependency relation is trivially acyclic, which makes $T_1 \vdash T_2$ an expansion proof.

2.3.1 Extracting Expansion Proofs from LK

We will now describe the extraction algorithm for expansion trees from LK proofs used by Miller to prove theorem 2.3.6 in more detail. In section 3.5 we will extend this algorithm to provide expansion proofs from LK_{skc} proofs with propositional cuts and so called passive cuts, where ancestors of the cut-formula never occur as the primary formula of an inference.

The logical rules can be directly represented by introducing a corresponding expansion tree node but contraction is a bit more challenging. In principle we are given two expansion trees T_a, T_b which have the same shallow formula $\mathrm{Sh}(T_a) = \mathrm{Sh}(T_b)$ but might differ on the deep formula. Since the two trees structurally agree on their shallow part, the merge operation joins the quantifier instantiations in the respective nodes.

Definition 2.3.8 (Merge of expansion trees)**.** Let T_a and T_b be expansion trees with $Sh(T_a)$ is α-equal to $Sh(T_b)$. We define $merge(T_a, T_b)$ by distinguishing (T_a, T_b) on their (identical) root node:

- Atom: then both atoms are identical and $merge(T_a, T_b) = T_a$
- Negation $(\neg T'_a, \neg T'_b)$: then $merge(T_a, T_b) = \neg merge(T'_a, T'_b)$.
- Binary Connective $(T_{1l} \circ T_{1r}, T_{2l} \circ T_{2r})$: then $merge(T_a, T_b) = merge(T_{1l}, T_{2l}) \circ merge(T_{1r}, T_{2r})$
- Strong Quantifier $(Q\ F +^x T'_a, Q\ F +^x T'_b)$: then $merge(T_a, T_b) = Q\ F +^x (merge(T'_a, T'_b))$ with $Q \in \{\forall, \exists\}$.
- Weak Quantifier $(Q\ F +^{t_{a1}} T_{a1} + \ldots +^{t_{an}} T_{an}, Q\ F +^{t_{b1}} T_{b1} + \ldots +^{t_{bm}} T_{bm})$: then $merge(T_a, T_b) = Q\ F +^{t_{a1}} T_{a1} + \ldots +^{t_{an}} T_{an} +^{t_{b1}} T_{b1} + \ldots +^{t_{bm}} T_{bm}$ with $Q \in \{\forall, \exists\}$.

Having the merge operation available, we can now sketch the extraction of an expansion sequent from an LK proof: given the expansion sequent $\Lambda \vdash T_A, T'_A, \Pi$ for the proof π in the contraction inference

$$\frac{\begin{array}{c}(\pi)\\ \Gamma \vdash A, A, \Delta\end{array}}{\Gamma \vdash A, \Delta}\, c : r$$

then we use the expansion sequent $\Lambda \vdash merge(T_A, T'_A), \Pi$. Furthermore, given the expansion sequent $\Lambda \vdash \Pi$ for the proof π in weakening inference

$$\frac{\begin{array}{c}(\pi)\\ \Gamma \vdash \Delta\end{array}}{\Gamma \vdash A, \Delta}\, w : r$$

and any expansion tree[10] T with $\mathrm{Sh}(T) = A$, then we use the expansion sequent $\Lambda \vdash \Pi$.

The handling of logical rules all work in a similar fashion: suppose $\Lambda \vdash T_A, T_B, \Pi$ is the expansion sequent corresponding to the sub-proof π in the inference

$$\frac{\begin{array}{c}(\pi)\\ \Gamma \vdash A, \Delta\end{array}}{\Gamma \vdash A \vee B, \Delta}\, \vee : r$$

then we use $\Lambda \vdash T_A \vee T_B, \Pi$ as the expansion sequent for the conclusion. Likewise, given the expansion sequent $\Lambda_1 \vdash T_A, \Pi_2$ for the sub-proof π_1 and the expansion sequent $\Lambda_2 \vdash T_B, \Pi_2$ for the sub-proof π_2 in the inference

$$\frac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \Gamma_1 \vdash A, \Delta_1 & \Gamma_2 \vdash B, \Delta_2\end{array}}{\Gamma \vdash A \wedge B, \Delta}\, \wedge : r$$

then we use $\Lambda_1, \Lambda_2 \vdash T_A \wedge T_B, \Pi_2$ as the expansion sequent for the conclusion. In all cases, if the parent sequents are grounded expansion proofs, so is the inferred expansion sequent.

Now we turn our focus to soundness and completeness results relative to ETT. Miller provides both results as theorem 2.4.37 and 2.5.42 of his PhD thesis [71]:

Theorem 2.3.9. If the formula A has an expansion proof, then $\vdash_{\mathcal{T}} A$.

Theorem 2.3.10. If A is a formula such that $\vdash_{\mathcal{T}} A$, then A has a grounded expansion proof.

[10] There is always an expansion tree reflecting the shallow structure of the formula with quantifiers formulas at the leaves.

2.3.2 Skolem Expansion Proofs

In his PhD thesis [71], Miller also describes a variant of expansion proofs which replaces strong quantifier nodes containing selected variables with skolem quantifier nodes containing skolem terms (as defined in section 2.1.2).

Definition 2.3.11. A Skolem expansion tree is defined like an expansion tree except as follows: Instead of a strong quantifier node $Q\ F +^x T$ we use a Skolem quantifier node $Q\ F +^{ft_1 \dots t_p} T$ where $ft_1 \dots t_p \in \mathcal{U}$ is a skolem term with the skolem function f of arity p.

Each skolem expansion tree must fulfill two global requirements:

- Each skolem quantifier node introduces a unique skolem function f.
- The path from the root to a skolem quantifier node contains exactly p weak quantifier nodes with expansion terms t_1 to t_p (in that order).

The definitions of the shallow formula, the deep formula and skolem expansions proof property are adapted accordingly.

The global restrictions are necessary to preserve soundness. For instance, the axiom of choice applied to functions, written as $\forall x_\iota \exists y_\iota P(x,y) \rightarrow \exists f_{\iota > \iota} \forall z_\iota P(z, f(z))$, suddenly becomes the provable formula $\forall x P(x, g(x)) \rightarrow \exists f P(h(f), f(h(f)))$ after skolemization: instantiating f with $\lambda x\ g(x)$ and x with $h(\lambda x\ g(x))$ leads to $P(h(\lambda x\ g(x)), g(h(\lambda x\ g(x)))) \rightarrow P(h(\lambda x\ g(x)), g(h(\lambda x\ g(x))))$. The latter is clearly a tautology but the axiom of choice is not.

Miller shows [72] the following equivalence results (as theorem 6.8 and 6.12 in the paper):

Theorem 2.3.12. If A has an expansion proof, it has a skolem expansion proof.

Theorem 2.3.13. If A has a skolem expansion proof, it has an expansion proof.

2.4 Reductive Cut-Elimination

The traditional cut-elimination methods defined by Gentzen [39] and Schütte / Tait [103] can both be characterized as rewrite systems on LK proofs. As a general term for cut-elimination by step-wise proof rewriting, Baaz and Leitsch introduced the term *reductive cut-elimination* [61, p.105]. Since CERES$^\omega$ first converts a proof with arbitrary cuts into one on passive cuts, where no rule operates on the cut formula and then uses the Gentzen method to remove those passive cuts entirely, we will repeat the rewrite rules here. We distinguish two kinds of rules, those which reduce the grade – that is the number of connectives – of the upper-most cut formula and those which reduce the rank – that is the longest distance of the cut formula to one of the axioms introducing it – of the cut inference.

Definition 2.4.1 (Grade and Rank)**.** Let ρ be a cut-inference on the formula C of the form

$$\dfrac{\begin{array}{cc}(\varphi_1) & (\varphi_2) \\ \Gamma_1 \vdash \Delta_1 & \Gamma_2 \vdash \Delta_2\end{array}}{\Gamma_1, \Gamma_2^* \vdash \Delta_1^*, \Delta_1}\ cut$$
$$(\pi)$$

Then $grade(\rho) = compl(A)$ with

$$compl(A) = \begin{cases} 1 & \text{if } C \text{ is an atom} \\ 1 + compl(A) & \text{if } C \text{ is of the form } \neg A, \forall x.A, \exists x.A \\ 1 + max(compl(A), compl(B)) & \text{if } C \text{ is of the form } A \wedge B, A \vee B, A \to B \end{cases}$$

.

Let L (R) be the set of upwards paths of occurrences of C in Γ_2^* (Δ_1^*) in the end-sequent of π. Then $rank(\rho) = 2 + max(\{len(l)|l \in L\}) + max(\{len(r)|r \in R\})$ where $len(p)$ denotes the length of the path p.

To show termination, Gentzen introduced a macro-rule called mix which integrates contraction inferences:

Definition 2.4.2 (Mix rule)**.**
The mix rule on the formula A has at least one auxiliary occurrence of A in the succedent(antecedent) of the left(right) parent occurrence and no primary formula. It is translated to the following pattern ending in a cut:

$$\frac{\Gamma_1 \vdash A, \ldots, A, \Delta_1 \qquad \Gamma_2, A, \ldots, A \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,\text{mix} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 \vdash A, \ldots, A, \Delta_1}{\Gamma_1 \vdash A, \Delta_1}\,c:r \qquad \dfrac{\Gamma_2, A, \ldots, A \vdash \Delta_2}{\Gamma_2, A \vdash \Delta_2}\,c:l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,cut$$

To show termination with cut instead of mix, the traditional lexical ordering on the pair $(grade(\pi), rank(\pi))$ needs to ignore the sequence of contractions immediately preceding the cut rule for computing the rank of a cut-formula [110]. Apart from the translation to cut, we follow the presentation of Borisavljević [19] and distinguish between trivial cases, rank reduction, grade reduction and cuts on contracted formulas:

Definition 2.4.3 (Reductive cut-elimination)**.**

1. The cut formula appears in the antecedent of the left parent proof:

$$\frac{\dfrac{}{\Gamma_1, A \vdash A, \Delta_1}(\pi_1) \qquad \dfrac{}{\Gamma_2, A \vdash \Delta_2}(\pi_2)}{\Gamma_1, \Gamma_2, A \vdash \Delta_1, \Delta_2}\,cut \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{}{\Gamma_2, A \vdash \Delta_2}(\pi_2)}{\Gamma_1, \Gamma_2, A \vdash \Delta_2}\,w:l}{\Gamma_1, \Gamma_2, A \vdash \Delta_1, \Delta_2}\,w:r$$

   This case covers the introduction rule $A \vdash A$, where Γ_1 and Δ_1 are empty. If $A \vdash A$ appears as a subsequent in the right parent, the situation is symmetrical.

2. The cut formula is introduced by weakening:

$$\frac{\dfrac{}{\Gamma_1 \vdash \Delta, F}(\pi_1) \qquad \dfrac{\dfrac{}{\Gamma_2 \vdash \Delta_2}(\pi_2)}{F, \Gamma_2 \vdash \Delta_2}\,w:l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,cut \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash \Delta_2}(\pi_2)}{\Gamma_1, \Gamma_2 \vdash \Delta_2}\,w:l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,w:r$$

3. Grade reduction:
   All cases assume that the cut formula is the primary formula in both parent inferences. We distinguish on the kind of this rule:

a) The cut formula is introduced by negation:

$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1, A \vdash \Delta_1}}{\Gamma_1 \vdash \Delta_1, \neg A}\neg : r \quad \dfrac{\dfrac{(\pi_2)}{\Gamma_2 \vdash A, \Delta_2}}{\Gamma_2, \neg A \vdash \Delta_2}\neg : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut \quad \rightsquigarrow \quad \dfrac{\dfrac{(\pi_1)}{\Gamma_2 \vdash \Delta_2, A} \quad \dfrac{(\pi_2)}{\Gamma_1, A \vdash \Delta_1}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

b) The cut formula is introduced by conjunction:

$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash \Delta_2, B}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B}\wedge : r \quad \dfrac{\dfrac{(\pi_3)}{\Gamma_3, A \vdash \Delta_3}}{\Gamma_3, A \wedge B \vdash \Delta_3}\wedge : l}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Gamma_3}\, cut \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_3)}{\Gamma_3, A \vdash \Delta_3}}{\Gamma_1, \Gamma_3 \vdash \Delta_1, \Delta_3}\, cut}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3}\, w : *$$

c) The cut formula is introduced by disjunction/implication:
Similar to the conjunction/negation case.

d) The cut formula is introduced by universal quantification:

$$\dfrac{\dfrac{\dfrac{(\pi_1(\alpha))}{\Gamma_1 \vdash \Delta_1, P(\alpha)}}{\Gamma_1 \vdash \Delta_1, \forall x P(x)}\forall : r \quad \dfrac{\dfrac{(\pi_2)}{\Gamma_2, P(t) \vdash \Delta_2}}{\Gamma_2, \forall x P(x) \vdash \Delta_2}\forall : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut \quad \rightsquigarrow \quad \dfrac{\dfrac{(\pi_1(t))}{\Gamma_1 \vdash \Delta_1, P(t)} \quad \dfrac{(\pi_2)}{\Gamma_2, P(t) \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

The proof $\pi(t)$ is obtained by applying the substitution $\{\alpha \leftarrow t\}$ to $\pi(\alpha)$. The substitution leaves context unchanged because it does not contain the eigenvariable α.

e) The cut formula is introduced by existential quantification:
Analogous to the universal case.

4. Right rank reduction: These cases handle the situation where the right parent inference does not have the cut formula as primary formula. We permute the cut upwards by distinguising unary and binary inferences. The contraction case is handled seperately.

a) The right parent inference is a unary rule, but not contraction:

$$\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{\dfrac{(\pi_2)}{\Gamma_2 \vdash A, \Delta_2}}{\Gamma_2' \vdash A, \Delta_2'}\rho}{\Gamma_1, \Gamma_2' \vdash \Delta_1, \Delta_2'}\, cut \quad \rightsquigarrow \quad \dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash A, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut}{\Gamma_1, \Gamma_2' \vdash \Delta_1, \Delta_2'}\,\rho$$

b) The right parent inference is a binary rule:

$$\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{\dfrac{(\pi_2)}{A, \Gamma_2 \vdash \Delta_2} \quad \dfrac{(\pi_3)}{\Gamma_3 \vdash \Delta_3}}{A, \Gamma_2', \Gamma_3' \vdash \Delta_2', \Delta_3'}\rho}{\Gamma_1, \Gamma_2', \Gamma_3' \vdash \Delta_1, \Delta_2', \Delta_3'}\, cut \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{A, \Gamma_2 \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut \quad \dfrac{(\pi_3)}{\Gamma_3 \vdash \Delta_3}}{\Gamma_1, \Gamma_2', \Gamma_3' \vdash \Delta_1, \Delta_2', \Delta_3'}\,\rho$$

5. Left rank reduction
This is symmetrical to the right reduction. Gentzen's version always performs right rank reduction before left rank reduction, but the actual rules do not depend on that. In any case, grade reduction only happens when rank reduction is not applicable anymore.

6. The parent inferences is a series of contractions:
In the most general case, there appears a series of contractions after the introduction rules for the outermost connective of the cut formula on both parent branches of the cut-rule. Like in grade reduction, we distinguish on the outermost symbol of the cut formula and reduce the number of contraction inferences of both parents by one. To make the reduction better visible, we denote n occurrences of a formula F as F^n:

a) The cut-formula is a conjunction inference $F = A \wedge B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F^b, B, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash F^{a+b}, A \wedge B, \Delta_1} \wedge : r}{\Gamma_1, \Gamma_2 \vdash F, A \wedge B, \Delta_1, \Delta_2} (a+b-1) \times c : r}{\Gamma_1, \Gamma_2 \vdash F, \Delta_1, \Delta_2} c : r \qquad \dfrac{\dfrac{\dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A \wedge B, F^n \vdash \Delta_3} \wedge : l}{\Gamma_3, A \wedge B, F \vdash \Delta_3} (n-1) \times c : l}{\Gamma_3, F \vdash \Delta_3} c : l$$
$$\dfrac{\Gamma_1, \Gamma_2 \vdash F, \Delta_1, \Delta_2 \qquad \Gamma_3, F \vdash \Delta_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3} cut$$

This proof contains $a + b + n$ contractions between the conjunction inferences and the cut. In principle, we use the same arguments as for rank reduction, but also permute them over the contractions. Since π_3 contains A instead of B, the number b of occurrences of F in the end-sequent of π_2 is irrelevant, in the symmetrical case this holds for the number a of occurrences in the end-sequent of π_1.[11] We distinguish on the lengths a and n of the contraction sequences in the parent proofs of the cut:

- $a = 0, n = 0$: This is the grade reduction case 3 without contractions.
- $a = 0, n > 0$: Since there are no additional occurrences of F in the succedent of the end-sequent of π_1, we can directly cut with π_3 and contract later:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash A, \Delta_1} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F^b, B, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash F^b, A \wedge B, \Delta_1} \wedge : r}{\Gamma_1, \Gamma_2 \vdash F, \Delta_1, \Delta_2} b \times c : r \qquad \dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash A, \Delta_1} \quad \dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_1, \Gamma_3, F^n \vdash \Delta_1, \Delta_3} cut}{\Gamma_1, \Gamma_3, F \vdash \Delta_1, \Delta_3} (n-1) \times c : r$$
$$\dfrac{\Gamma_1, \Gamma_2 \vdash F, \Delta_1, \Delta_2 \qquad \Gamma_3, F \vdash \Delta_1, \Delta_3}{\dfrac{\Gamma_1, \Gamma_3, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_3, \Delta_1, \Delta_2}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3} c : l + c : r} cut$$

  We now have $b + n - 1$ contraction inferences but needed to duplicate the cut.
- $a > 0, n = 0$: This case is symmetrical to the preceding one:

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1} \quad \dfrac{(\pi_3)}{\Gamma_3, A \vdash \Delta_3}}{\Gamma_1, \Gamma_3 \vdash F^a, \Delta_1, \Delta_2} cut}{\Gamma_1, \Gamma_3 \vdash F, \Delta_1, \Delta_3} a \times c : r \qquad \dfrac{\dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A \wedge B, F^n \vdash \Delta_3} \wedge : l}{\Gamma_3, F \vdash \Delta_3} n \times c : l}{\dfrac{\Gamma_1, \Gamma_3, \Gamma_3 \vdash \Delta_1, \Delta_3, \Delta_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3} c : l + r, w : l + r} cut$$

- $a > 0, n > 0$ uses:

$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1}}{\Gamma_1 \vdash F, A, \Delta_1} (a-1) \times c : r \qquad \dfrac{\dfrac{\dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A \wedge B, F^n \vdash \Delta_3} \wedge : l}{\Gamma_3, A \wedge B, F \vdash \Delta_3} (n-1) \times c : l}{\Gamma_3, F \vdash \Delta_3} c : l}{\Gamma_1, \Gamma_3 \vdash A, \Delta_1, \Delta_3} cut$$
$$(\pi_4)$$

[11] To make the contraction clearer, we make the last contraction explicit. In the cases of $a + b = 0$ and $n = 0$, this means that we also need to skip the explicit step.

and

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F^b, B, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash F^{a+b}, A \wedge B, \Delta_1}\wedge : r}{\Gamma_1, \Gamma_2 \vdash F, A \wedge B, \Delta_1, \Delta_2}(a+b-1) \times c : r \quad \dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A, F \vdash \Delta_3}(n-1) \times c : l}{\Gamma_1, \Gamma_2, \Gamma_3, A \vdash \Delta_1, \Delta_2, \Delta_3}\, cut$$

(π_5)

which are combined again by cut:

$$\dfrac{\dfrac{\dfrac{(\pi_4)}{\Gamma_1, \Gamma_3 \vdash A, \Delta_1, \Delta_3} \quad \dfrac{(\pi_5)}{\Gamma_1, \Gamma_2, \Gamma_3, A \vdash \Delta_1, \Delta_2, \Delta_3}}{\Gamma_1, \Gamma_3, \Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_3, \Delta_1, \Delta_2, \Delta_3}\, cut}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3}\, c : l + c : r$$

The proof π_4 contains $a + n - 1$ contraction inferences whereas the proof π_5 contains $a + b + n - 1$ contraction inferences. Applying cut-elimination to the parent proofs eliminates the contractions before the cut-formula in the final cut, such that cut-elimination also terminates in this case.

b) The cases of disjunction and implication work analogously to the conjunction case.

c) The cut-formula is a quantification $\forall x F(x)$: The pattern looks as follows:

$$\dfrac{\dfrac{\dfrac{(\pi_1(\alpha))}{\Gamma_1 \vdash (\forall x F(x))^n, F(\alpha), \Gamma_2}\forall : r}{\Gamma_1 \vdash (\forall x F(x))^n, \forall x F(x), \Gamma_2}\, n \times c : r \quad \dfrac{\dfrac{(\pi_2)}{\Gamma_2, (\forall x F(x))^m, F(t) \vdash \Delta_2}\forall : l}{\Gamma_2, (\forall x F(x))^m, \forall x F(x) \vdash \Delta_2}\, m \times c : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

We only describe the most complicated case with $n > 0, m > 0$, the others work accordingly. Again we create two proofs, skipping the $\forall : r$ respectively the $\forall : l$ rule to obtain

$$\dfrac{\dfrac{\dfrac{(\pi_1(t))}{\Gamma_1 \vdash (\forall x F(x))^n, F(\alpha), \Gamma_2}}{\Gamma_1 \vdash \forall x F(x), F(t) \Gamma_2}(n-1) \times c : r \quad \dfrac{\dfrac{(\pi_2)}{\Gamma_2, (\forall x F(x))^m, F(t) \vdash \Delta_2}\forall : l}{\Gamma_2, (\forall x F(x))^m, \forall x F(x) \vdash \Delta_2}\, m \times c : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, F(t)}\, cut$$

(π_3)

where $\pi_1(t)$ is the instance of $\pi(\alpha)$ and

$$\dfrac{\dfrac{\dfrac{(\pi_1(\alpha))}{\Gamma_1 \vdash (\forall x F(x))^n, F(\alpha), \Gamma_2}\,\forall : r}{\Gamma_1 \vdash (\forall x F(x))^n, \forall x F(x), \Gamma_2}\,n \times c : r \qquad \dfrac{\dfrac{(\pi_2)}{\Gamma_2, (\forall x F(x))^m, F(t) \vdash \Delta_2}\,(m-1) \times c : l}{\Gamma_2, \forall x F(x), F(t) \vdash \Delta_2}}{\Gamma_1, \Gamma_2, F(t) \vdash \Delta_1, \Delta_2}\,cut$$
(π_4)

Both proofs have $n + m - 1$ contractions leading into the cut which can be successively removed until they vanish completely and reduce to the grade reduction case. They can be combined as follows:

$$\dfrac{\dfrac{(\pi_3)}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, F(t)} \qquad \dfrac{(\pi_4)}{\Gamma_1, \Gamma_2, F(t) \vdash \Delta_1, \Delta_2}\,cut}{\Gamma_1, \Gamma_2, \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, \Delta_1, \Delta_2}\,c : l + r$$
$$\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$$

d) The cut-formula is one of the other unary logical rules or weakening: The is essentially the same pattern as the case for universal quantification.

The reduction rule 1 is the only one where a cut-formula becomes an end-sequent ancestor. This becomes problematic in one of the post-processing steps of the CERES method. Instead of immediately discarding one parent proof, we can permute the cut upwards via rank-reduction and eliminate it at its introduction. This is either a weakening rule which is covered in case 2 or an axiom rule which is covered by the following reduction:

Definition 2.4.4. *Modified reductive cut-elimination* is the rewrite system obtained by taking all rules from definition 2.4.3 apart from rule 1, 3d and 3e. Furthermore we add the rule:

1a. $$\dfrac{A \vdash A \qquad A \vdash A}{A \vdash A}\,cut \quad \rightsquigarrow \quad A \vdash A$$

2.5 CERES

Reductive cut-elimination – that is cut-elimination based on quantifier shifting like the methods of Gentzen [39] or Schütte / Tait [103] – may produce multiple cut-free proofs, depending on the order in which the cut rules are shifted. While these methods apply locally, the CERES method [7, 8, 61] restructures the proof globally. The proof is first reduced to a set of proof projections, which are cut-free proofs of a combination of the original end-sequent with additional atom formulas. These formulas can also be interpreted as clauses and are extracted from the axiom rules of the input proof which contribute to the proofs of cut formulas. From the construction of this characteristic clause set, we know it is contradictory. In particular, it has a resolution refutation which can be simulated in sequent calculus with the projections taking the roles of the clauses. Where the resolution refutation leads to the empty clause, the simulation removes the additional clauses from the end-sequent, resulting in a proof of solely the original end-sequent. The flexibility of the CERES method comes from its dependence on the form of the refutation, which produces proofs reductive methods cannot achieve. Even a propositional proof

showing double negation elimination by using the law of excluded middle (see also Reis' work on CERES in intuitionistic logic [80]) exhibits this phenomenon:

$$\dfrac{\dfrac{\dfrac{P \vdash P}{\neg P, P \vdash}\,\neg:l}{\dfrac{P \vdash P \quad \neg P \vdash \neg P}{P \vee \neg P \vdash P, \neg P}\,\vee:l}\,\neg:r}{P \vee \neg P \vdash P \vee \neg P}\,\vee:r \qquad \dfrac{\dfrac{\dfrac{P \vdash P}{\neg P, P \vdash}\,\neg:l}{\dfrac{\neg P \vdash \neg P}{\neg\neg P, \neg P \vdash}\,\neg:l}\,\neg:r}{P \vee \neg P, \neg\neg P \vdash P}\,\vee:l \quad \dfrac{}{P \vee \neg P \vdash \neg\neg P \to P}\,{\to}:r$$
$$P \vee \neg P \vdash \neg\neg P \to P \; cut$$

Having the law of excluded middle in the end-sequent allows to formulate it as an intuitionistic proof. Using Gentzen's cut-elimination, the trivial cut on the law of excluded middle vanishes, again resulting in an intuitionistic proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{P \vdash P}{\neg P, P \vdash}\,\neg:l}{\neg P \vdash \neg P}\,\neg:r}{\neg(\neg P), \neg P \vdash}\,\neg:l \quad P \vdash P}{P \vee (\neg P), \neg(\neg P) \vdash P}\,\vee:l}{P \vee (\neg P) \vdash (\neg(\neg P)) \to P}\,{\to}:r$$

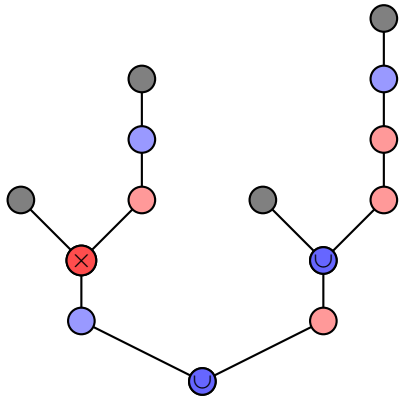
But the CERES method produces the following purely classical proof, where the law of excluded middle is just added by weakening:

$$\dfrac{\dfrac{\dfrac{P \vdash P}{\vdash P, \neg P}\,\neg:r}{\neg\neg P \vdash P}\,\neg:l}{\vdash \neg\neg P \to P}\,{\to}:r}{P \vee \neg P \vdash \neg\neg P \to P}\,w:l$$

Since the Gentzen method always preserves intuitionistic proofs, the one produced by CERES is genuinely different. Let us explore how this proof was produced. The first observation is that the differentiation into unary ($\vee : r$, $\wedge : l$, $\to\!: r$) and binary logical rules ($\wedge : r$, $\vee : l$, $\to\!: l$) characterizes them as either disjunctive or conjunctive. When writing the sequents of the $\wedge : l$ inference as formulas (see definition 2.2.2), it is obvious that $\bigwedge \Gamma \wedge A \wedge B \to \bigvee \Delta$ is equivalent to $\neg \bigvee \Gamma \vee \neg A \vee B \bigvee \Delta$ which exhibits the disjunctive nature of the rule. The binary cut-rule also fits into the characterisation as conjunctive, since it can be seen as $\neg \bigvee \Gamma_1 \vee \bigvee \Delta_1 \vee (\neg F \wedge F) \vee \neg \bigvee \Gamma_2 \vee \bigvee \Delta_2$. We can focus on the branching nature of our input proof by drawing graph with each node representing a sequent and each edge connecting child- and parent proofs.

Since our task is the elimination of the cut rule we generate a set of projections from the input proof by leaving out all inferences which prove a cut formula. In case of a binary rule the question is how to combine the projections of the parent proof. For sure it is related to the fact

whether the main formula contributes to a cut-formula or not. In the first case we color a node in our graph blue, in the second case we color the node red, leading to the following graph for our example:



This graph gives us the template to construct the set of projections mentioned above: inferences of red nodes are necessary to prove the original end-sequent and are preserved in a projection. Blue nodes – in particular cuts – are left out, leading to additional atoms in the end-sequent. In case of binary rules, we create the union of the parent projection sets. In our example, the projections are the following:

$$\dfrac{\dfrac{P \vdash P \qquad \dfrac{P \vdash P}{\neg P, P \vdash}\,\neg : l}{P \vee \neg P, P \vdash P}\,\vee : l}{P \vee \neg P, P \vdash P, \neg(\neg P) \to P}\,w : r \qquad (\pi_1)$$

$$\dfrac{\dfrac{\dfrac{P \vdash P}{\neg(\neg P),\ P \vdash P}\,w : l}{P \vdash \neg(\neg P) \to P}\,{\to}: r}{P \vee (\neg P),\ P \vdash \neg(\neg P) \to P}\,w : l \qquad (\pi_2)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P \vdash P}{\vdash P, \neg P}\,\neg : r}{\neg(\neg P) \vdash P}\,\neg : l}{\neg(\neg P) \vdash P,\ P}\,w : r}{\vdash P, \neg(\neg P) \to P}\,{\to}: r}{P \vee (\neg P) \vdash P, \neg(\neg P) \to P}\,w : l \qquad (\pi_3)$$

The corresponding characteristic sets of additional sub-sequents is $CCS = \{P \vdash P;\ P \vdash ;\vdash\ P\}$. When we remember that we created the union of projections when a binary rule contributes to a cut-formula, it is evident its conjunctive nature reflects also in the interpretation of the set as conjunction of the contained sequents. Similarly the skipped unary inferences' disjunctive nature is reflected in the (classical) interpretation of a sequent as $\bigvee \Gamma \vee \bigvee \Delta$. Under the condition that axiom rules introduce only atomic formulas, the set CCS can be seen as a clause set – the so called characteristic clause set.

As mentioned above, the characteristic clause set always has a refutation, in the case of the example it is the following:

$$\frac{P \vdash \qquad \vdash P}{\vdash} \; Res$$

Since ground resolution corresponds to the cut on an atom formula the resolution proof serves as template to remove the additional content from the projections:

$$\dfrac{\dfrac{\dfrac{\dfrac{P \vdash P}{\neg(\neg P),\ P \vdash P}\, w:l}{P \vdash (\neg(\neg P)) \to P}\, \to:r}{P \vee (\neg P),\ P \vdash (\neg(\neg P)) \to P}\, w:l \qquad \dfrac{\dfrac{\dfrac{\dfrac{P \vdash P}{\vdash P, \neg P}\, \neg:r}{\neg(\neg P) \vdash P}\, \neg:l}{\neg(\neg P) \vdash P,\ P}\, w:r}{\vdash P, (\neg(\neg P)) \to P}\, \to:r}{P \vee (\neg P) \vdash P, \neg(\neg P) \to P}\, w:l$$

$$\dfrac{P \vee (\neg P), P \vee (\neg P) \vdash \neg(\neg P) \to P, \neg(\neg P) \to P}{P \vee (\neg P) \vdash \neg(\neg P) \to P, \neg(\neg P) \to P}\, c:l \qquad cut$$

$$\dfrac{P \vee (\neg P) \vdash \neg(\neg P) \to P, \neg(\neg P) \to P}{P \vee (\neg P) \vdash \neg(\neg P) \to P}\, c:r$$

We now have effectively reduced a proof with arbitrary cut formulas to a proof with atomic cut formulas, its so called atomic-cut normal-form (ACNF). To obtain a cut-free proof we can use the Gentzen method where only rank reductions are necessary since the cut formulas always have grade 1.

To generalize the method to first-order logic, we also need to consider the quantifier rules. Disregarding the quantifier constant itself, a formula quantified over is a generalization of its auxiliary formula. In other words each formula $\forall \lambda x F$ introduced is inferred from the formula $(\lambda x F)\alpha$ for an eigenvariable α (in the case of a strong quantifier inference) or $(\lambda x F)t$ for any term t (in the case of a weak quantifier inference). The refutation of the characteristic sequent set then completes the necessary term instantiations which again can be different from those the Gentzen method generates. There are two requirements which need to be fulfilled for this to work: the proof must be regular i.e. the eigenvariables must be globally unique. Otherwise the instantiation terms of different proof branches are implicitly unified, possibly leading to a satisfiable CCS. Furthermore a projection may have more free variables than the original sequent, which could invalidate the eigenvariable conditions of strong quantifier inferences on ancestors of end-sequent formulas. The solution there is to perform proof Skolemization as a preprocessing step [6]. This is unproblematic in first-order logic since De-Skolemization is always possible.

Apart from the richer number of cut-free normal-forms, which have mostly proof-theoretic value, CERES provides additional value when taking equality rules into account. Right now we describe the binary equality rules from defintion 2.2.3 as they are the historically evolved ones – section 3.2 discusses the (dis-)advantages of the simpler unary rules. In principle, the binary equality rules are treated like a cut: the rule does not contribute to a projection in case the primary formula $F[t]$ is a cut-ancestor and it contributes to the projection otherwise. Now the Gentzen method can not permute a cut with an equality rule where the primary formula is a cut-ancestor. It is possible to move equality rules towards the leaves such that cuts can move upwards too, but the cuts cannot be completely removed leaving the proof in atomic-cut normal-form. Moreover,

the up-shifting of equality rules causes an unwanted rewriting of the proof modulo the theory defined by the equality $s = t$ which complicates the interpretation of this ACNF. Contrary to reductive methods, CERES simulates paramodulation[12] inferences whenever they occur in the resolution refutation of the CCS, leading to a more natural mix of equational reasoning and atomic cuts.

2.6 CERES$^{\omega}$

This section now presents CERES$^{\omega}$, the formulation of CERES for higher-order logic. It draws heavily on the original presentation in Weller's PHD thesis [111] and its journal version [48]. Extended with first-order equality (see section 3.1) it will be the main method underlying our experiments.

When generalizing CERES to higher-order logic, the embedding of resolution refutations into sequent calculus becomes more difficult. Before applying CERES in first-order logic, the input proof needs to be skolemized [6], since projections can only be computed when the end-sequent does not contain strong quantifiers. But in a higher-order resolution calculus, the substitution of a predicate variable may also introduce quantifiers. Even worse, without restrictions, a skolemized formula may be provable even though the original formula is not (see section 2.3.2 for an example).

The approach taken here is to modify the strong quantifier rules of the sequent calculus to use Skolem terms instead of eigenvariables, when they contribute to the end-sequent. Rules with eigenvariables conditions are still necessary to infer formulas contributung to cuts because they appear both negatively and positively in the cut rule, which would lead to different skolem normal forms.

To preserve soundness, the Skolem terms must correspond contain the instances of the weak quantifier inferences which will be applied subsequently. In order to propagate this non-local information, each formula occurrence has a label which annotates the primary formulas of later weak quantifier rules, from which the Skolem context is generated. Another non-local property of Skolem quantifier rules is each quantifier in the end-sequent must have a unique Skolem symbol assigned i.e. duplicated Skolem symbols are only allowed when a subsequent contraction will lead to the same quantifier occurrence in the end-sequent.

The approach taken in CERES$^{\omega}$ is to start with an LK proof of the end-sequent S containing the usual quantifier rules with eigenvariable conditions (see figure 2.2.2). It is then transferred to a proof of S in the calculus LK_{skc} by replacing quantifier inferences contributing to the end-sequent with Skolem quantifier rules. The context of weak quantifier inferences operating on successors of a formula is traced by the means of labels attached to each formula. Out of the LK_{skc} proof, the characteristic sequent set and the cut-free proof projections to each sequent in the set are generated. After finding a resolution refutation of the characteristic sequent set in the R_{al} calculus, it serves as a template to create a proof of S in LK_{skc}. The only cut-formulas in this proof come from simulations of the resolution rule. Even though the cut-formulas may contain quantifiers, no inference operates on its ancestors[13]. Therefore Gentzen style reductive

[12] This also applies to superposition, since a superposition rule can be seen as a restriction of paramodulation.

[13] The restrictedness property will ensure exactly that.

cut-elimination will perform only rank reduction steps, resulting in an LK_{sk} proof . Further post-processing is required since the simulation may generate Skolem quantifier inferences in places of the proof where the eigenvariable condition would not hold. After pushing these inferences towards the root, the Skolem quantifier rules can be replaced by the usual ones, resulting in a cut-free proof of S in LK.

2.6.1 The Sequent Calculi LK_{sk} and LK_{skc}

Like for LK, the formula language used in all calculi are terms of simply typed lambda calculus in β-normal-form. A labeled formula F is written as $\langle F \rangle^{\ell}$, where ℓ is a set of simply typed lambda terms. If the label can be inferred from the context, it is omitted. We also call the unlabeled formula $lsF\emptyset$ the reduct of $\langle F \rangle^{\ell}$. Analogous to an unlabeled sequent, a labeled sequent is a sequent of labeled formulas.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{Y(s) \vdash Y(s) \quad Y(t) \vdash Y(t)}{Y(s) \to Y(t), Y(s) \vdash Y(t)} \to : l}{(\forall X(X(s) \to X(t))), Y(s) \vdash Y(t)} \forall : l}{(\forall X(X(s) \to X(t))) \vdash \neg Y(s), Y(t)} \neg : r}{(\forall X(X(s) \to X(t))) \vdash \neg Y(s) \vee Y(t)} \vee : r}{(\forall X(X(s) \to X(t))) \vdash (\forall X(\neg X(s) \vee X(t)))} \forall : r \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\forall x P(x,s)) \vdash (\forall x P(x,s))}{\neg(\forall x P(x,s)), (\forall x P(x,s)) \vdash} \neg : l \quad (\forall x P(x,t)) \vdash (\forall x P(x,t))}{\neg(\forall x P(x,s)) \vee (\forall x P(x,t)), (\forall x P(x,s)) \vdash (\forall x P(x,t))} \vee : l}{\neg(\forall x P(x,s)) \vee (\forall x P(x,t)) \vdash (\forall x P(x,s)) \to (\forall x P(x,t))} \to : r}{(\forall X(\neg X(s) \vee X(t))) \vdash (\forall x P(x,s)) \to (\forall x P(x,t))} \forall : l}{(\forall X(X(s) \to X(t))) \vdash (\forall x P(x,s)) \to (\forall x P(x,t))} \, cut$$

Figure 2.1: An example of an LK proof

An example showing the application of a replacement rule similar to Leibniz equality can be seen in figure 2.1. Even though the cut formula $\forall X(\neg X(s) \vee X(t))$ can be mainly obtained from $\forall X(X(s) \to (X(t))$ with simple propositional reasoning, it still contains a higher-order quantifier.

The calculus LK_{skc} now adds the Skolem quantifier rules to LK and requires that the label of auxiliary formulas and the primary formula in logical and structural rules are identical. Axiom rules may introduce differently labeled formulas though.

Definition 2.6.1 (Skolem quantifier rules and LK_{skc} Trees)**.**

Skolem Quantifier rules:

$$\dfrac{\Gamma, \langle F(fS_1 \dots S_n) \rangle^{\ell} \vdash \Delta}{\Gamma, \langle \exists_\alpha F \rangle^{\ell} \vdash \Delta} \exists^{sk} : l \qquad \dfrac{\Gamma \vdash \Delta, \langle F(fS_1 \dots S_n) \rangle^{\ell}}{\Gamma \vdash \Delta, \langle \forall_\alpha F \rangle^{\ell}} \forall^{sk} : r$$

with $\ell = S_1, \dots, S_n$ and, if $\tau(S_i) = \alpha_i$ for $1 \leqslant i \leqslant n$, then $f \in \mathcal{K}_{\alpha_1,\dots,\alpha_n,\alpha}$ is a Skolem symbol.

$$\dfrac{\langle FT \rangle^{\ell,T}, \Gamma \vdash \Delta}{\langle \forall_\alpha F \rangle^{\ell}, \Gamma \vdash \Delta} \forall^{sk} : l \qquad \dfrac{\Gamma \vdash \Delta, \langle FT \rangle^{\ell,T}}{\Gamma \vdash \Delta, \langle \exists_\alpha F \rangle^{\ell}} \exists^{sk} : r$$

An LK_{skc} tree consists of inferences from LK together with the rules in definition 2.6.1, where Skolem quantifier rules only apply to ancestors of end-sequent formulas and LK quantifier rules only apply to ancestors of cut-formulas.

In order to express the restrictions on Skolem quantifier rules discussed above, it is necessary to introduce the notions of proper and weakly regular derivations.

Definition 2.6.2. Let π be an LK_{skc} derivation where every label of an end-sequent formula is empty. Then π is called proper.

The intuition behind a weakly regular LK_{skc} tree is that an occurrence of a strong quantifier in the end-sequent will always be introduced using the same Skolem symbol. In particular, this happens if a contraction rule applies after the introduction of that strong quantifier. To make this intuition precise, we also introduce the notions of homomorphic paths and inferences.

Definition 2.6.3 (Homomorphic Paths and Inferences)**.** Let $F(\omega)$ denote the the formula at occurrence ω. Let $P(\mu)$ be the sequence of formulas resulting from mapping each occurrence ω in the sequence of formula occurrences μ to $F(\omega)$, omitting repetitions. Then the sequences of formula occurrences μ and ν are homomorphic, if $P(\mu) = P(\nu)$.

Let $\alpha_1, \ldots, \gamma_1$ and $\alpha_2, \ldots, \gamma_2$ be two homomorphic paths of formula occurrences and c be contraction rule with auxiliary formulas γ_1 and γ_2. Then α_1 and α_2 are homomorphic in c.

Two formula occurrences ω_1 and ω_2 are homomorphic, if there exists a c such that they are homomorphic in c. Let ρ_1 and ρ_2 be two inferences of the same type with auxiliary formula occurrences α_1^1 and α_2^1 in the case of a unary rule and with auxiliary formula occurrences α_1^1, α_1^2 and α_2^1, α_2^2 in the case of a binary rule. The inferences ρ_1 and ρ_2 are homomorphic, if there exists a contraction inference in c such that α_1^1 and α_2^1 are homomorphic in c, and, if applicable, also α_1^2 and α_2^2 are homomorphic in c.

A homorphic inference captures the intuition that a skolem function is associated with a strong quantifier occurrence in the end-sequent. In fact, the full skolem terms agree:

Proposition 2.6.4. If two strong skolem quantifier inferences are homomorphic, then they have identical skolem terms.

Now we can define a notion of proof:

Definition 2.6.5 (Weak Regularity, LK_{skc} and LK_{sk} proofs)**.** An LK_{skc} tree with end-sequent S is weakly regular, if for all distinct strong Skolem quantifier inferences ρ_1, ρ_2 holds: if ρ_1 and ρ_2 have the same Skolem terms, then they are homomorphic. An LK_{skc} proof is a proper, weakly regular LK_{skc} tree. An LK_{sk} proof is an LK_{skc} proof without the cut-rule.

The proof in figure 2.2 shows the LK_{skc} proof corresponding to the LK proof in figure 2.1.

In order to translate an LK proof into LK_{skc}, weak regularity is insufficient. Similar to the first-order notion of regularity, Eigenvarables need to be globally unique and the same reasoning is applied to strong Skolem quantifier rules.

Definition 2.6.6 (Regular LK_{skc} / LK trees)**.** An LK_{skc} / LK tree π is regular if:

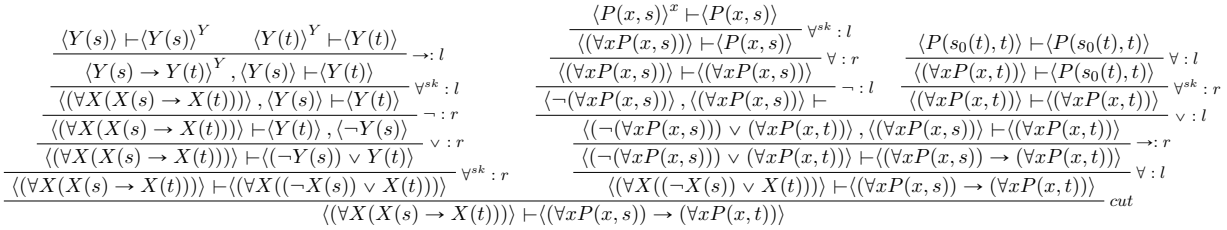


Figure 2.2: The LK_{skc} translation of the LK example from figure 2.1

1. Each strong labeled quantifier inference has a unique Skolem symbol
2. The eigenvariable of each strong quantifier inference ρ occurs only in the sub-proof of ρ within π.

Since each Skolem symbol is unique, every regular proof is also weakly regular. The constructive translation to LK_{skc} given by Hetzl et al. [48] replaces eigenvariables of end-sequent ancestors with according Skolem terms built from fresh Skolem functions, leading to the following theorem:

Theorem 2.6.7 (Skolem Translation)**.** Given a regular LK proof of the end-sequent S, there exists a regular LK_{skc} proof of S.

The distinction whether a formula is an ancestor of a cut-formula or an end-sequent formula is not only relevant to which kind of quantifier rule can be applied. For CERES^{ω}, it is necessary to construct the projections of a proof but the notion of a formula contributing to a cut is visible within the whole method.

Definition 2.6.8 (Cut-ancestors and End-sequent ancestors)**.** Given an LK_{skc} proof π with end-sequent S. For each rule in π inferring the sequent T, define $\mathrm{cutanc}(T)$ as the subsequent of T where each formula is an ancestor of a cut-formula. Likewise, define $\mathrm{esanc}(T)$ as the subsequent of T where each formula is an ancestor of a formula in S.

In any rule, its sequent S can be expressed as $\mathrm{cutanc}(S) \times \mathrm{esanc}(S)$, where the merge of two sequents $\Gamma_1 \vdash \Delta_1 \times \Gamma_2 \vdash \Delta_2$ is defined as $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$. We also lift the merge operation to sequent sets by defining $S \times T$ as the set $\{s \times t | s \in S, t \in T\}$.

To simplify proof transformation algorithms, we introduce a shortcut notation for often used operations. Given a unary inference σ, a binary inference ρ and the LK_{skc} trees π and ψ, we define $\sigma(\pi)$ and $\rho(\pi, \psi)$ as an application of the rules to the respective proof(s). These definitions are not applicable in general, but for some transformations which construct a proof $Transform(\pi)$ with end-sequent T from a proof π with end-sequent S and last inference ρ, there exists a mapping from the formulas in S to formulas in T, such that ρ can be uniquely applied to the image of the auxiliary formula(s) $f(A_1)$ (and $f(A_2)$). A simple example is the application of a substitution τ to all formulas in a proof π: we define $sub(\tau, \pi)$ by distinguishing on the last inference in ρ, referring to the parent proof(s) as π_1 (and π_2).

- ρ is an axiom introducing the sequent S: then $sub(\tau, \pi)$ is the Axiom introducing τS.
- ρ is a quantifier rule with the primary formula $\forall x.Fx$: let $\tau'(v) = \tau(v)$ for $v \neq x$ and $\tau'(x) = x$. Then $sub(\tau, \pi) = \rho(sub(\tau', \pi_1))$.
- ρ is weakening rule introducing F, then $sub(\tau, \pi)$ is obtained by applying a weakening of τF to π_1.
- ρ is a unary logical rule introducing $F \circ G$ with $\circ \in \{\wedge, \vee, \rightarrow\}$ and auxiliary formula F (G): then $sub(\tau, \pi) = \rho(sub(\tau, \pi_1))$, with primary formula $\tau F \circ \tau G$.
- ρ is any other unary rule: then $sub(\tau, \pi) = \rho(sub(\tau, \pi_1))$
- ρ is any binary rule: then $sub(\tau, \pi) = \rho(sub(\tau, \pi_1), sub(\tau, \pi_2)$

We can also lift this notion to sequent sets by defining $S \times_\rho T$ as $\{\rho(s,t)|s \in S, t \in T\}$.

Since the version of LK given here uses multiplicative rules, it is sometimes necessary to add multiple successive weakenings. Given an LK_{skc} tree π, we write $\pi^{\Gamma \vdash \Delta}$ for the proof obtained by applying weakening rules to add the sequent $\Gamma \vdash \Delta$ to π.

2.6.2 Proof Projections

The idea underlying proof projections is to transform a proof of the end-sequent S into a set of cut-free proofs of a weaker end-sequent $S \times C$. The CERES method assembles these projections into a proof of S. Since projections are cut-free, a cut in the completed proof must come from an assembly step. In the case of CERES$^\omega$, a resolution refutation in the R_{al} calculus (see section 2.6.3) serves as a template for the proof construction. Its input – called the characteristic sequent set – is obtained by mapping each projection's end-sequent $S \times C$ to the additional part C. We now state Weller's definition [48] of projections and the characteristic sequent set for CERES$^\omega$.

Definition 2.6.9 (Proof Projections and Characteristic Sequent Set)**.** Let π be a regular LK_{skc}-proof. For each inference ρ in π, we define a set of LK_{sk}-trees, the set of *projections* $\mathcal{P}_\rho(\pi)$, and a set of labeled sequents, the *characteristic sequent set* $\mathrm{CS}_\rho(\pi)$.

- If ρ is an axiom with conclusion $S = \langle A \rangle^{\ell_1} \vdash \langle A \rangle^{\ell_2}$, distinguish:
  - $\mathrm{cutanc}(S) = S$. Then $\mathrm{CS}_\rho(\pi) = \mathcal{P}_\rho(\pi) = \emptyset$.
  - $\mathrm{cutanc}(S) \neq S$. Distinguish:
    (a) If $\mathrm{cutanc}(S) = \vdash \langle A \rangle^{\ell_2}$ then $\mathrm{CS}_\rho(\pi) = \{\vdash \langle A \rangle^{\ell_1}\}$ and $\mathcal{P}_\rho(\pi) = \{\langle A \rangle^{\ell_1} \vdash \langle A \rangle^{\ell_1}\}$,
    (b) if $\mathrm{cutanc}(S) = \langle A \rangle^{\ell_1} \vdash$ then $\mathrm{CS}_\rho(\pi) = \{\langle A \rangle^{\ell_2} \vdash\}$ and $\mathcal{P}_\rho(\pi) = \{\langle A \rangle^{\ell_2} \vdash \langle A \rangle^{\ell_2}\}$,
    (c) if $\mathrm{cutanc}(S) = \vdash$ then $\mathrm{CS}_\rho(\pi) = \{\vdash\}$ and $\mathcal{P}_\rho(\pi) = \{S\}$.
- If ρ is a unary inference with immediate predecessor ρ' with $\mathcal{P}_{\rho'}(\pi) = \{\psi_1, \ldots, \psi_n\}$, distinguish:

(a) ρ operates on ancestors of cut formulas. Then

$$\mathcal{P}_\rho(\pi) = \mathcal{P}_{\rho'}(\pi)$$

(b) ρ operates on ancestors of the end-sequent. Then

$$\mathcal{P}_\rho(\pi) = \{\rho(\psi_1), \ldots, \rho(\psi_n)\}$$

In any case, $\mathrm{CS}_\rho(\pi) = \mathrm{CS}_{\rho'}(\pi)$.

- Let ρ be a binary inference with immediate predecessors ρ_1 and ρ_2.

  (a) If ρ operates on ancestors of cut-formulas, let $\Gamma_i \vdash \Delta_i$ be the ancestors of the end-sequent in the conclusion sequent of ρ_i and define

  $$\mathcal{P}_\rho(\pi) = \mathcal{P}_{\rho_1}(\pi)^{\Gamma_2 \vdash \Delta_2} \cup \mathcal{P}_{\rho_2}(\pi)^{\Gamma_1 \vdash \Delta_1}$$

  For the characteristic sequent set, define

  $$\mathrm{CS}_\rho(\pi) = \mathrm{CS}_{\rho_1}(\pi) \cup \mathrm{CS}_{\rho_2}(\pi)$$

  (b) If ρ operates on ancestors of the end-sequent, then

  $$\mathcal{P}_\rho(\pi) = \mathcal{P}_{\rho_1}(\pi) \times_\rho \mathcal{P}_{\rho_2}(\pi).$$

  For the characteristic sequent set, define

  $$\mathrm{CS}_\rho(\pi) = \mathrm{CS}_{\rho_1}(\pi) \times \mathrm{CS}_{\rho_2}(\pi)$$

The *set of projections* of π, $\mathcal{P}(\pi)$ is defined as $\mathcal{P}_{\rho_0}(\pi)$, and the *characteristic sequent set* of π, $\mathrm{CS}(\pi)$ is defined as $\mathrm{CS}_{\rho_0}(\pi)$, where ρ_0 is the last inference of π.

Coming back to the example from figure 2.2, we can calculate the characteristic sequent set $\{S_1, S_2, S_3\}$ with $S_1 : Y(s) \vdash Y(t)$, $S_2 : \vdash (\forall x P(x, s))$ and $S_3 : (\forall x P(x, t)) \vdash$. The corresponding projections can be seen in figure 2.3, where the formulas contributing to the CS sub-sequents are indicated in colors.

A central property of the characteristic sequent set is that – disregarding labels – it always has a refutation in LK:

Proposition 2.6.10. Let π be a regular LK_{skc}-proof. Then there exists an LK-refutation of the reduct of $\mathrm{CS}(\pi)$.

$$\dfrac{\dfrac{\dfrac{Y(s) \vdash Y(s) \qquad Y(t) \vdash Y(t)}{Y(s), Y(s) \to Y(t) \vdash Y(t)} \to : l}{Y(s), (\forall X(X(s) \to X(t))) \vdash Y(t)} \forall^{sk} : l}{Y(s), (\forall X(X(s) \to X(t))) \vdash Y(t), (\forall x P(x,s)) \to (\forall x P(x,t))} w : r$$

$$(P_{S_1})$$

$$\dfrac{\dfrac{\dfrac{(\forall x P(x,s)) \vdash (\forall x P(x,s))}{(\forall x P(x,s)) \vdash (\forall x P(x,s)), (\forall x P(x,t))} w : r}{\vdash (\forall x P(x,s)), (\forall x P(x,s)) \to (\forall x P(x,t))} \to : r}{(\forall X(X(s) \to X(t))) \vdash (\forall x P(x,s)), (\forall x P(x,s)) \to (\forall x P(x,t))} w : l$$

$$(P_{S_2})$$

$$\dfrac{\dfrac{\dfrac{(\forall x P(x,t)) \vdash (\forall x P(x,t))}{(\forall x P(x,t)), (\forall x P(x,s)) \vdash (\forall x P(x,t))} w : l}{(\forall x P(x,t)) \vdash (\forall x P(x,s)) \to (\forall x P(x,t))} \to : r}{(\forall x P(x,t)), (\forall X(X(s) \to X(t))) \vdash (\forall x P(x,s)) \to (\forall x P(x,t))} w : l$$

$$(P_{S_3})$$

Figure 2.3: The projections of the example from figure 2.2

2.6.3 The Resolution Calculus R_{al}

Since axioms in LK_{skc} are always of the form $F \vdash F$, it is possible to expand it to a proof from axioms containing only atomic formulas [61]. Then the characteristic sequent set actually becomes a clause set. In the first-order case, this set is contradictory and can be refuted by resolution [7]. Unfortunately, in higher-order logic the clause form is not preserved under substitution. For instance, applying the substitution $\sigma = \{Y \leftarrow \lambda z(P(x,z))\}$ to the sequent S_1 from the example, we obtain the sequent $(\forall x P(x,s)) \vdash (\forall x P(x,t))$ which is not in clause form anymore. In order to keep resolution only to apply on literals, a higher-order resolution calculus therefore needs to integrate rules for clause normal form transformation and the introduction of Skolem terms.

Definition 2.6.11 (R_{al} rules, deductions and refutations)**.**

Resolution rules:

$$\frac{}{\langle F \rangle^{\ell_1} \vdash \langle F \rangle^{\ell_2}} \; intro \qquad \frac{S}{S\{X \leftarrow T\}} \; \text{Sub}$$

$$\frac{\Gamma \vdash \Delta, \langle A \rangle^{\ell_1}, \ldots, \langle A \rangle^{\ell_n} \quad \langle A \rangle^{\ell_{n+1}}, \ldots, \langle A \rangle^{\ell_m}, \Pi \vdash \Lambda}{\Gamma, \Pi \vdash \Delta, \Lambda} \; \text{Cut}$$

Logical rules:

$$\frac{\Gamma \vdash \Delta, \langle \neg A\rangle^\ell}{\langle A\rangle^\ell, \Gamma \vdash \Delta}\ \neg^T \qquad \frac{\langle \neg A\rangle^\ell, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \langle A\rangle^\ell}\ \neg^F$$

$$\frac{\Gamma \vdash \Delta, \langle A \vee B\rangle^\ell}{\Gamma \vdash \Delta, \langle A\rangle^\ell, \langle B\rangle^\ell}\ \vee^T \qquad \frac{\langle A \vee B\rangle^\ell, \Gamma \vdash \Delta}{\langle A\rangle^\ell, \Gamma \vdash \Delta}\ \vee_l^F \qquad \frac{\langle A \vee B\rangle^\ell, \Gamma \vdash \Delta}{\langle B\rangle^\ell, \Gamma \vdash \Delta}\ \vee_r^F$$

$$\frac{\langle A \wedge B\rangle^\ell, \Gamma \vdash \Delta}{\langle A\rangle^\ell, \langle B\rangle^\ell, \Gamma \vdash \Delta}\ \wedge^F \qquad \frac{\Gamma \vdash \Delta, \langle A \wedge B\rangle^\ell}{\Gamma \vdash \Delta, \langle A\rangle^\ell}\ \wedge_l^T \qquad \frac{\Gamma \vdash \Delta, \langle A \wedge B\rangle^\ell}{\Gamma \vdash \Delta, \langle B\rangle^\ell}\ \wedge_r^T$$

$$\frac{\Gamma \vdash \Delta, \langle A \to B\rangle^\ell}{\langle A\rangle^\ell, \Gamma \vdash \Delta, \langle B\rangle^\ell}\ \to^T \qquad \frac{\langle A \to B\rangle^\ell, \Gamma \vdash \Delta}{\Gamma \vdash \Delta, \langle A\rangle^\ell}\ \to_l^F \qquad \frac{\langle A \to B\rangle^\ell, \Gamma \vdash \Delta}{\langle B\rangle^\ell, \Gamma \vdash \Delta}\ \to_r^F$$

Quantifier rules:

$$\frac{\Gamma \vdash \Delta, \langle \forall_\alpha A\rangle^\ell}{\Gamma \vdash \Delta, \langle AX\rangle^{\ell,X}}\ \forall^T \qquad \frac{\langle \forall_\alpha A\rangle^\ell, \Gamma \vdash \Delta}{\langle A(fS_1 \dots S_n)\rangle^\ell, \Gamma \vdash \Delta}\ \forall^F$$

$$\frac{\langle \exists_\alpha A\rangle^\ell, \Gamma \vdash \Delta}{\langle AX\rangle^{\ell,X}, \Gamma \vdash \Delta}\ \exists^F \qquad \frac{\Gamma \vdash \Delta, \langle \exists_\alpha A\rangle^\ell}{\Gamma \vdash \Delta, \langle A(fS_1 \dots S_n)\rangle^\ell}\ \exists^T$$

The auxiliary formulas A in the Cut rule is atomic. The variable X in the weak quantifier rules $\forall^T$ and $\exists^F$ does not occur in Γ, Δ and A. The label ℓ in the strong quantifier rules $\forall^F$ and $\exists^T$ consists of $S_1, \dots, S_n$. Moreover $f \in \mathcal{K}_{\alpha_1,\dots,\alpha_n,\alpha}$ is a Skolem symbol with $\tau(S_i) = \alpha_i$ for $1 \leqslant i \leqslant n$. An application of a strong quantifier rule is called *source inference* of $fS_1 \dots S_m$, and $fS_1 \dots S_m$ is called the *Skolem term* of this inference.

It might seem counter-intuitive that the weak quantifier rules introduce a fresh variable. At a second glance, it is similar to the first-order CNF transformation where the formula $P(a) \wedge \neg Q(b) \wedge (\exists x P(x) \to \forall x Q(x))$ corresponds to the clause set $\{ \vdash P(a); Q(b) \vdash ; P(x) \vdash Q(y)\}$ where one occurrence of the bound variable x had to be renamed. Like in first-order resolution the free variable may become instantiated further on in the derivation.

Definition 2.6.12 (R_{al} refutation)**.** Let $\mathcal{C}$ be a set of sequents. A sequence of sequents $S_1, \dots, S_n$ is an *R_{al}-deduction of S_n from $\mathcal{C}$* if for all $1 \leqslant i \leqslant n$ either

1. $S_i \in \mathcal{C}$ or
2. S_i is derived from S_j (and S_k) by an R_{al} rule, where $j, k < i$.

In addition, we require that all $\forall^F$ and $\exists^T$ inferences used have pairwise distinct Skolem symbols. An R_{al}-deduction of the empty sequent from $\mathcal{C}$ is called an *R_{al}-refutation of $\mathcal{C}$*.

A possible R_{al} refutation of the example for a characteristic sequent set from section 2.6.2 is given in figure 2.4.

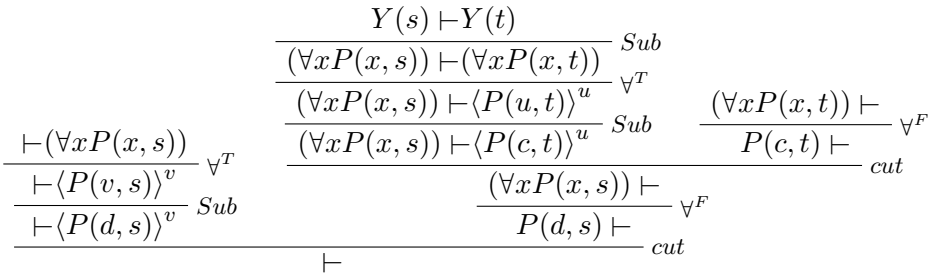


Figure 2.4: A R_{al} refutation of the characteristic sequent set from example in section 2.6.2.

2.6.4 CERES$^\omega$

We now turn to the task of simulating the R_{al} refutation using the projection proofs as axioms which results in an LK_{skc} proof where ancestors of cut formulas are never active formulas of an inference. The analogue notion in first-order CERES is the atomic-cut normal-form. The reason why the restriction to atomic cuts fails is that higher-order atom formulas are not closed under substitution. Nevertheless, these cuts behave like atom formulas in the sense that Gentzen's method only applies rank-reduction (reducing the length of the ancestor introducing a cut formula to the occurrence in the cut rule), never grade reduction rules (reducing the number of logical symbols in the cut formula).

The other complication in contrast to the first-order method is that a substitution might require a CNF transformation in higher-order resolution. Even though a projection preserves the order of quantifier inferences, the CNF transformations happen at the leaves of the proof tree. A possible introduced quantifier then needs to be shifted into a suitable place before De-Skolemizing the proof.

From Projections and an R_{al} refutation to an LK_{skc} Proof

The transformations in the following sections have additional requirements on LK_{skc} trees. The first one comes from the observation that there are no inferences operating on formulas from the characteristic sequent to which a proof is projected. Then each of these formulas can be traced to the unique inference which introduces it.

Definition 2.6.13. Restrictedness and Linearity
Let S be a set of formula occurrences in an LK_{skc} tree π. Then π is called S-linear, if no inferences operate on ancestors of S. If no inferences except contraction operate on ancestors of S, then π is called S-restricted.

Additionally, we call π restricted, if S is the set of cut-formulas in π.

Proposition 2.6.14. Let π be an S-linear proof, where $\omega \in S$ is an occurrence within the end-sequent of π. Then there exists a unique axiom introducing the ancestor of ω.

Proposition 2.6.14 is a direct consequence of linearity: since no inference operates on the ancestors of ω, there is always a unique ancestor in every inference. Looking back at the projec-

tion to S_1 shown in figure 2.3, both the occurrence of $Y(s)$ and $Y(t)$ are linear and introduced by the axioms $Y(s) \vdash Y(s)$ and $Y(t) \vdash Y(t)$.

Moreover, proofs which are S-linear have the property, that the labels of any formula in S can be deleted.

Proposition 2.6.15. Let π be an LK_{skc}-tree, and $\mathcal{S}$ a set of formula occurrences in π that is closed under descendants, and let π be $\mathcal{S}$-linear. If π' is obtained from π by replacing all labels of ancestors of occurrences in $\mathcal{S}$ by the empty label, then π' is an LK_{skc}-tree.

We also have the tools to formally define passive cuts:

Definition 2.6.16. A cut rule on the formula F with parent proofs π_1 and π_2 is called *passive*, if the occurrence of F in the conclusion of π_1 is restricted and the occurrence of F in the conclusion of π_2 is restricted.

A proof π is in *passive-cut normal-form (PCNF)* if all cut rules in π are passive.

In preparation to simulate R_{al}'s substitution rule, we introduce the notion of Skolem parallelism. It expresses that whenever the skolem terms of strong quantifier inferences in two different proofs are unifiable, they must be homomorphic. In particular, an LK_{skc} tree is Skolem parallel to its substitution instances.

Definition 2.6.17 (Skolem parallel)**.** Let ρ_1, ρ_2 be strong labeled quantifier inferences in LK_{skc}-trees π_1, π_2 with Skolem terms S_1, S_2 respectively. ρ_1, ρ_2 are called *Skolem parallel* if for all substitutions σ_1, σ_2, if $S_1\sigma_1 = S_2\sigma_2$ then $\mu_1\sigma_1, \mu_2\sigma_2$ are homomorphic, where μ_1, μ_2 are the maximal downwards paths starting at S_1, S_2 respectively. π_1, π_2 are called Skolem parallel if for all strong labeled quantifier inferences ρ_1, ρ_2 in π_1, π_2 respectively, ρ_1, ρ_2 are Skolem parallel.

In general, the introduced formula in an axiom rule of LK_{skc} may have different labels in the antecedent and succedent. However, if one of the axiom formulas is the ancestor of an element of the characteristic sequent set within the end-sequent of a projection, the labels must agree. Only then the skolem context is properly propagated to the resolution refutation and transferred back accordingly during the simulation of that proof. This property is captured by the notion of suitable axiom labels.

Definition 2.6.18 (Axiom Labels)**.** Let π be an LK_{skc}-tree, let ω be a formula occurrence in π, and let μ be an ancestor of ω that occurs in an axiom A. Then A is called a *source axiom* for ω. Let $\mathcal{S}$ be a set of formula occurrences in π. We say that π *has suitable axiom labels with respect to* $\mathcal{S}$ if for all formula occurrences ω in $\mathcal{S}$, the source axioms of ω are of the form $\langle F \rangle^{\ell} \vdash \langle F \rangle^{\ell}$.

If we recall the axiom case in the definition of the characteristic sequent set (see definition 2.6.9), it is the label of the axiom partner i.e. the antecedent (succedent) occurrence corresponding to the succedent (antecedent) occurrence of F in the axiom $\langle F \rangle^{\ell_1} \vdash \langle F \rangle^{\ell)_2}$ which is attached to a literal in the sequent. If both occurrences of F are cut-ancestors and therefore unlabeled the corresponding characteristic sequent set literal does not propagate the correct skolem context to the resolution calculus. For this reason, this situation is forbidden in a balanced projection.

Definition 2.6.19 (Balancedness)**.** Let π be an LK_{skc}-tree, and let $\mathcal{S}$ be a set of formula occurrences in π. We call π *$\mathcal{S}$-balanced* if for every axiom $\langle F\rangle^{\ell_1} \vdash \langle F\rangle^{\ell_2}$ in π, at least one occurrence of F is an ancestor of a formula occurrence in $\mathcal{S}$. We say that π is *balanced* if π is $\mathcal{S}$-balanced, where $\mathcal{S}$ is the set of end-sequent occurrences of π.

Now we can tie up these definitions into the invariant which will hold on all intermediate LK_{skc}-proofs created during the simulation of the R_{al} refutation of its characteristic sequent set.

Definition 2.6.20 (CERES-projections)**.** Let S be a proper sequent, and C be a sequent. Then an LK_{skc}-tree π is called a CERES-*projection* for (S, C) if the end-sequent of π is $S \times C$ and π is weakly regular, $\mathcal{O}_C$-linear, $\mathcal{O}_S$-balanced, restricted, and has suitable axiom labels with respect to $\mathcal{O}_C$, where $\mathcal{O}_S$ resp. $\mathcal{O}_C$ is the set of formula occurrences of S resp. C in the end-sequent of π.

Let $\mathcal{C}$ be a set of sequents. A set of LK_{skc}-trees $\mathcal{P}$ is called a *set of* CERES-*projections for* $(S, \mathcal{C})$ if for all $C \in \mathcal{C}$ there exists a $\pi(C) \in \mathcal{P}$ such that $\pi(C)$ is a CERES-projection for (S, C) and moreover, for all $\pi_1, \pi_2 \in \mathcal{P}$, π_1 and π_2 are Skolem parallel.

In fact, the projections of a proof are always CERES-projections [48]. This allows us to use them as initial objects for the simulation if the resolution refutation.

Lemma 2.6.21. Let π be a regular LK_{skc}-proof of S. Then $\mathcal{P}(\pi)$ is a set of CERES-projections for $(S, \mathrm{CS}(\pi))$. Furthermore, for all $\psi \in \mathcal{P}(\pi)$, $|\psi| \leqslant |\pi|$.

Definition 2.6.22. Assume we are given a proper sequent S, a set of sequents $\mathcal{C}$, a set of CERES-projections $\mathcal{P}$ for $(S, \mathcal{C})$ and a R_{al} refutation γ of $\mathcal{C}$. We inductively define a series Π of sets of LK_{skc} trees on the length $0 < i < n$ of the refutation. Then the *γ-simulation from the projections $\mathcal{P}$* is the proof corresponding to $S_n = \vdash$ with the end-sequent S, which is contained in $\mathcal{P}_n$.

To construct $\mathcal{P}_n$, we set $\mathcal{P}_0 = \mathcal{P}$ for $i = 0$ and distinguish on the source of the inferred sequent S_i for $i > 0$:

1. $S_i \in \mathcal{C}$: Then we set $\mathcal{P}_i = \mathcal{P}_{i-1}$.

2. S_i is derived from S_j (and S_k) and $\mathcal{P}_{i-1}$ is constructed from $(S, \mathcal{C} \cup \{S_1, \ldots, S_{i-1})$. Since $j < i$ (and $k < i$), we can choose the CERES-projections π_j for (S, S_j) (and π_k for (S, S_k)) from $\mathcal{P}_{i-1}$.

   We set $\mathcal{P}_i = \mathcal{P}_{i-1} \cup \{\pi_i\}$, where π_i is an LK_{skc}-tree defined by distinguishing how S_i is inferred in γ:

   a) $S_i = \langle A\rangle^{\ell}, \Pi \vdash \Lambda$ is derived from $S_j = \Pi \vdash \Lambda, \langle \neg A\rangle^{\ell}$ by $\neg^T$. Then the end-sequent of π_j is $S \times S_j = \Gamma, \Pi \vdash \Lambda, \Delta, \langle \neg A\rangle^{\ell}$.

   By S_j-linearity of π_j, Proposition 2.6.14 we can uniquely identify the axiom $\langle \neg A\rangle^{\ell}$. Let μ end in $\langle \neg A\rangle^{\ell} \vdash \langle \neg A\rangle^{\ell}$ (the labels are identical because π_j has suitable axiom labels with respect to S_j).

By $\mathcal{S}$-balancedness, we may replace this axiom in π_j by

$$\frac{\langle A\rangle^{\ell} \vdash \langle A\rangle^{\ell}}{\langle A\rangle^{\ell}, \langle \neg A\rangle^{\ell} \vdash} \neg\colon l$$

to obtain π_i of $\langle A\rangle^{\ell}, \Gamma, \Pi \vdash \Lambda, \Delta = S \times S_i$.

b) S_i is derived from S_j by some other propositional rule: analogously to the previous case, there exists a unique axiom introducing the auxiliary formula of the inference in π_j. Depending on the rule applied, we perform one of the following replacements to obtain π_i:

$$\neg^F : \langle \neg A\rangle^{\ell} \vdash \langle \neg A\rangle^{\ell} \quad \rightsquigarrow \quad \frac{\langle A\rangle^{\ell} \vdash \langle A\rangle^{\ell}}{\vdash \langle \neg A\rangle^{\ell}, \langle A\rangle^{\ell}} \neg\colon r$$

$$\vee^T : \langle A \vee B\rangle^{\ell} \vdash \langle A \vee B\rangle^{\ell} \quad \rightsquigarrow \quad \frac{\langle A\rangle^{\ell} \vdash \langle A\rangle^{\ell} \quad \langle B\rangle^{\ell} \vdash \langle B\rangle^{\ell}}{\langle A \vee B\rangle^{\ell} \vdash \langle A\rangle^{\ell}, \langle B\rangle^{\ell}} \vee\colon l$$

$$\vee_l^F : \langle A \vee B\rangle^{\ell} \vdash \langle A \vee B\rangle^{\ell} \quad \rightsquigarrow \quad \frac{\langle A\rangle^{\ell} \vdash \langle A\rangle^{\ell}}{\langle A\rangle^{\ell} \vdash \langle A \vee B\rangle^{\ell}} \vee\colon r^1$$

$$\vee_r^F : \langle A \vee B\rangle^{\ell} \vdash \langle A \vee B\rangle^{\ell} \quad \rightsquigarrow \quad \frac{\langle B\rangle^{\ell} \vdash \langle B\rangle^{\ell}}{\langle B\rangle^{\ell} \vdash \langle A \vee B\rangle^{\ell}} \vee\colon r^2$$

The replacements for the cases of $\wedge^F, \wedge_l^T, \wedge_r^T, \rightarrow^T, \rightarrow_l^F, \rightarrow_r^F$ are analogous.

c) $S_i = \langle AS\rangle^{\ell}, \Pi \vdash \Lambda$ is derived from $S_j = \langle \forall A\rangle^{\ell}, \Pi \vdash \Lambda$ by $\forall^F$. Then the end-sequent of π_j is $\langle \forall A\rangle^{\ell}, \Pi, \Gamma \vdash \Delta, \Lambda$. By S_j-linearity and suitable axiom labels there exists a unique axiom $\langle \forall A\rangle^{\ell} \vdash \langle \forall A\rangle^{\ell}$ introducing the ancestor of $\langle \forall A\rangle^{\ell}$. By $\mathcal{S}$-balancedness, we may replace it by

$$\frac{\langle AS\rangle^{\ell} \vdash \langle AS\rangle^{\ell}}{\langle AS\rangle^{\ell} \vdash \langle \forall A\rangle^{\ell}} \forall^{sk}\colon r$$

to obtain π_i of $\langle AS\rangle^{\ell}, \Pi, \Gamma \vdash \Delta, \Lambda$.

d) $S_i = \Pi \vdash \Lambda, \langle AX\rangle^{\ell,X}$ is derived from $S_j = \Pi \vdash \Lambda, \langle \forall A\rangle^{\ell}$ by $\forall^T$. By (IH) we have an LK_{skc}-tree π_j of $\Pi, \Gamma \vdash \Delta, \Lambda, \langle \forall A\rangle^{\ell}$. By S_j-linearity there exists a unique axiom $\langle \forall A\rangle^{\ell} \vdash \langle \forall A\rangle^{\ell}$ introducing the ancestor of $\langle \forall A\rangle^{\ell}$. By $\mathcal{S}$-balancedness, we may replace it by

$$\frac{\langle AX\rangle^{\ell,X} \vdash \langle AX\rangle^{\ell,X}}{\langle \forall A\rangle^{\ell} \vdash \langle AX\rangle^{\ell,X}} \forall^{sk}\colon l$$

to obtain π_i of $\Pi, \Gamma \vdash \Delta, \Lambda, \langle AX\rangle^{\ell,X}$.

e) S_i is inferred from S_j by Sub with substitution σ. Then $\pi_i = \pi_j\sigma$ is an LK_{skc}-tree of $S_j\sigma \times S$.

f) $S_i = \Gamma_j, \Gamma_k \vdash \Delta_j, \Delta_k$ is derived from $S_j = \Gamma_j \vdash \Delta_j, \langle A\rangle^{\ell_1}, \ldots, \langle A\rangle^{\ell_n}$ and $S_k = \langle A\rangle^{\ell_{n+1}}, \ldots, \langle A\rangle^{\ell_m}, \Gamma_k \vdash \Delta_k$ by Cut. By Proposition 2.6.15, we may delete labels from the ancestors of occurrences of A from π_j, π_k respectively, denote these trees by π'_j, π'_k. Take for π_i

$$\dfrac{\dfrac{\dfrac{(\pi'_j)}{\Gamma, \Gamma_j \vdash \Delta, \Delta_j, A, \ldots, A}}{\Gamma, \Gamma_j \vdash \Delta, \Delta_j, A}\,\mathrm{contr}\colon r \quad \dfrac{\dfrac{(\pi'_k)}{A, \ldots, A, \Gamma_k, \Gamma \vdash \Delta_k, \Delta}}{A, \Gamma_k, \Gamma \vdash \Delta_k, \Delta}\,\mathrm{contr}\colon l}{\dfrac{\Gamma, \Gamma, \Gamma_j, \Gamma_k \vdash \Delta, \Delta, \Delta_j, \Delta_k}{\Gamma, \Gamma_j, \Gamma_k \vdash \Delta, \Delta_j, \Delta_k}\,\mathrm{contr}\colon *}\,\mathrm{cut}$$

Proposition 2.6.23. All the transformations in definition 2.6.21 deriving $S \circ S_i$ preserve S_i-linearity, S-balancedness and restrictedness as well as suitable axiom labels with respect to S.

Most properties carry over trivially, detailed arguments for each case are contained in Lemma 3 of the original publication [48]. Furthermore, since all cuts are on restricted formulas, the simulation of an R_{al} proof produces an LK_{skc} proof in passive cut normal-form.

Even for a small proof of length 12 like our example refutation (Figure 2.4), the series π_i of simulations already takes too much space to fit on a page. Therefore figure 2.5 only contains the final proof π_{12}, but the full series[14] is given in appendix C.

[14] We assume the inferences are numbered by a pre-order traversal of the proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{P(c,t)\vdash P(c,t)}{P(c,t)\vdash \forall x P(x,t)}\forall:r}{P(c,t),\forall x P(x,s)\vdash \forall x P(x,t)}w:l}{P(c,t)\vdash \forall x P(x,s)\to\forall x P(x,t)}\to:r}{P(c,t),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t)}w:l$$

(π_9)

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(d,s)\vdash P(d,s)}{P(d,s)\vdash \forall x P(x,s)}\forall^{sk}:r \quad \dfrac{\langle P(c,t)\rangle^{\lambda x P(x,t)}\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}{\forall x P(x,t)\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\forall^{sk}:l}{P(d,s),\forall x P(x,s)\to\forall x P(x,t)\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\to:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\forall^{sk}:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)},\forall x P(x,s)\to\forall x P(x,t)}w:r \quad \dfrac{(\pi_9)}{P(c,t),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t)}}{P(d,s),\forall X(X(s)\to X(t)),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t),\forall x P(x,s)\to\forall x P(x,t)}cut}{P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t),\forall x P(x,s)\to\forall x P(x,t)}c:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t)}c:r$$

(π_{11})

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle P(d,s)\rangle^{\lambda x P(x,s)}\vdash P(d,s)}{\forall x P(x,s)\vdash P(d,s)}\forall^{sk}:l}{\forall x P(x,s)\vdash P(d,s),\forall x P(x,t)}w:r}{\vdash P(d,s),\forall x P(x,s)\to\forall x P(x,t)}\to:r}{\forall X(X(s)\to X(t))\vdash P(d,s),\forall x P(x,s)\to\forall x P(x,t)}w:l \quad \dfrac{(\pi_{11})}{P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t)}}{\forall X(X(s)\to X(t)),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t),\forall x P(x,s)\to\forall x P(x,t)}cut}{\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t),\forall x P(x,s)\to\forall x P(x,t)}c:l}{\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to\forall x P(x,t)}c:r$$

(π_{12})

Figure 2.5: The refutation simulation on the projections of the example from Figure 2.2

From LK_{skc} PCNFs to LK_{sk}

Since the refutation simulation produces a proof in passive-cut normal form, it is sufficient to apply rules similar to the rank reduction case in Gentzen's reductive cut-elimination [105, p.27]. The skolem quantifier rules do not have any eigenvariable conditions and can also be shifted upwards without restrictions. The only interesting case is that of a contraction of a cut-ancestor which comes from the simulation of a resolution cut rule. Here the cut needs to be duplicated.

$$\dfrac{\Gamma_1\vdash\Delta_1,C \quad \dfrac{C,C,\Gamma_2\vdash\Delta_2}{C,\Gamma_2\vdash\Delta_2,F}c:r}{\Gamma_1,\Gamma_2\vdash\Delta_1,\Delta_2}cut \qquad\qquad \dfrac{\Gamma_1\vdash\Delta_1,C \quad \dfrac{\Gamma_1\vdash\Delta_1,C \quad C,C,\Gamma_2\vdash\Delta_2}{\Gamma_1,\Gamma_2\vdash C,\Delta_1,\Delta_2}cut}{\Gamma_1,\Gamma_2\vdash\Delta_1,\Delta_2}cut$$

Therefore, if we can find a R_{al} refutation of the characteristic clause set of a (regular and proper) LK_{skc} proof, we can use the simulation to create an LK_{skc} proof with exclusively passive cuts, which can be removed by Gentzen style rank reduction. In the end, we obtain a (cut-free) LK_{sk} proof.

Theorem 2.6.24. Let π be a regular, proper LK_{skc}-proof of S such that there exists an R_{al}-refutation of $\mathrm{CS}(\pi)$. Then there exists an LK_{sk}-proof of S.

Eliminating the passive cuts in the example, we obtain the derivation in Figure 2.6.

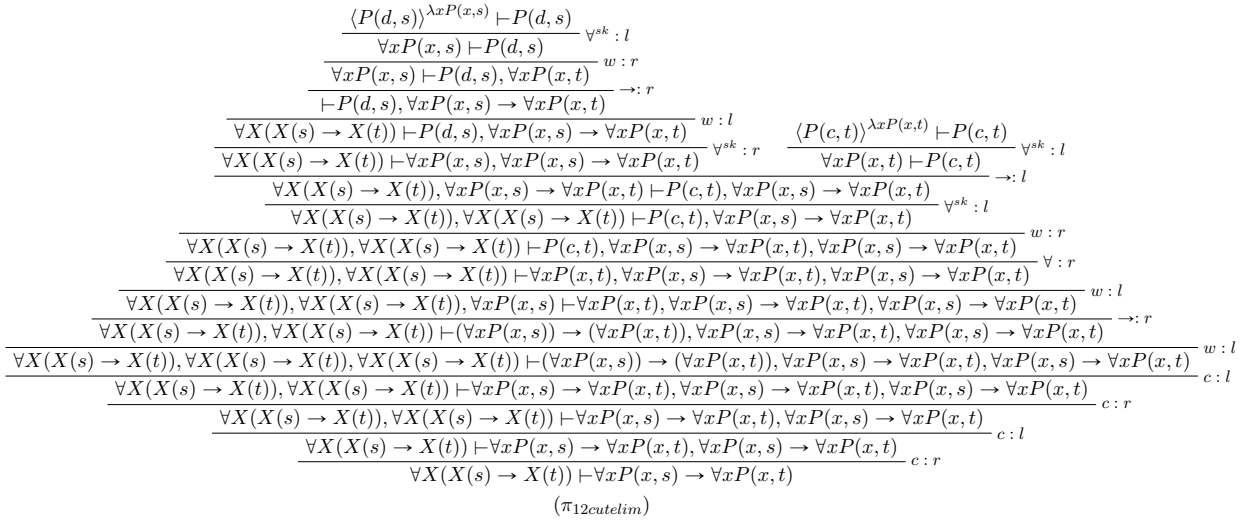


Figure 2.6: The simulation of the example proof after elimination of atomic cuts.

From LK_{sk} to LK

The last part of CERES$^{\omega}$ concerns itself with the removal of the skolem quantifier rules, i.e. with De-Skolemization. Since the LK_{skc} skolem quantifier rules need not adhere to eigenvariable conditions but the LK quantifier rules do. A series of rewrite systems move the quantifier rules into an appropriate place where it can be safely replaced by a strong LK quantifier rule. The first step is to remove all homomorphic inferences on the same branch i.e. all sequential homomorphic pairs.

Definition 2.6.25 (Sequential pruning)**.** Let π be an LK_{sk}-tree and ρ, ρ' inferences in π. Then ρ, ρ' are called *sequential* if they are on a common branch in π. We define the set of *sequential homomorphic pairs* as

$$\mathrm{SHP}(\pi) = \{\langle \rho, \rho' \rangle \mid \rho, \rho' \text{ homomorphic in } \pi \text{ and } \rho, \rho' \text{ sequential}\}.$$

We say that π is *sequentially pruned* if $\mathrm{SHP}(\pi) = \emptyset$.

First, a rewrite relation $\rhd_c$ permutes contraction rules with inferences independent from it as close to the root as possible.

Definition 2.6.26. Let ρ be an inference above an inference σ. Then ρ and σ are *independent* if the auxiliary formula of σ is not a descendent of the main formula of ρ.

Definition 2.6.27 (The relation $\rhd_c$)**.** We will now define the rewrite relation $\rhd_c$ for LK_{sk}-trees π, π', where we assume the inferences $\mathrm{contr}: *$ and σ to be independent:

1. Given the two proofs π, π'

$$\dfrac{\dfrac{\Pi, \Pi, \Gamma \vdash \Delta, \Lambda, \Lambda}{\Pi, \Gamma \vdash \Delta, \Lambda}\, c : *}{\Pi, \Gamma' \vdash \Delta', \Lambda}\, \sigma \qquad \dfrac{\dfrac{\Pi, \Pi, \Gamma \vdash \Delta, \Lambda, \Lambda}{\Pi, \Pi, \Gamma' \vdash \Delta', \Lambda, \Lambda}\, \sigma}{\Pi, \Gamma' \vdash \Delta', \Lambda}\, c : *$$

(π) (π')

then $\pi \rhd_c^1 \pi'$.

2. Given the two proofs π, π'

$$\dfrac{\dfrac{\Pi, \Pi, \Gamma \vdash \Delta, \Lambda, \Lambda}{\Pi, \Gamma \vdash \Delta, \Lambda}\, c : * \qquad \Sigma \vdash \Theta}{\Pi, \Gamma' \vdash \Delta', \Lambda}\, \sigma \qquad \dfrac{\dfrac{\Pi, \Pi, \Gamma \vdash \Delta, \Lambda, \Lambda \qquad \Sigma \vdash \Theta}{\Pi, \Pi, \Gamma' \vdash \Delta', \Lambda, \Lambda}\, \sigma}{\Pi, \Gamma' \vdash \Delta', \Lambda}\, c : *$$

(π) (π')

then $\pi \rhd_c^1 \pi'$.

3. symmetric to case 2

The $\rhd_c$ relation is then defined as the transitive and reflexive closure of the compatible closure of the $\rhd_c^1$ relation.

Rewriting via $\rhd_c$ preserves weak regularity and allows to "zip up" the homomorphic paths into one, such that two sequential homomorphic skolem quantifier inferencs can be merged into one, providing an elimination procedure for sequential homomorphic pairs.

Lemma 2.6.28. Let π be a LK_{sk}-tree with end-sequent S such that π is not sequentially pruned. Then there exists a LK_{sk}-tree π' with end-sequent S such that

$$|\mathrm{SHP}(\pi')| < |\mathrm{SHP}(\pi)|$$

Furthermore, if π is weakly regular, so is π'.

We only state the rewrite rules and refer to the proof of Lemma 2.6.28 of the paper [48] for the full argumentation.

Definition 2.6.29. We define the rewrite relation $\rhd_{zip}^1$ as follows:

- The outermost symbol is a unary operator:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \Delta, \langle F \rangle^{\ell_1}}{\Gamma \vdash \Delta, \langle G \rangle^{\ell_2}}\, \rho}{\vdots}}{\dfrac{\dfrac{\Gamma' \vdash \Delta', \langle F \rangle^{\ell_1}, \langle G \rangle^{\ell_2}}{\Gamma' \vdash \Delta', \langle G \rangle^{\ell_2}, \langle G \rangle^{\ell_2}}\, \rho'}{\vdots}}{\dfrac{\Gamma^* \vdash \Delta^*, \langle G \rangle^{\ell_2}, \langle G \rangle^{\ell_2}}{\Gamma^* \vdash \Delta^*, \langle G \rangle^{\ell_2}}\, c} \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash \Delta, \langle F \rangle^{\ell_1}}{\vdots}}{\dfrac{\Gamma' \vdash \Delta', \langle F \rangle^{\ell_1}, \langle F \rangle^{\ell_1}}{\Gamma' \vdash \Delta', \langle F \rangle^{\ell_1}}\, c}}{\dfrac{\vdots}{\Gamma^* \vdash \Delta^*, \langle G \rangle^{\ell_2}}\, \rho'}$$

- The outermost symbol is a binary operator:

$$\dfrac{\dfrac{\dfrac{\langle F\rangle^\ell, \Gamma \vdash \Delta \qquad \langle G\rangle^\ell, \Pi \vdash \Lambda}{\langle F \vee G\rangle^\ell, \Gamma, \Pi \vdash \Delta, \Lambda}\,\rho}{\vdots}}{\dfrac{\dfrac{\langle F\rangle^\ell, \langle F \vee G\rangle^\ell, \Gamma^* \vdash \Delta^* \qquad \langle G\rangle^\ell, \Pi^* \vdash \Lambda^*}{\langle F \vee G\rangle^\ell, \langle F \vee G\rangle^\ell, \Gamma^*, \Pi^* \vdash \Delta^*, \Lambda^*}\,\rho'}{\vdots}}{\dfrac{\langle F \vee G\rangle^\ell, \langle F \vee G\rangle^\ell, \Gamma^+ \vdash \Delta^+}{\langle F \vee G\rangle^\ell, \Gamma^+ \vdash \Delta^+}\,c : l} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\langle F\rangle^\ell, \Gamma \vdash \Delta}{\langle F\rangle^\ell, \Gamma, \Pi \vdash \Delta, \Lambda}\,w : *}{\vdots}}{\dfrac{\dfrac{\langle F\rangle^\ell, \langle F\rangle^\ell, \Gamma^* \vdash \Delta^*}{\langle F\rangle^\ell, \Gamma^* \vdash \Delta^*}\,c \qquad \langle G\rangle^\ell, \Pi^* \vdash \Lambda^*}{\langle F \vee G\rangle^\ell, \Gamma^*, \Pi^* \vdash \Delta^*, \Lambda^*}\,\rho'}{\vdots}}{\langle F \vee G\rangle^\ell, \Gamma^+ \vdash \Delta^+}$$

The relation $\rhd_{zip}$ is the reflexive, transitive closure of $\rhd^1_{zip}$.

The next step gives us the tools to permute an independent unary ($\rhd_u$) or independent binary rule ($\rhd_b$) ρ over an arbitrary inference σ. We will only give examples of the rules and refer to definitions 21 and 22 in the original publication [48] for a full list. Suppose σ is the $\exists : r$ rule, then the rewrite pattern for $\rhd^1_u$ is:

$$\dfrac{\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash M, G, \Delta}\,\rho}{\Gamma \vdash M, N, \Delta}\,\sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F, N, \Delta}\,\sigma}{\Gamma \vdash M, N, \Delta}\,\rho$$

Suppose σ is the $\vee : l$ rule, then the rewrite pattern for $\rhd^1_u$ is:

$$\dfrac{\dfrac{\Gamma \vdash F, G_1, \Delta}{\Gamma \vdash M, G_1, \Delta}\,\rho \qquad \Pi, G_2 \vdash \Lambda}{\Gamma, \Pi, M, G_1 \vee G_2 \vdash \Delta, \Lambda}\,\sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash F, G_1, \Delta \qquad \Pi, G_2 \vdash \Lambda}{\Gamma, \Pi, F, G_1 \vee G_2 \vdash \Delta, \Lambda}\,\sigma}{\Gamma \vdash M, G_1 \vee G_2, \Delta}\,\rho$$

Since we use the multiplicate version of binary rules, we take possible contractions into account. Supose σ is again the $\exists : l$ rule, then the rewrite pattern for $\rhd^1_b$ is:

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F, G_1 \vdash \Delta_1, \Lambda \qquad \Pi, \Gamma_2, G_2 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, F, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, F, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,c : *}{\Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,\sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F, G_1 \vdash \Delta_1, \Lambda}{\Pi, \Gamma_1, M, G_1 \vee G_2 \vdash \Delta_1, \Lambda}\,\sigma \qquad \Pi, \Gamma_2, G_2 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,c : *$$

Suppose σ is the $\vee : l$ rule, then the rewrite pattern for $\rhd^1_b$ is from

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F_1, G_1 \vdash \Delta_1, \Lambda \qquad \Pi, \Gamma_2, F_2, G_1 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, F_1 \vee F_2, G_1, G_1 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, F_1 \vee F_2, G_1 \vdash \Delta_1, \Delta_2, \Lambda}\,c : * \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda}\,\sigma$$

to

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F_1, G_1 \vdash \Delta_1, \Lambda \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_3, F_1, G_1 \vee G_2 \vdash \Delta_1, \Delta_3, \Lambda}\,\sigma \qquad \dfrac{\Pi, \Gamma_2, F_2, G_1 \vdash \Delta_2, \Lambda \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_3, F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_3, \Lambda}\,\sigma}{\Pi, \Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda}\,c : *$$

The rewrite relations $\rhd_u$ and $\rhd_b$ are the reflexive transitive closures of their one step counterparts $\rhd_u^1$ and $\rhd_b^1$. Again, both rewrite systems preserve weak regularity of a proof.

All but one rewrite rule of $\rhd_c$, $\rhd_u$ and $\rhd_b$ preserve or reduce the number of inferences (because some formulas are now introduced by weakening instead). The only exception is the case of two binary rules in $\rhd_b$, when the auxiliary formula of the downwards inference is contracted (see above). Then the rule permutation needs to duplicate the sub-tree proving $G_2, \Gamma_3 \vdash \Delta_3$. Therefore we want to be able to talk about the branchings occuring between the occurrence of skolem quantifier rule and its destined position in the tree:

Definition 2.6.30. Let π be a LK_{sk}-tree, and let ξ be a branch in π. Let σ, ρ be inferences on ξ and w.l.o.g. let σ be above ρ. Let $\xi_1, \ldots, \xi_n$ be the binary inferences between σ and ρ. For $1 \leqslant i \leqslant n$, let λ_i be the subproofs ending in a premise sequent of ξ_i such that λ_i do not contain σ. Then $\lambda_1, \ldots, \lambda_n$ are called the *parallel trees between σ and ρ.*

Now we formulate an equivalent to the eigenvariable condition by means of Skolem terms. A proof in which each strong skolem quantifier inference is correctly placed can be safely De-Skolemized.

Definition 2.6.31. Let σ be a strong labelled quantifier inference in π with Skolem term S, and ρ be a weak labelled quantifier inference in π with substitution term T. We say that ρ *blocks* σ if ρ is below σ and T contains S. We call σ *correctly placed* if no weak labelled quantifier inference in π blocks σ.

Using the rewrite systems $\rhd_c$, $\rhd_u$ and $\rhd_b$ we can correctly place all strong skolem quantifiers. The two latter are used for the actual movement whereas $\rhd_c$ sequentially prunes the proof which ensures termination despite the possible duplication of sub-proofs during the permutation of two binary inferences.

Lemma 2.6.32. Let π be a LK_{sk}-proof of S. Then there exists an LK_{sk}-proof π' of S such that all strong labelled quantifier inferences in π' are correctly placed.

As a last step, we only need to replace the strong skolem quantifier rules with skolem terms by quantifier rules with eigenvariables and disregard the labels to end up with a cut-free LK proof. Therefore the CERES$^\omega$ method is sound.

Theorem 2.6.33 (Soundness)**.** Let π be a LK_{sk}-proof of S. Then there exists a cut-free LK-proof of S.

2.6.5 Soundness and Completeness Results

We now have a complete cut-elimination procedure: given an LK proof with cuts and an R_{al} refutation of the characteristic sequent set of its LK_{skc} conversion, we can produce an LK proof without cuts (basically putting together the methods showing theorems 2.6.7, 2.6.24 and 2.6.33). There is still one open problem, namely the dependence on the R_{al} refutation. By proposition 2.6.10 we know that the reduct of the characteristic sequent set always has a refutation in LK – and therefore also in Andrew's (unlabeled) $\mathcal{R}$. But it is an open question, if

this implies that there is also an R_{al} refutation for every (labeled) characteristic sequent set of a regular, proper LK_{skc} proof. In practice this result is of less importance than expected, because we have found that even the characteristic clause sets produced in the complete first-order formulation of CERES are usually too complex to be refuted by current first-order provers. Interestingly, the complexity of the problems in different proofs grows so fast that a problem is either provable by the quite outdated Prover 9 as well as by Vampire and the E prover or by none of the three at all[15].

2.7 Automated Higher-Order Theorem Proving

So far we disregarded the actual automation of higher-order resolution. Additional to first-order theorem proving, the main problems are the semi-decidability of higher-order unification and the cut-simulation property of important axioms like Leibniz equality.

2.7.1 Higher-Order Unification

Huet showed the undecidability of higher-order unification by encoding the Post correspondence problem [54] into third-order unification. For this reason, an automated theorem prover can not decide when to stop looking for a unifier even for a single inference. Moreover, the complete set of minimal unifiers is possibly infinite: the problem $X_{\iota>\iota}(f_{\iota>\iota>\iota}(a,b)) = f(X(a),b)$ with the set of unifiers $\{X = \lambda x f(x,b), X = \lambda x f(f(x,b),b), X = \lambda x f(f(f(x,b),b,b)), \ldots\}$ is an example for this. Even with a finite bound, the search space for a prover becomes substantially larger than in the first-order case. Still, Huet gave a semi-decidable algorithm for enumerating the complete set of minimal unifiers [55] which was later on formulated as a rule based system [94]. Terms are classified as rigid, when the head symbol is a constant, or as flexible, when the head symbol is a variable. Then the decomposition rule of Martelli-Montanari [67] handles the unification of rigid with rigid terms. Flexible-rigid pairs are either treated by imitation or projection. With imitation, the variable mimics the head symbol of the rigid term and introduces a fresh variable to apply more rules later on. A projection lifts one of the arguments of the variable to the root of the term tree. One of Huet's insights was that flexible-flexible unifications can be postponed. In principle, the situation allows for both variables to project to an arbitrary term but more unifications further on might require a more specific one. By carrying the flexible-flexible pairs on as constraints in a pre-unifier, the projection terms can be specified lazily. We refrain from giving the whole set of rules but show a possible derivation for the first unifier of the example above:

[15] It is obvious that this claim is quite subjective – it is evident in the schematic analysis of Fürstenberg's proof of the infinitude of primes where only the existence of more than two primes can be found automatically – although Vampire finds the refutation more than 200 times faster than Prover9. It has also been confirmed by other analysis attempts of the author and to a point also in this thesis. Still, all of this is anecdotal and might only depend on finding the right settings for a particular prover.

$\{X(f(a,b)) = f(X(a),b)\}$	imitate $X = \lambda x f(Y(x), Z(x))$
$\{f(Y(f(a,b)), Z(f(a,b))) = f(f(Y(a), Z(a)), b)\}$	decompose
$\{Y(f(a,b)) = f(Y(a), Z(a)); Z(f(a,b))) = b\}$	imitate $Z = \lambda x.b$
$\{Y(f(a,b)) = f(Y(a), b); b = b\}$	remove
$\{Y(f(a,b)) = f(Y(a), b)\}$	project $Y = \lambda x.x$
$\{f(a,b) = f(a,b)\}$	remove
$\{\}$	

Dougherty also proposed a unification algorithm based on combinators [28], but to the best of our knowledge, it has not been used in an automated theorem prover yet. For the resolution calculus R_{al} used in this thesis, all details of unification are hidden by the substitution rule which does not specify where the substitution actually comes from.

2.7.2 Cut-simulation

The expressiveness of higher-order logic allows the use of axioms which are sufficiently strong to replace rules built into the calculus. When the calculus is specifically tailored to restrict the search-space, the simulation of an unsuitable rule undermines the whole effort. In the case of sequent calculus, this is the cut rule: since the reasoning process proceeds from the end-sequent upwards, applying a cut rule amounts to inventing an arbitrary cut-formula. A simple axiom that simulates cut is $\forall X(X \to X)$: assuming the axiom occurs in the context Γ_1, Γ_2, each inference

$$\frac{\Gamma_1 \vdash \Delta_1, F \qquad \Gamma_2, F \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

can be rewritten to

$$\dfrac{\dfrac{\dfrac{\Gamma_1 \vdash \Delta_1, F \qquad \Gamma_2, F \vdash \Delta_2}{\Gamma_1, \Gamma_2, F \to F \vdash \Delta_1, \Delta_2}\, \to : l}{\Gamma_1, \Gamma_2, \forall X(X \to X) \vdash \Delta_1, \Delta_2}\, \forall : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, c : l$$

Benzmüller and Kohlhase showed [10] that similar simulations exist for Leibniz equality, extensionality, comprehension and the axiom of choice. Consequently, an effective higher-order theorem prover replaces these axioms with built-in rules that avoid cut-simulation.

2.7.3 Leo II

The main prover used for the experiments in this thesis is Leo II [11, 16]. In the meantime, the successor Leo III [13] is sufficiently developed to take part in the CASC competition [23] and was regularly used together with the other contestant Satallax [20, 37]. Here, we just give a short overview of Leo II to develop an intuition of problems which are hard for the prover.

The basic inference rules [9] of Leo II resemble those of R_{al}: for each operator there is a decomposition rule; the quantifier rules introduce skolem terms or arbitrary terms, depending on the polarity. The resolution rule postpones unification of two terms s and t by adding the equality

$s = t$ as a constraint to the clause. This allows the rules of pre-unification to co-exist on the same level as the other inference rules, preventing a strict division into phases of resolution and unification. Special rules handle Boolean and functional extensionality, Leibniz and Andrews rules handle equality as a primitive symbol of the language, with a final set of rules handling choice. Factoring is only applied when the sequent is completely in clause form and primitive substitution applies some limited guessing to overcome situations where a sequent set needs to be specialized without having a partner clause to guide the process by unification. Leo II also uses the E prover [86,87] as backend by encoding sub-problems into untyped or typed first-order logic [15], depending on the choice of the user. Relevant for this work is also that extensionality can be explicitly disabled.

This part of the thesis develops the techniques we will be using for our analysis in chapter 4. In section 3.1 we add first-order equality rules to the CERES$^\omega$ method to gain a more natural input language and section 3.2 discusses the differences between unary and binary equality rules. Section 3.6 provides definition rules, an additional way of structuring an input proof, which becomes invisible to the CERES$^\omega_=$ method since definitions can be removed in the pre-processing phase.

We also approach the expensive post-processing steps of eliminating passive and propositional cuts (section 3.5) together with the de-skolemization procedure (section 3.4). Section 3.3 will give us the necessary lemmas for expansion trees to do that. Moreover, we also document the integration of first-order resolution provers (section 3.10) into GAPT, which can be seen as a simplified variant of CERES. To be useful for higher-order cut-elimination, a higher-order input formula must be projected to the first-order fragment. Section 3.9 describes the (rather naive) lambda-lifting and type reconstruction techniques implemented.

Furthermore we will discuss the GAPT implementation and the visualization of proofs in PROOFTOOL (section 3.11) for the investigation of local inferences. The part on sunburst trees (section 3.13) develops a complementary global visualization of tree-shaped proofs.

3.1 CERES$^\omega_=$

Adding first order equality rules to LK is unproblematic since the rules can be simulated assuming the axioms of reflexivity and replacement of equals by equals. A similar translation exists for an unlabeled version of R_{al}. To simplify matters, we will refer to the extended versions of the corresponding calculi of section 2.6.1 as $LK_=$, $LK_{sk=}$, $LK_{skc=}$ and $R_{al=}$.

In the labeled calculi LK_{sk} and R_{al}, the simulation of an equality rule working on $s = t$ needs to introduce weak quantifiers since s and t may contain Skolem terms which would carry on to the end-sequent, thus violating the proper-ness of an LK_{sk} proof. Consequently, the corresponding weak quantifier rules require additional labels in the ancestors of the auxiliary formula up to the axioms introducing them. When applying the CERES$^\omega$ method, this leads to two problems:

For one, a CERES$^\omega$ projection needs to have suitable axioms labels with regard to the ancestors of the characteristic sequent set, thus propagating the additional labels to the axioms of the resolution refutation. It is unclear, if the additional arguments those labels create in an R_{al} refutation are admissible. In principle, each strong quantifier rule of R_{al} introduces a distinct Skolem symbol, which means that a second occurrence of a Skolem term not occurring in the

LK_{sk} proof can only come from a substitution $\{x \leftarrow t[s(\ell_1, \ldots, \ell_n)]\}$. Nonetheless it is not straightforward to prove that given a proof with Skolem terms $s(\ell_1, \ldots, \ell_n)$ without equality arguments can be extended to a proof with equality arguments.

The second problem is the simulation of equality inferences in the R_{al} proof. In principle, an R_{al} equality rule can be mapped to an LK_{sk} equality rule (preferred) or be simulated otherwise. There is also the choice of simulating the inference on the root of the clause projections (similar to first-order CERES) or at the axiom introducing the literal. We will explore some of these possibilities and their interaction with the labels.

The third, and most severe problem is that a simulation disrupts the ancestor relationship of a formula when an LK_{sk} equality rule is applied. In particular, it is unclear if a path which is homomorphic in LK stays homomorphic in $LK_{skc=}$. As an immediate consequence, two strong quantifier rules which were required to introduce the same Skolem symbol before might now introduce different ones.

In any case the following properties need to be fulfilled in the end:

1. The equality translation may only introduce cuts on linear formulas.
   Since atomic formulas are not closed under substitutions, the first-order notion of atomic-cut normal-form can not be upheld in higher-order logic. The closest we can get is a linear formula, which behaves passively and can be treated like an atom during the reductive cut-elimination following the resolution simulation.

2. The properties of CERES-projections need to be preserved.
   The properties are restrictedness, linearity and suitable axiom labels with regard to resolution simulation ancestors and balancedness with regard to end-sequent ancestors.

3. Introduced quantifier rules should be in place.
   We aim for a solution where rewriting steps like sequential pruning are not necessitated by equality rules. An equality translation with out-of-order quantifier rules would invalidate the results on any fragment which preserves the order otherwise. Also, any additional theory formulas in the sequent necessary to simulate equality need to be closed. They are preserved down to the end-sequent and should not bring a subsequent quantifier inference out of place (i.e. invalidate the eigenvariable condition after deskolemization).

4. There should be a straightforward relationship between the refutation of a proof with equality rules and the one without. In particular the extracted expansion proof should have unmodified instantiations.

3.1.1 General observations on labels

The only rules which modify labels are the weak quantifier rules, going from premise to conclusion, the LK_{skc} rules remove labels and the R_{al} rules introduce labels. The reason for that is that a weak R_{al} inference

$$\frac{\Gamma \vdash \Delta, \langle \forall_\alpha A \rangle^\ell}{\Gamma \vdash \Delta, \langle AX \rangle^{\ell, X}} \forall^T$$

is simulated at the (unique) axiom introducing the formula by the rewrite step:

$$\dfrac{\dfrac{\dots \qquad \langle \forall_\alpha A\rangle^\ell \vdash \langle \forall_\alpha A\rangle^\ell}{\dots}}{\Gamma \vdash \Delta, \langle \forall_\alpha A\rangle^\ell} \quad \Rightarrow \quad \dfrac{\dfrac{\dots \qquad \dfrac{\langle AX\rangle^{\ell,X} \vdash \langle AX\rangle^{\ell,X}}{\forall_\alpha A \vdash \langle AX\rangle^{\ell,X}}\forall : l}{\dots}}{\Gamma \vdash \Delta, \langle AX\rangle^{\ell,X}}$$

The labeling of a resolution rule therefore directly depends on its simulation. In particular, whenever the simulation of a $\forall^T$ rule is performed at an axiom, the corresponding resolution rule must introduce a label and whenever a simulation of the rule is performed at the projection root, it must remove a label (since the order is the same as in LK).

The next observation is that the only direct interaction of labels with formulas happens in the strong quantifier rules. Therefore adding an LK equality rule which removes labels will extend the Skolem terms introduced by ancestor inferences on the formula by exactly these labels. Moreover, the extended labels do not reflect on the formula level as long as there is no simulation of strong R_{al} quantifier inferences.

Another role of labels is that they constrain some inferences because they require identical labels on the auxiliary formulas. This also reflects in a projection's property of suitable axiom labels (if one occurrence of F in the axiom $\langle F\rangle^{\ell_1} \vdash \langle F\rangle^{\ell_2}$ is an ancestor of a formula which is part of the simulation of the CSS, then $\ell_1 = \ell_2$), which ensures that Skolem terms introduced in the resolution refutation are correct in their simulation in LK_{skc}.

3.1.2 Without Labels

The axioms used are $Ref = \forall x x = x$, $Repl_1 = \forall u \forall v \forall F(u = v \to (F(u) \to F(v))$ and $Repl_2 = \forall u \forall v \forall F(v = u \to (F(u) \to F(v)))$.

Alternatively, the $\forall F$ can be moved inside the implication, but that does not change the labeling significantly later on[1]. Also given Ref, $Repl_2$ is provable from $Repl_1$ (and vice versa)[2].

The reflexivity axiom $\vdash t = t$ is translated to a proof $Ref \vdash t = t$.

A binary rule is translated as:

$$\dfrac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \Gamma_2 \vdash F[s], \Delta_2}{\Gamma \vdash F[t], \Delta} =: r \quad \Rightarrow \quad \dfrac{\dfrac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \dfrac{\Gamma_2 \vdash P[s], \Delta_2 \qquad P[t] \vdash P[t]}{\Gamma_2, P[s] \to P[t] \vdash P[t], \Delta_2} \to : l}{\Gamma, s = t \to P[s] \to P[t] \vdash P[t], \Delta} \to : l}{\Gamma, (\forall u \forall v \forall F(u = v \to F(u) \to F(v))) \vdash P[t], \Delta} \forall : l \; (EQ_1 : r)$$

with $\Gamma = \Gamma_1, \Gamma_2$ and $\Delta = \Delta_1, \Delta_2$.

The problem with here is that the occurrence of $P[s]$ and $s = t$ in the parent proofs are now always end-sequent ancestors. This changes the CSS significantly which invalidates requirement 4.

[1] The formula $s = t$ will then be labeled by only two instead of three expressions

[2] by proving the instance $s = t \to (\lambda x x = s)s \to (\lambda x x = s)t) \vdash t = s$ and then using $t = s$

3.1.3 LK_{sk} with labels

Unfortunately, requirement 3 enforces the introduction of quantifiers, since s and t need not be ground. This leads to additional labels in the transformation to LK_{skc}, but like for quantifier rules, we need to differentiate if the active formula is a cut-ancestor or not.

We start with equality inferences on end-sequent ancestors, since we already have a translation that. In parallel to the quantifier rules, we call this $=_{sk}\colon r$ which receives the following labels:

$$\dfrac{\dfrac{\Gamma_1 \vdash \langle s = t\rangle^{\ell_1,\ell_2,\ell_3}, \Delta_1 \qquad \dfrac{\Gamma_2 \vdash \langle P[s]\rangle^{\ell_1,\ell_2,\ell_3}, \Delta_2 \qquad \langle P[t]\rangle^{\ell_1,\ell_2,\ell_3} \vdash P[t]}{\Gamma_2, \langle P[s] \to P[t]\rangle^{\ell_1,\ell_2,\ell_3} \vdash P[t], \Delta_2} \to\colon l}{\Gamma, \langle s = t \to P[s] \to P[t])\rangle^{\ell_1,\ell_2,\ell_3} \vdash P[t], \Delta} \to\colon l}{\Gamma, (\forall u \forall v \forall F(u = v \to F(u) \to F(v))) \vdash P[t], \Delta} \forall : l$$
$$(EQ_1 : r)$$

with $\ell_1 = \lambda u \forall v \forall F(u = v \to F(u) \to F(v)))$, $\ell_2 = \lambda v \forall F(s = v \to F(s) \to F(v)))$, $\ell_3 = \lambda F(s = t \to F(s) \to F(t)))$. The corresponding rule is then:

$$\dfrac{\Gamma_1 \vdash \langle s = t\rangle^{\ell,\ell_1,\ell_2,\ell_3}, \Delta \qquad \Gamma_2 \vdash \langle F[s]\rangle^{\ell,\ell_1,\ell_2,\ell_3}, \Delta}{\Gamma_1, \Gamma_2 \vdash \langle F[t]\rangle^{\ell}, \Delta_1, \Delta_2} =_{sk}\colon r$$

In particular, since $P[t]$ is an end-sequent ancestor, the axiom $\langle P[t]\rangle^{\ell_1,\ell_2,\ell_3} \vdash P[t]$ does not need to have suitable axiom labels.

Rules on operating on cut-ancestors don't introduce labels, therefore the equality rule $=\colon r$ looks like:

$$\dfrac{\Gamma_1 \vdash s = t, \Delta \qquad \Gamma_2 \vdash F[s], \Delta}{\Gamma_1, \Gamma_2 \vdash F[t], \Delta_1, \Delta_2} =\colon r$$

Since the simulation of a resolution refutation only creates cuts on linear formulas (requirement 2), the only $=\colon r$ rules occurring will come from the simulation.

3.1.4 Simulating Equality rules in R_{al}

The CSS and the Projections

The best upfront translation of $=\colon r$ we found (appendix 3.1.7) enforces the application of equalities and disallows refutations in which these applications are not necessary. Therefore we drop requirement 4 and try a different approach. There we define cut-ancestorship directly via the equality rules and remove the resolution equalities during the simulation of the refutation.

Extending definition 2.6.9 (respectively definition 11 of [48]) we define the characteristic sequent set $\mathcal{C}_\rho(\pi)$ and $\mathcal{P}_\rho(\pi)$ as :

Definition 3.1.1.

- The inference ρ is an $=\colon r_{sk}$ rule inferring $F[t]$ from $F[s]$ and $s = t$. The rule always works on end-sequent ancestors, therefore we define the CSS as $\mathcal{C}_\rho(\pi) = \mathcal{C}_{\rho_1}(\pi_1) \times \mathcal{C}_{\rho_2}(\pi_2)$. Like all rules working on end-sequent ancestors, the projection repeats the rule:

$$\dfrac{\begin{array}{cc}(P_1) & (P_2)\\ \Gamma_1 \vdash \langle s = t\rangle^{\ell}, \Delta_1 & \Gamma_2 \vdash \langle F[s]\rangle^{\ell}, \Delta_2\end{array}}{\begin{array}{c}\Gamma_1, \Gamma_2 \vdash F[t], \Delta_1, \Delta_2\\ (P)\end{array}} =\colon r$$

  with $\ell = \{\ell_1, \ell_2, \ell_3\}$, $P_1 \in \mathcal{P}_{\rho_1}$ and $P_2 \in \mathcal{P}_{\rho_2}$.

- The inference ρ is an $=\colon r$ rule inferring $F[t]$ from $F[s]$ and $s = t$. The rule always works on cut-ancestors, therefore we define the CSS as $\mathcal{C}_\rho = \mathcal{C}_{\rho_1} \cup \mathcal{C}_{\rho_2}$. Like all rules working on cut-ancestors, $\mathcal{P} = \mathcal{P}_{\rho_1} \cup \mathcal{P}_{\rho_2}$.

Since the extended definition does not treat equalities different from other binary rules, the argumentation in the proof of proposition 2.6.10[3] still holds for $LK_=$.

Simulating an equality rule

Since the labeling of the equality rule of R_{al} directly depends on its simulation, we concentrate on the shape first and insert the label denoted with a question-mark (?) later:

$$\dfrac{\Gamma_1 \vdash \langle s = t\rangle^{\ell_1}, \Delta_1 \qquad \Gamma_2 \vdash \langle F[s]\rangle^{\ell_2}, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \langle F[t]\rangle^{?}, \Delta_1, \Delta_2} =^F$$

Simulating such a rule needs to extend the proof of Lemma 3 in [48] with the following case:

- S_i is derived from S_j and S_k by the $=^T$ rule:
  By definition, there exist projections $\pi_j \in \mathcal{P}_{i-1}$ to $\Gamma, \Pi_1 \vdash \langle s = t\rangle^{\ell_1}, \Delta, \Lambda_1$ and $\pi_k \in \mathcal{P}_{i-1}$ to $\Gamma, \Pi_2 \vdash \langle F[s]\rangle^{\ell_2}, \Delta, \Lambda_2$.

  By induction hypothesis we assume π_j and π_k to be weakly regular. It also provides us with the knowledge that π_j is $\Pi_1 \vdash \Lambda_1$ -linear, that π_k is $\Pi_2 \vdash \Lambda_2$ -linear and that both proofs are $\Gamma \vdash \Delta$ -balanced and restricted. They also have suitable axiom labels with respect to $\Pi_1 \vdash \Lambda_1$ and $\Pi_2 \vdash \Lambda_2$ respectively.

  Since the occurrences of $s = t$ and $F[s]$ are linear, we may obtain the proofs π'_j and π'_k by deleting the labels ℓ_1 and ℓ_2 from their ancestors in π_j and π_k.

  Then we can use the projection $EQPROJ$ to $Repl_1$ to connect the proofs via two applications of the cut-rule:

[3] see the proof of proposition 10 in [48] for details

$$\dfrac{\dfrac{\dfrac{(\pi'_j)}{\Gamma, \Pi_1 \vdash s = t, \Delta, \Lambda_1} \qquad \dfrac{(EQPROJ)}{Repl, s = t, F[s] \vdash \langle F[t] \rangle^{\ell}}\, cut}{\Gamma, Repl, \Pi_1, F[s] \vdash \langle F[t] \rangle^{\ell}, \Delta, \Lambda_1} \qquad \dfrac{(\pi'_k)}{\Gamma, \Pi_2 \vdash F[s], \Delta, \Lambda_2}\, cut}{\Gamma, Repl, \Pi_1, \Pi_2 \vdash \langle F[t] \rangle^{\ell}, \Delta, \Lambda_1, \Lambda_2}$$
$$(\pi_i)$$

The projection $EQPROJ$ is defined as:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{s = t \vdash \langle s = t \rangle^{\ell}} \qquad \dfrac{\dfrac{}{F[s] \vdash \langle F[s] \rangle^{\ell}} \qquad \dfrac{}{\langle F[t] \rangle^{\ell} \vdash \langle F[t] \rangle^{\ell}}}{\langle F[s] \to F[t] \rangle^{\ell}, F[s] \vdash \langle F[t] \rangle^{\ell}} \to : l}{\langle s = t \to F[s] \to F[t] \rangle^{\ell}, F[s], s = t \vdash \langle F[t] \rangle^{\ell}} \to : l}{Repl, F[s], s = t \vdash \langle F[t] \rangle^{\ell}} \forall : l$$
$$(EQPROJ)$$

where $\ell = \{\lambda u \forall v \forall F(u = v \to F(u) \to F(v)), \lambda v \forall F(s = v \to F(s) \to F(v)), \lambda F(s = t \to F(s) \to F(t))\}$.

An inspection of the tree π_i reveals it is $\Pi_1, \Pi_2 \vdash \Lambda_1, \Lambda_2, F[t]$ -linear: linearity carries over from the parent proofs for $\Pi_{1,2}$ and $\Lambda_{1,2}$ and no inference operates on ancestors of $F[t]$. The tree is also $\Gamma \vdash \Delta$ -balanced and restricted, since nothing changed with regard to Γ and Δ. Finally, the tree has also suitable axiom labels with respect to $\Pi_1, \Pi_2 \vdash \Lambda_1, \Lambda_2, F[t]$: again, the property carries over from the parent proofs for $\Pi_{1,2}$ and $\Lambda_{1,2}$. The introduction rule for $\langle F[t] \rangle^{\ell} \vdash \langle F[t] \rangle^{\ell}$ in EQPROJ also has suitable axiom labels.

Now we can insert the missing label into the equality rule given above:

$$\dfrac{\Gamma_1 \vdash \langle s = t \rangle^{\ell_1}, \Delta_1 \qquad \Gamma_2 \vdash \langle F[s] \rangle^{\ell_2}, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \langle F[t] \rangle^{\ell}, \Delta_1, \Delta_2} =^{F}$$

with ℓ as defined above and ℓ_1, ℓ_2 arbitrary. What is surprising here is that the labels ℓ_1 and ℓ_2 vanish and $F[t]$ starts with a fresh set of labels coming from the translation of the equality rule.

As noted above, the refutation simulation still contains $=_{sk}$ rules but no more $=$ rules. The translation from section 3.1.3 allows us to obtain an LK_{skc} proof and continue by applying the original CERES$^{\omega}$ method.

3.1.5 Simulating first order equality proofs

The first observation we can make is that we can enforce the CSS to be in clause form by expanding non-atomic introduction rules $F \vdash F$. The algorithm [61][4] recursively applies the

[4]The algorithm is contained in the proof of Lemma 4.1.1 within the book.

outermost operator of F to both sides of the sequent. The only interesting case is that of a quantifier where the strong inference has to be applied below the weak inference. Now if the SUB rule never substitutes predicate variables with quantifier-free instances, the clause form is preserved. Therefore, no $R_{al=}$ refutation of the characteristic sequent set will contain a quantifier inference.

The second observation is that the computation of a projection preserves the order of inferences, since only some inferences are skipped. Also the conversion to LK_{sk} preserves the order of inferences. Together with requirement 3, which is preserved by the simulation of equality rules defined above, the strong quantifier inferences are still in place. Therefore, no further post-processing is necessary for de-skolemization.

Since cuts on propositional formulas do not interfere with the extraction of an expansion proof, we can obtain it directly from the refutation simulation. In the case of a first-order refutation, no simulation will be applied to axioms. In the terms of expansion sequent proofs, this means that the tree shape is invariant over first-order substitutions. Then the substitution obtained from grounding the refutation applied to each projection will create expansion sequents which can be directly merged. The subsequent of trees for end-sequent ancestors is then the same as the one extracted from the atomic-cut normal-form.

3.1.6 CERES^{ω} can produce quantified cuts

In this section we investigate a phenomenon of CERES^{ω} which at first seems counter-intutive: since the atomicity of formulas is not preserved under substitution, the simulation of a cut on an atom formula might subsequently be expanded to a cut on an arbitrary formula. Take for instance the following R_{al} refutation:

$$\dfrac{\dfrac{\vdash P(t) \qquad \dfrac{\dfrac{P(X) \vdash X(b) \qquad \dfrac{X(y) \vdash Q(y)}{X(b) \vdash Q(b)}\,Sub}{P(X) \vdash Q(b)}\,cut}{P(t) \vdash Q(b)}\,Sub}{\vdash Q(b)}\,cut \qquad Q(b) \vdash}{\vdash}\,cut$$

With $t = \lambda x \forall Y(Y(x) \to Y(x))$. Now the corresponding ground refutation cuts on $\forall Y(Y(b) \to Y(b))$:

$$\dfrac{\dfrac{\vdash P(t) \qquad \dfrac{P(t) \vdash \forall Y(Y(b) \to Y(b)) \qquad \forall Y(Y(b) \to Y(b)) \vdash Q(b)}{P(t) \vdash Q(b)}\,cut}{\vdash Q(b)}\,cut \qquad Q(b) \vdash}{\vdash}\,cut$$

If the set $\{\vdash P(t); P(X,b) \vdash X(b); X(y) \vdash Q(y)\}$ is the characteristic sequent set of an LK_{skc} proof π, then the non-atomic cut carries over to the simulation of the refutation on the projections of π. But in contrast to π, every occurrence of a cut-ancestor formula is restricted. Since only contraction inferences are allowed to operate on ancestors of restricted occurrences, the structure of the cut-formula is irrelevant.

3.1.7 A failed translation

A possible way to fix this problem is to create two rules, one for equality operations on end-sequent ancestors ($=_{sk}$) and one for cut-ancestors ($=$). Then the translation of $=_{sk1} : r$ will be the one from $EQ_1 : r$ and the translation of $=_1 : r$ will be that of $CEQ_1 : r$ below, which adds artificial cuts:

$$\dfrac{\dfrac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \dfrac{s = s \vdash s = t \qquad \dfrac{\dfrac{\Gamma_2 \vdash P[s], \Delta_2 \qquad \dfrac{P[s] \vdash P[s] \qquad P[t] \vdash P[t]}{\Gamma_2, P[s], P[s] \to P[t] \vdash P[t]} \to : l}{\Gamma_2, P[s] \to P[t] \vdash P[t], \Delta_2} cut}{\Gamma, s = t, s = t \to (P[s] \to P[t]) \vdash P[t], \Delta} \to : l}{\Gamma, s = t \to (P[s] \to P[t]) \vdash P[t], \Delta} cut}{\Gamma, (\forall u \forall v \forall F(u = v \to F(u) \to F(v))) \vdash P[t], \Delta} \forall : l$$
$$(CEQ_1 : r)$$

The cut on $s = t$ is unproblematic, but the cut on $P[s]$, even when restricted to $P[s]$ being an atom, may become non-atomic during subsequent substitutions if P is a predicate variable. If we restrict the substitution such that this case does not occur, we still need to investigate the relation of this rule to the simulation of a resolution refutation.

If we define the characteristic sequent set analogously to the other binary rules, then for the proof π with parents π_1 and π_2, $CS_{=}(\pi) = \mathrm{CS}(\pi_1) \cup \mathrm{CS}(\pi_2)$ and $CS_{=_{sk}}(\pi) = \mathrm{CS}(\pi_1) \times \mathrm{CS}(\pi_2)$. In contrast to that, the translated CSS is $CS_{=}(\pi) = \mathrm{CS}(\pi_1) \cup (s = t \vdash \times (\mathrm{CS}(\pi_2) \cup F(s) \vdash F(t))) = \mathrm{CS}(\pi_1) \cup (s = t \vdash \times \mathrm{CS}(\pi_2)) \cup s = t, F(s) \vdash F(t)))$. A projection to $s = t, F(s) \vdash F(t)$ consists of the application of the equality rule or directly its translation:

$$\dfrac{\dfrac{s = t \vdash s = t \qquad \dfrac{\dfrac{P[s] \vdash P[s], \qquad P[t] \vdash P[t]}{P[s], P[s] \to P[t] \vdash P[t],} \to : l}{s = t, P[s], s = t \to P[s] \to P[t]) \vdash P[t]} \to : l}{s = t, P[s], (\forall u \forall v \forall F(u = v \to F(u) \to F(v))) \vdash P[t]} \forall : l$$
$$(\mathcal{P}_{s=t, P[s] \vdash P[t]})$$

The problem here is that the additional occurrence of each equation $s = t$ applied to a cut-ancestor ends up in the antecedent of the corresponding characteristic sequent. This means we must use $\mathcal{P}_{s=t, P[s] \vdash P[t]}$ or resolve with a positive occurrence of $s = t$ to remove it. Therefore this approach excludes refutations without equality for these cases, which is undesirable.

3.1.8 Relative completeness

Similar to proposition 2.6.10 we can state that the reduct of an $LK_{skc=}$ proof is refutable in $LK_{=}$.

Proposition 3.1.2. Let π be a regular $LK_{skc=}$-proof. Then there exists an $LK_{=}$-refutation of the reduct of $\mathrm{CS}(\pi)$.

Proof. We extend the inductive construction of an $LK_=$ refutation γ_ρ for an inference ρ with conclusion S in π used to prove proposition 2.6.10[5] by the case for the binary equality rule. It is treated exactly like the other binary rules.

- ρ is an equality inference: w.l.o.g. assume the inference is an $=: r/=_{sk}$: r inference proving $F[t]$ from $s = t$ and $F[s]$, the other side is symmetrical. Let ρ_1, ρ_2 be the parent inferences of ρ in π with the conclusion S_1, S_2. By IH we can assume that the refutations γ_{ρ_1} and γ_{ρ_2} of $\mathcal{C}_{\rho_1}(\pi), \mathcal{C}_{\rho_2}(\pi)$ have been constructed. Then we distinguish on the cut-ancestorship of $F[t]$:
  - $F[t]$ is a cut-ancestor:
    Then ρ must be an $=: r$ inference and $\mathcal{C}_\rho(\pi) = \mathcal{C}_{\rho_1}(\pi) \cup \mathcal{C}_{\rho_2}(\pi)$. We reapply ρ on γ_{ρ_1} and γ_{ρ_2} and obtain γ_ρ, which has exactly the axioms of $\mathcal{C}_{\rho_1}(\pi) \cup \mathcal{C}_{\rho_2}(\pi)$.
  - $F[t]$ is an end-sequent ancestor:
    Then ρ must be an $=_{sk}$ inference and $\mathcal{C}_\rho(\pi) = \mathcal{C}_{\rho_1}(\pi) \times \mathcal{C}_{\rho_2}(\pi)$. Now we replace all axioms A in γ_{ρ_2} by the weakened sequent $A \times C$ for each $C \in \mathcal{C}_{\rho_1}(\pi)$ and contract all duplications of C to obtain a proof γ_C. Then replace all axioms $C \in \mathcal{C}_{\rho_1}(\pi)$ in γ_{ρ_1} by the proof γ_C and again contract duplications in the end-sequent. Each axiom now is a a merge $C_1 \times C_2$ with $C_1 \in \mathcal{C}_{\rho_1}(\pi), C_2 \in \mathcal{C}_{\rho_2}(\pi)$ which is exactly $\mathcal{C}_{\rho_1}(\pi) \times \mathcal{C}_{\rho_2}(\pi)$.

Since we only simulate inferences on cut-ancestor occurrences and the end-sequent is by definition free of cut-ancestors, we have constructed an LK proof of the empty sequent with axioms from $\mathcal{C}_\rho(\pi)$. □

Since it is still unclear if there is a R_{al} refutation whenever there is an LK-refutation of the reduct of a sequent, we also leave the same question with regard to $LK_=$ and $R_{al=}$ open.

3.2 Unary versus Binary Equality Rules in CERES

The first-order formulation of CERES is historically defined with binary equality rules for LK which directly mirror the paramodulation resolution rule [4]. But proof theoretically, a unary equality rule seems easier to handle. A rule infers $F(t)$ from $F(s)$ under the condition that the context contains the equation $s = t$:

Definition 3.2.1 (Unary Equality Rules)**.**

$$\frac{\Gamma, s = t, F(s) \vdash \Delta}{\Gamma, s = t, F(t) \vdash \Delta} =: ul \qquad \frac{\Gamma, s = t \vdash F(s), \Delta}{\Gamma, s = t \vdash F(t), \Delta} =: ur$$

At the first glance it seems there is just one primary formula but this is misleading: if $s = t$ is the ancestor of a cut-formula, permutating the corresponding cut over the equality would invalidate the side-condition of the rule and prevent its application. Therefore we consider the rule to

[5] The construction is given in the proof of Proposition 10 in [48].

have two primary formulas, marking the ancestors of the occurence of the primary formula $F(t)$ red and marking the ancestors of the occurence of the primary formula $s = t$ blue. To show the difference to the unary rule, we repeat the binary version and color the ancestors of $F(t)$ red:

$$\frac{\Gamma_1 \vdash s = t, \Delta_1 \qquad \Gamma_2 \vdash F(s), \Delta_2}{\Gamma_1, \Gamma_2 \vdash F(t), \Delta_1, \Delta_2} =: r$$

In principle, both sets of rules are equivalent because they can simulate each other. Expressing the binary rule by means of the unary one needs to introduce the equality by weakening which is then cut with the second parent proof deriving $\Gamma_1 \vdash s = t, \Delta_1$.

Lemma 3.2.2 (Binary equality is simulated by unary equality.)**.**

$$\dfrac{\begin{array}{cc} (\pi_1) & \dfrac{\begin{array}{c}(\pi_2)\\ \Gamma_2 \vdash F(s), \Delta_2\end{array}}{\Gamma_2, s = t \vdash F(s), \Delta_2}\, w : l \\ \Gamma_1 \vdash s = t, \Delta_1 & \Gamma_2, s = t \vdash F(t), \Delta_2 \end{array}}{\Gamma_1, \Gamma_2 \vdash F(t), \Delta_1, \Delta_2}\, \begin{array}{l} =: ur \\ cut \end{array}$$

Likewise, using the tautological axiom $s = t \vdash s = t$ to provide the equality in the consequent, it is easy to express the binary rule:

Lemma 3.2.3 (Unary equality is simulated by binary equality.)**.**

$$\dfrac{\dfrac{\begin{array}{cc} s = t \vdash s = t & \dfrac{\begin{array}{c}(\pi_3)\\ \Gamma, s = t \vdash F(s), \Delta\end{array}}{\Gamma, s = t, s = t \vdash F(t), \Delta}\, =: r \end{array}}{\Gamma, s = t \vdash F(t), \Delta}\, c : l}{\begin{array}{c}(\pi)\end{array}}$$

As an alternative it would be possible to use the theory axiom $\vdash s = t$ and save the contraction. This would only work in the first-order formulation of CERES but not for the higher-order formulation $\mathrm{CERES}^{\omega}_{=}$(see section 3.1) since the latter does not allow non-tautological axioms. Therefore we will stick to the translation provided above. As a side-effect, the translation improves the visibility of both active formulas in the unary rules. cut-ancestorship of both primary formulas to calculate the projections and the characteristic sequent set.

3.2.1 The CS and Projection for Unary Equality Rules

We use the simulation π (Lemma 3.2.3) of the unary rule by the binary rule and use the definitions for the latter (see section 3.1.1[6]) to compute the set of projections $\mathcal{P}_\rho(\pi)$ and the characteristic sequent set $\mathcal{C}_\rho(\pi)$. To simplify matters we will refer to the axiom inference introducing $s = t \vdash s = t$ as proof π_4.

[6] The $\mathrm{CERES}^{\omega}_{=}$ rules are identical to their first-order counterparts.

Then by induction, we assume that the parent sets $\mathcal{P}_{\rho_3}(\pi_3)$, $\mathcal{C}_{\rho_3}(\pi_3)$, $\mathcal{P}_{\rho_4}(\pi_4)$, $\mathcal{C}_{\rho_4}(\pi_4)$ have been constructed, where ρ, ρ_3 and ρ_4 are the respective last inference in π, π_3 and π_4. We differentiate on the cut-ancestorship of $F(t)$ and $s = t$ in the end-sequent of proof π to create $\mathcal{P}_\rho(\pi)$ and $\mathcal{C}_\rho(\pi)$:

1. if $F(t)$ is a cut-ancestor: Then the equality inference operates on a cut-ancestor and $\mathcal{C}_\rho(\pi) = \mathcal{C}_{\rho_3}(\pi_3) \cup \mathcal{C}_{\rho_4}(\pi_4)$.

   a) If $s = t$ is a cut-ancestor: Then $\mathcal{C}_{\rho_4}(\pi_4) = \{\}$ and $\mathcal{C}_\rho(\pi)$ is $\mathcal{C}_{\rho_3}(\pi_3) \cup \{\} = \mathcal{C}_{\rho_3}(\pi_3)$. The projections are unchanged, i.e. $\mathcal{P}_\rho(\pi) = \mathcal{P}(\pi_3)$.

   b) If $s = t$ is an es-ancestor: Then $\mathcal{C}_{\rho_4}(\pi_4) = \{\vdash s = t\}$ and $\mathcal{C}_\rho(\pi)$ is $\mathcal{C}_{\rho_3}(\pi_3) \cup \{\vdash s = t\}$. Since

   $$\begin{array}{c} s = t \vdash s = t \\ (\pi_4) \end{array}$$

   is an LK axiom as well as the projection of itself, we can add it to the set of projections i.e. $\mathcal{P}_\rho(\pi) = \mathcal{P}(\pi_3) \cup \{\pi_4\}$.

2. if $F(t)$ is an end-sequent ancestor: Then the equality inference operates on an end-sequent ancestor and $\mathcal{C}_\rho(\pi) = \mathcal{C}_{\rho_3}(\pi_3) \times \mathcal{C}_{\rho_4}(\pi_4)$.

   a) If $s = t$ is a cut-ancestor: If we look at the simulation in transformation 3.2.3, we see that the $eq : r$ rule needs to be applied but the $c : l$ must not, leading to a proof of $\Gamma, \Pi, s = t \vdash \Delta, \Lambda F(t)$ where the sub-sequent $\Pi \vdash \Lambda$ contains the literals of the characteristic sequent we are projecting to.
   Then $\mathcal{C}_{\rho_4}(\pi_4) = \{s = t \vdash\}$ and $\mathcal{C}_\rho(\pi)$ is $\mathcal{C}_{\rho_3}(\pi_3) \times \{s = t \vdash\}$. For the projections $\mathcal{P}_\rho(\pi)$ we apply the following inference:

   $$\dfrac{\dfrac{\begin{array}{c}(P)\\ \Gamma, \Pi \vdash \Lambda, \Delta F(s)\end{array}}{\Gamma, \Pi, s = t \vdash \Lambda, \Delta, F(s)}\, w : l}{\Gamma, \Pi, s = t \vdash \Lambda, \Delta, F(t)} =: ur$$

   to each projection $P \in \mathcal{C}_{\rho_3}(\pi_3)$.

   b) If $s = t$ is an es-ancestor: Then $\mathcal{C}_{\rho_4}(\pi_4) = \{\vdash\}$ and $\mathcal{C}_\rho(\pi)$ is $\mathcal{C}_{\rho_3}(\pi_3) \times \{\vdash\} = \mathcal{C}_{\rho_3}(\pi_3)$. The equation rule is repeated, i.e. we infer

   $$\dfrac{\begin{array}{c}(P)\\ \Gamma, s = t \vdash F(s), \Delta\end{array}}{\begin{array}{c}\Gamma, s = t \vdash F(t), \Delta\\ (P')\end{array}} =: ur$$

   for each parent projection P. Now the new set of projections $\mathcal{C}_\rho(\pi) = \{P' | P \in \mathcal{P}_{\rho_3}(\pi_3)\}$.

This way it is possible to add the unary equality rules to the implementation and treat them implicitly as their binary version.

3.2.2 GAPT Integration

Earlier versions of GAPT used the binary rules, but they have recently been replaced by their unary equivalents. There are still input proofs with binary equality which now use the simulation via unary rules. In particular this holds for the LLK input format (see section 3.12) which predates this development. We will have a short look at the impact on the CERES method. Since we defined the characteristic sequent set and the projections via their simulation by the binary inferences, we actually compute the corresponding sets as if each binary inference

$$\dfrac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ \Gamma_1 \vdash s = t, \Delta_1 & \Gamma_2 \vdash F(s), \Delta_2\end{array}}{\Gamma_1, \Gamma_2 \vdash F(t), \Delta_1, \Delta_2}\ =: r$$

were replaced by the pattern:

$$\dfrac{\begin{array}{cc}(\pi_1) & \dfrac{\dfrac{\dfrac{}{s = t \vdash s = t} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F(s), \Delta_2}}{\Gamma_2, s = t \vdash F(s), \Delta_2}\ w : l}{\Gamma_2, s = t, s = t \vdash F(t), \Delta_2}\ =: r\\ \Gamma_1 \vdash s = t, \Delta_1 & \dfrac{\Gamma_2, s = t, s = t \vdash F(t), \Delta_2}{\Gamma_2, s = t \vdash F(t), \Delta_2}\ c : l\end{array}}{\Gamma_1, \Gamma_2 \vdash F(t), \Delta_1, \Delta_2}\ cut$$

If we assume, the characteristic sequent sets $\mathcal{C}(\pi_1)$ and $\mathcal{C}(\pi_2)$ have been constructed, we distinguish again on the cut-ancestorship of $F[t]$.

- $F[t]$ is a cut-ancestor: Then the resulting set for the original inference is $\mathcal{C}(\pi_1) \cup \mathcal{C}(\pi_2)$ and the one for the double-simulation is also $\mathcal{C}(\pi_1) \cup \mathcal{C}(\pi_2)$ ($s = t \vdash s = t$ is a tautology).
- $F[t]$ is an end-sequent ancestor: Then the resulting set for the original inference is $\mathcal{C}(\pi_1) \times \mathcal{C}(\pi_2)$. But since the $s = t$ in the end-sequent of π_1 is now a $cut - ancestor$ independent of the cut-ancestorship of $F[t]$, the characteristic sequent set is different. If we call this set $\mathcal{C}'(\pi_1)$ then the resulting set of the double-simulation is $\mathcal{C}'(\pi_1) \cup (s = t \vdash \times\, \mathcal{C}(\pi_2))$.

To compare $\mathcal{C}(\pi_1)$ and $\mathcal{C}'(\pi_1)$, we see that $s = t$ is atomic and therefore only other equality rules or contractions can be applied to ancestors of the this equation. In the case that the occurrence of $s = t$ is linear (i.e. no inference operates on its ancestors), $s = t$ will occur as an additional consequent in some of the sequents in $\mathcal{C}'\pi_1$. The intuition is that in these cases, the additional succedent occurrences of $s = t$ in $\mathcal{C}'(\pi_1)$ can be resolved with the additional antecedent occurrence of $s = t$ in $s = t \vdash \times\, \mathcal{C}(\pi_2)$ to obtain $\mathcal{C}(\pi_1) \times \mathcal{C}_{\pi_2}$ directly or a consequence of it. Then, similarly, multiple equality inferences on ancestors of $s = t$ would just add additional literals to resolve on. Currently, it is still unclear how to exactly prove this intuition so we will leave it as a conjecture:

Conjecture 3.2.4. Let π be an LK proof with binary equality equality rules and let π' be the proof obtained by translating the binary equality rules to unary ones. Then $\mathcal{C}(\pi')$ is refutable if $\mathcal{C}(\pi)$ is refutable.

3.3 Some Properties of Skolem Expansion Trees

We will prove a few lemmas about skolem expansion trees, which will be useful in section 3.5 where we show the validity of skolem expansion trees extracted from LK_{skc} proofs with propositional and passive cuts. The first can be seen as an idempotence lemma for the merge operation. Even though merging two copies of an expansion tree does not result in the exact same tree because of the merge of weak quantifier nodes, we can still prove that their deep formulas are equivalent.

Lemma 3.3.1. Let A be a (skolem) expansion tree. Then $\mathrm{Dp}(A) \Leftrightarrow \mathrm{Dp}(merge(A, A))$.

Proof. We proceed by induction on the structure of A:

- A is an atom node: Then $merge(A, A) = A$ and therefore also $\mathrm{Dp}(A) \Leftrightarrow \mathrm{Dp}(merge(A, A))$ holds.
- A is a negation node $\neg B$ or a binary logical node $B \circ C$ with $\circ \in \{\wedge, \vee, \rightarrow\}$:
  By definition of merge, $merge(\neg B, \neg B) = \neg merge(B, B)$ and $merge(B \circ C, B \circ C) = merge(B, B) \circ merge(C, C)$ respectively. By IH we can assume that $\mathrm{Dp}(merge(B, B)) \Leftrightarrow \mathrm{Dp}(B)$ and $\mathrm{Dp}(merge(C, C)) \Leftrightarrow \mathrm{Dp}(C)$. Then also $\neg\mathrm{Dp}(merge(B, B)) \Leftrightarrow \neg\mathrm{Dp}(B)$ and $\mathrm{Dp}(merge(B, B)) \circ \mathrm{Dp}(merge(C, C)) \Leftrightarrow \mathrm{Dp}(B) \circ \mathrm{Dp}(C)$ hold.
- A is a strong (skolem) quantifier node $QF +^s +B$:
  By definition of merge, $merge(QF +^s B, QF +^s B) = QF +^s merge(B, B)$ and again by IH we can assume that $\mathrm{Dp}(merge(B, B)) \Leftrightarrow \mathrm{Dp}(B)$. Since by definition of the deep formula, $\mathrm{Dp}(QF +^s B) = \mathrm{Dp}(B)$ holds, the equivalence again extends to $\mathrm{Dp}(QF +^s B) \Leftrightarrow \mathrm{Dp}(merge(QF +^s +B, QF +^s +B))$.
- A is a weak quantifier node $QF +^{t_1} B_1 + \ldots +^{t_n} B_n$: then $merge(A, A) = QF +^{t_1} B_1 + \ldots +^{t_n} B_n +^{t_1} B_1 + \ldots +^{t_n} B_n$. Now $\mathrm{Dp}(QF +^{t_1} B_1 + \ldots +^{t_n} B_n +^{t_1} B_1 + \ldots +^{t_n} B_n) = \mathrm{Dp}(B_1) \circ \ldots \mathrm{Dp}(B_n) \circ \mathrm{Dp}(B_1) \circ \ldots \mathrm{Dp}(B_n)$ where $\circ$ stands for a disjunction if A occurs positively and where it stands for a conjunction if A occurs negatively. Both operators are associative, commutative and idempotent, allowing us to reduce the formula to $\mathrm{Dp}(B_1) \circ \ldots \mathrm{Dp}(B_n)$ which is equivalent to $\mathrm{Dp}(QF +^{t_1} B_1 + \ldots +^{t_n} B_n)$.

□

Also, the associativity of disjunction and conjunction transfers to the merge operation:

Lemma 3.3.2. Let A, B, C be expansion trees such that $\mathrm{Sh}(A) = \mathrm{Sh}(B) = \mathrm{Sh}(C)$. Then $\mathrm{Dp}(merge(A, merge(B, C))) = \mathrm{Dp}(merge(merge(A, B), C))$.

Proof. We proceed again by structural induction. The only non-trivial case is the merge of weak quantifiers:

- Let $A = QF +^{r_1} R_1 + \ldots +^{r_l} R_l$, $B = QF +^{s_1} +S_1 \ldots +^{s_m} S_m$, , $C = QF +^{t_1} +T_1 \ldots +^{t_n} T_n$. Then $\mathrm{Dp}(merge(A, merge(B, C))) = \bigvee_{i=1}^{l} R_i \vee (\bigvee_{j=1}^{m} S_j \vee \bigvee_{k=1}^{n} T_k) = (\bigvee_{i=1}^{l} R_i \vee \bigvee_{j=1}^{m} S_j) \vee \bigvee_{k=1}^{n} T_k = \mathrm{Dp}(merge(merge(A, B), C))$.

□

Furthermore, we can show that the merge operation is similar to weakening: the deep formula of an expansion tree A implies the deep formula of A merged with some expansion tree B.

Lemma 3.3.3. Let A, B be positive occurrences of (skolem) expansion trees with $\mathrm{Sh}(A) = \mathrm{Sh}(B)$. Then $\mathrm{Dp}(A) \to \mathrm{Dp}(merge(A, B))$. If A, B are negative occurrences, then $\neg\mathrm{Dp}(A) \to \neg\mathrm{Dp}(merge(A, B))$.

Proof. We proceed by induction on the stucture of A, but only the weak quantifier case is non-trivial:

- A is a weak quantifier node $QF +^{t_1} A_1 + \ldots +^{t_n} A_n$ and B is a weak quantifier node $QF +^{t_1} B_1 + \ldots +^{t_n} B_m$:
  Then $merge(A, B) = QF +^{t_1} A_1 + \ldots +^{t_n} A_n +^{t_1} B_1 + \ldots +^{t_n} B_m$ and $\mathrm{Dp}(merge(A, B)) = \mathrm{Dp}(A_1) \circ \ldots \circ \mathrm{Dp}(A_n) \circ \mathrm{Dp}(B_1) \circ \ldots \circ \mathrm{Dp}(B_m) \Leftrightarrow \mathrm{Dp}(A) \circ \mathrm{Dp}(B)$ where $\circ \in \{\vee, \wedge\}$, depending on the polarity of A. If A occurs positively, this leads to $\mathrm{Dp}(A) \to (\mathrm{Dp}(A) \vee \mathrm{Dp}(B))$, if A occurs negatively, this leads to $\neg\mathrm{Dp}(A) \to \neg(\mathrm{Dp}(A) \wedge \mathrm{Dp}(B))$. Both cases are instances of weakening and are easily checked for validity.

□

The relationship between disjunction and the merge operation is not immediately obvious. The expansion of a positive occurence of $\exists F +^s F_s +^t F_t$ to $F_s \vee F_t$ would hint that they are equivalent, but merge is in fact a stronger operation than disjunction: if we consider the two trees $T_1 = (\exists X X +^{\bot} \bot) \wedge (\exists X X +^{\top} \top)$ and $T_2 = (\exists X X +^{\top} \top) \wedge (\exists X X +^{\bot} \bot)$, we see that $\mathrm{Dp}(merge(T_1, T_2)) = (\bot \vee \top) \wedge (\top \vee \bot) = \top$ but $\mathrm{Dp}(T_1 \vee T_2) = (\bot \wedge \top) \vee (\top \wedge \bot) = \bot$.

The converse direction holds though:

Lemma 3.3.4. Let A, B expansion trees of positive polarity such that $\mathrm{Sh}(A) = \mathrm{Sh}(B)$. Then $\mathrm{Dp}(merge(A, B)) \to (\mathrm{Dp}(A) \vee \mathrm{Dp}(B))$. If they are of negative polarity, then $\neg\mathrm{Dp}(merge(A, B)) \to \neg(\mathrm{Dp}(A) \wedge \mathrm{Dp}(B))$.

Proof. We proceed by induction on the shape of the shallow formula and distinguish on the polarity:

- Positive polarity:
  - Atom F: then $\mathrm{Dp}(A) = \mathrm{Dp}(B) = F$ and we have to check $F \to F$, which trivially holds.
  - Negation $\neg F$: since the polarity of $\neg F$ is positive, the polarity of F is negative. Let the expansion trees A and B have the shape $\neg C$ and $\neg D$. Then we have to check $\mathrm{Dp}(merge(\neg C, \neg D)) \to (\mathrm{Dp}(\neg C) \vee \mathrm{Dp}(\neg D))$ which unfolds to $\neg\mathrm{Dp}(merge(C, D)) \to (\neg(\mathrm{Dp}(C) \wedge \neg\mathrm{Dp}(D)))$. But since the complexity of $\mathrm{Sh}(C)$ is less than the one of $\mathrm{Sh}(\neg C)$, the IH gives us exactly the formula in question.

- Conjunction $F \wedge G$: let us call F_T and G_T the expansion trees corresponding to F and G. Then we unfold the definition of deep formulas from $\mathrm{Dp}(F_T \wedge G_T)$ to $\mathrm{Dp}(F_T) \wedge \mathrm{Dp}(G_T)$. Then $\mathrm{Dp}(F_T) \wedge \mathrm{Dp}(G_T) \to (\mathrm{Dp}(F_T) \vee \mathrm{Dp}(G_T))$.
- Disjunction $F \vee G$: similar to the conjunctive case, we need to check that $(\mathrm{Dp}(F_T) \vee \mathrm{Dp}(G_T)) \to (\mathrm{Dp}(F_T) \vee \mathrm{Dp}(G_T))$ holds, which again is trivial.
- Weak quantifier $\exists F$: let the shape of A be $\exists F +^{t_1} C_1 + \ldots +^{t_n} + C_n$ and the shape of B be $\exists F +^{t_1} D_1 + \ldots +^{t_n} + D_n$ with $\mathrm{Sh}(C_i) = \mathrm{Sh}(D_i)$ for $i \in \{1, \ldots, n\}$ with common instantiation terms t_i. Then we need to show that $\mathrm{Dp}(\exists F +^{t_1} C_1 + \ldots +^{t_n} + C_n +^{t_1} D_1 + \ldots +^{t_n} + D_n) \to (\mathrm{Dp}(\exists F +^{t_1} C_1 + \ldots +^{t_n} + C_n) \vee \mathrm{Dp}(\exists F +^{t_1} D_1 + \ldots +^{t_n} + D_n)$. Unfolding the deep formula, we obtain $(\mathrm{Dp}(C_1) \vee \ldots \vee \mathrm{Dp}(C_n) \vee \mathrm{Dp}(D_1) \vee \ldots \vee \mathrm{Dp}(D_n)) \to ((\mathrm{Dp}(C_1) \vee \ldots \vee \mathrm{Dp}(C_n)) \vee (\mathrm{Dp}(D_1) \vee \ldots \mathrm{Dp}(D_n)))$ which is valid by the associativity of disjunction.
- Strong quantifier $\forall F$: let the shape of A be $\forall F +^s C$ and the shape of B be $\forall F +^s D$. We need to show that $\mathrm{Dp}(merge(\forall F +^s C, \forall F +^s D)) \to (\mathrm{Dp}(\forall F +^s C) \vee \mathrm{Dp}(\forall F +^s D))$ which unfolds to $\mathrm{Dp}(merge(C, D)) \to (\mathrm{Dp}(C) \vee \mathrm{Dp}(D))$ by the definition of merge and deep formulas. Since the complexity of $\mathrm{Sh}(C)$ is less than the complexity of $\forall F +^s C$, we can invoke the IH which again gives us this implication.

- Negative polarity:
  The cases are symmetrical to the positive case.

□

As a kind of corrolary of lemma 3.3.4, we can show that permuting merge over a logical operator strengthens the deep formula of an expansion tree.

Lemma 3.3.5. Let A, B, C, D be expansion trees such that $\mathrm{Sh}(A) = \mathrm{Sh}(C)$ and $\mathrm{Sh}(B) = \mathrm{Sh}(D)$. Then for all operators $\circ$, the implications $\mathrm{Dp}(\circ merge(A, C)) \to \mathrm{Dp}(merge(\circ A, \circ C))$ and $\mathrm{Dp}(merge(A, C) \circ merge(B, D)) \to \mathrm{Dp}(merge(A \circ C, B \circ D))$ hold.

Proof.
We distinguish on the node $\circ$:

- Negation: by the definition of merge, $merge(\neg A, \neg C) = \neg merge(A, C)$ and the implication is trivially true.
- Conjunction, Disjunction, Implication: similarly, by the definition of merge, $merge(A \circ C, B \circ D) = merge(A, C) \circ merge(B, D)$ in these cases.
- Strong quantifier node: also $merge(QF +^s G_1, QF +^s G_2) = QF +^s merge(G_1, G_2)$ which transfers to an implication of the deep formulas.
- Weak quantifier node: w.l.o.g. assume the quantifier is existential and we are in a positive context. Since $\mathrm{Sh}(A) = \mathrm{Sh}(C)$, the instantiation term t must be the same in both cases. We therefore want to show $\mathrm{Dp}(\exists F^t merge(A, C)) \to \mathrm{Dp}(merge(\exists F +^t A, \exists F +^t C))$.

By definition of the deep formula and merge, this amounts to $\mathrm{Dp}(merge(A, C)) \rightarrow (\mathrm{Dp}(A) \vee \mathrm{Dp}(C))$ which we proved in lemma 3.3.4.

□

Unfortunately, the restrictions on the weak quantifier enforce a its application in a restricted context. Nevertheless, both lemma 3.3.4 and lemma 3.3.5 will be vital in the next section.

3.4 Extracting Expansion Proofs from LK_{sk} Proofs

We now consider the extraction of skolem expansion proofs from the cut-free calculus LK_{sk}. The algorithm is similar to Miller's (see section 2.3.1) but skolem quantifier inferences produce skolem quantifier nodes in the extracted tree. Since the notion of a skolem term depends on the global structure of the tree, sub-trees of skolem expansion trees are not necessarily skolem expansion trees themselves. We therefore extract a tree of skolem expansion nodes first and show later that an extraction from an LK_{sk} proof really provides a skolem expansion tree. To explicitly track weakening in expansion sequents, we also introduce a weakening node:

Definition 3.4.1. Let F be a formula, then $\mathbf{W}{+}F$ is an expansion tree with $\mathrm{Sh}(\mathbf{W}{+}F) = F$. If the tree occurs positively then $\mathrm{Dp}(\mathbf{W}{+}F) = \bot$, if it occurs negatively then $\mathrm{Dp}(\mathbf{W}{+}F) = \top$.

Definition 3.4.2. Let π be an LK_{sk} tree ending in the inference ρ. We distinguish on the inference type of ρ:

- ρ is a strong skolem quantifier inference: w.l.o.g. suppose the sequent $\Gamma \vdash \langle \forall F \rangle^{\ell}, \Delta$ was inferred from the sequent $S : \Gamma \vdash \langle F(f(S_1, \ldots, S_n)) \rangle^{\ell}$ with $\ell = S_1, \ldots, S_n$ and that $\Lambda \vdash Q, \Pi$ is the tree for S. Then the tree for π is $\Lambda \vdash \forall\ F +^{fS_1 \ldots S_n} Q, \Pi$.
- ρ is a weak skolem quantifier inference: w.l.o.g. suppose the sequent $\Gamma \vdash \langle \exists F \rangle^{\ell}, \Delta$ was inferred from the sequent $S : \Gamma \vdash \langle FT \rangle^{\ell,T}$ and that $\Lambda \vdash Q, \Pi$ is the tree for S. Then the tree for π is $\Lambda \vdash \exists\ F +^{T} Q, \Pi$.
- ρ is an axiom rule, a logical rule or a contraction rule: Then we follow the pattern from section 2.3.1.
- ρ is a weakening rule: suppose the expansion sequent of the premise is $\Gamma \vdash \Delta$ and the formula F is introduced by a weakening to the right. Then the expansion sequent of the conclusion is $\Gamma \vdash \Delta, \mathbf{W}{+}F$. The weakening to the left rule is symmetrical.

We call the function implementing this algorithm $extract(\pi)$.

We also extend the definition of the merge operation (see definition 2.3.8):

Definition 3.4.3. Let T_a and T_b be expansion trees with $Sh(T_a)$ is α-equal to $Sh(T_b)$. We define $merge(T_a, T_b)$ by distinguishing (T_a, T_b) on their (identical) root node:

- Strong skolem quantifier node $(Q\ F{+}^{fS_1 \ldots S_n}, Q\ F{+}^{fS_1 \ldots S_n})$:
  then $merge(Q\ F{+}^{fS_1 \ldots S_n}, Q\ F{+}^{fS_1 \ldots S_n}) = Q\ F{+}^{fS_1 \ldots S_n}$.

- Weak skolem quantifier node $(Q\ F +^{t_1,\ldots,t_m} T'_a, Q\ F +^{t_{m+1},\ldots,t_n} T'_b)$:
  then $merge(Q\ F +^{t_1,\ldots,t_m} T'_a, Q\ F +^{t_{m+1},\ldots,t_n} T'_b) = Q\ F +^{t_1,\ldots,t_n} merge(T'_a, T'_b)$.

- Weakening node:
  then $merge(T, \mathbf{W}{+}F) = merge(\mathbf{W}{+}F, T) = T$. [7]

- Atom or Logical Connective node: this is the same as in definition 2.3.8.

The definition is incomplete in the sense that the merge of strong quantifier nodes is only defined when the skolem terms agree. This is similar to the merge of quantifier nodes with selected variables. For that case, Miller mentions that two expansion proofs Q_1 and Q_3 in a contraction can be picked such that they can be merged.[8] In our case, we can peform the renaming on the LK_{sk} proof, because weak regularity is preserved and sequential pruning coincides with the merge operation: both are only performed on contracted inferences which are skolem parallel. For instance, the proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle P(s)\rangle^{x,y} \vdash P(s)}{\langle P(s)\rangle^{x,y} \vdash \forall x P(x)}\,\forall^{sk}:r \quad \dfrac{\langle P(t)\rangle^{x,y} \vdash P(t)}{\langle P(t)\rangle^{x,y} \vdash \forall x P(x)}\,\forall^{sk}:r}{\langle P(s) \vee P(t)\rangle^{x,y} \vdash \forall x P(x), \forall x P(x)}\,\vee:l}{\langle \forall y(P(s) \vee P(y))\rangle^{s} \vdash \forall x P(x), \forall x P(x)}\,\forall^{sk}:l}{\forall x \forall y(P(x) \vee P(y)) \vdash \forall x P(x), \forall x P(x)}\,\forall^{sk}:l}{\forall x \forall y(P(x) \vee P(y)) \vdash \forall x P(x)}\,c:r}{(\pi)}$$

has two skolem symbols s and t which were introduced by two $\forall_{sk} : r$ rules. Later on the inferred formula $\forall x P(x)$ is contracted (the homomorphic part of the path is colored blue and the non-homomorphic path is colored purple). Since the two skolem symbols are different, there is no sequential homomorphic pair in π. Renaming s to t allows us to perform sequential pruning, where the $\rhd_c$ relation pushes the contraction as close to the root as possible:

[7] The tree chosen is deterministic: in the case of a merge of two weakening nodes $merge(\mathbf{W}{+}F, \mathbf{W}{+}G)$, both trees need to agree on their shallow formula. But if $\mathrm{Sh}(\mathbf{W}{+}F) = \mathrm{Sh}(\mathbf{W}{+}G)$, then also $F = G$ and the nodes are identical.

[8] The relevant part is in the proof of case 1 in Lemma 3.13 of Miller:"A compact representation for proofs". Given a grounded expansion proof for $A \vee B$ and for $C \vee B$, we are looking for a grounded expansion proof for $(A \wedge C) \vee B$. Miller writes:

> Thus, assume that $A \vee B$ and $C \vee B$ have grounded ET-proofs $Q_1 \vee Q_2$ and $Q_3 \vee Q_4$. These expansion trees can be picked so that Q_1 and Q_3 can be merged to obtain Q_5 and that $Q := Q_5 \vee [Q_2 \wedge Q_4]$ contains no variable selected twice. Clearly, Q is a grounded expansion tree for $S \vee [A \wedge B]$.

Remark: There seems to be a mixup with the indices, because Q_1 and Q_3 can only be merged if they have the same shallow formula, i.e. if they are expansion proofs for B. Also the formulas are called A, B, C, not A, B, S in the statement of the lemma. If we infer $S \vee (A \wedge B)$ from $S \vee A$ and $S \vee B$, everything works as expected.

A merge of strong quantifiers is only defined for the same selected variable. "These expansion trees can be picked" then refers to renaming the selected variable accordingly.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle P(t)\rangle^{x,y} \vdash P(t)}{\langle P(t)\rangle^{x,y} \vdash \forall x P(x)}\,\forall^{sk}:r \quad \dfrac{\langle P(t)\rangle^{x,y} \vdash P(t)}{\langle P(t)\rangle^{x,y} \vdash \forall x P(x)}\,\forall^{sk}:r}{\langle P(t) \vee P(t)\rangle^{x,y} \vdash \forall x P(x), \forall x P(x)}\,\vee:l}{\langle \forall y(P(t) \vee P(y))\rangle^{t} \vdash \forall x P(x), \forall x P(x)}\,\forall^{sk}:l}{\forall x \forall y(P(x) \vee P(y)) \vdash \forall x P(x), \forall x P(x)}\,\forall^{sk}:l}{\forall x \forall y(P(x) \vee P(y)) \vdash \forall x P(x)}\,c:r$$

The expansion trees for the auxiliary $\forall x P(x)$ occurrences in the contraction are both $T_1 = T_2 = \forall x^t P(x)$ (with the deep formula $P(t)$). Afterwards, the duplicated paths are zipped up, leading to the proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\langle P(t)\rangle^{x,y} \vdash P(t) \quad \langle P(t)\rangle^{x,y} \vdash P(t)}{\langle P(t) \vee P(t)\rangle^{x,y} \vdash P(t), P(t)}\,\vee:l}{\langle P(t) \vee P(t)\rangle^{x,y} \vdash P(t)}\,c:r}{\langle P(t)\rangle^{x,y} \vdash \forall x P(x)}\,\forall^{sk}:r}{\langle \forall y(P(t) \vee P(y))\rangle^{t} \vdash \forall x P(x)}\,\forall^{sk}:l}{\forall x \forall y(P(x) \vee P(y)) \vdash \forall x P(x)}\,\forall^{sk}:l$$

Sequential pruning merged the paths up to the strong quantifier inference, which also happens during $merge(T_1, T_2)$. Since such a renamed LK_{sk} proof is not regular, we will call it strict weak regularity.

Definition 3.4.4. Let π be a strict weak regular LK_{sk} proof. We call π *strictly weakly regular*, for all distinct strong Skolem quantifier inferences ρ_1, ρ_2 holds: if ρ_1 and ρ_2 are homomorphic, they have the same Skolem terms.

Since we can perform this renaming before the extraction, we only need to show that strict weak regularity is sufficient to perform the operation.

Lemma 3.4.5. Let π be a weakly regular LK_{sk} proof. Then $extract(\pi)$ is a skolem expansion tree.

Proof. We recall the two global properties in the definition of skolem expansion trees (see definition 2.3.11):

1. Each skolem quantifier node introduces a unique skolem function f.
2. The path from the root to a skolem quantifier node contains exactly p weak quantifier nodes with expansion terms t_1 to t_p (in that order).

Property 1 is fulfilled in a strictly weakly regular proof: since by definition, homomorphic quantifier inferences have a uniting contraction, their corresponding nodes will be merged into one during the application of $extract$. Furthermore, all non-homomorphic quantifier inferences

introduce a unique skolem function. Therefore, each quantifier node introduces a unique skolem function.

Property 2 is fulfilled since an LK_{sk} proof is proper and each label corresponds to a weak skolem quantifier inference: let $\omega : \langle \forall F \rangle^{S_1,\ldots,S_n}$ be the labeled formula occurrence inferred from the skolem term $f(S_1,\ldots,S_n)$ in a strong quantifier inference and let μ be a path from its descendant in the end-sequent to ω. Then $S_1,\ldots,S_n$ are exactly the instantiation terms of the weak quantifier rules with active formulas in μ. We prove this by induction on the number $i \in \{0,\ldots,n\}$ of labels: For $i = 0$ the list of labels must be empty. This is the case at the root because π is an LK_{sk} proof and therefore proper. The only rule which increases n is a weak quantifier skolem quantifier rule inferring an occurrence on μ. By IH let us assume the list $\ell = S_1,\ldots,S_{i-1}$ has been correctly created. Since a weak quantifier rule always extends the label by its instantiation term S_i, we can extend the list to ℓ, S_i. This result carries over to the skolem expansion tree, since $extract(\pi)$ preserves the order of quantifier inferences. Furthermore each weak quantifier in the LK_{sk} tree acting on an occurrence on μ maps to a distinct quantifier in the corresponding skolem expansion tree. Therefore the arguments of the skolem quantifier node Q corresponding to ω are exactly the instantiation terms of the weak quantifier nodes between the root of the tree and Q. □

Since we extract a skolem expansion tree from an LK_{sk} proof, we would expect that this is actually a skolem expansion proof.

Conjecture 3.4.6. Let T be a skolem expansion tree obtained by applying $extract(\pi)$ to a weakly regular LK_{sk} proof π. Then $\mathrm{Dp}(T)$ is a tautology and T is a skolem expansion proof.

If we are in the context of the functional fragment of CERES^{ω}, the situation is simple: already in the LK input proof, we can expand axioms introducing formulas to proofs with atomic introduction rules only. Then the characteristic sequent set does contain neither predicate variables nor non-atomic formulas in any sequent. Therefore a refutation of the characteristic sequent set does not contain any quantifier inferences and a simulation of this proof does not add any quantifier inferences either. Furthermore, the projection transformation, the simulation and reductive cut-elimination preserve the order of quantifiers such that they are still the same as in the input proof. But the input proof respects the eigenvariable condition, therefore all the quantifiers in the resulting LK_{skc} proof are correctly placed and preserved through the whole method.

For the general case, it seems likely that we could generate the deep formula, or at least an equivalent one, just by skipping the quantifier inferences in an LK_{sk} proof. However, the presence of contraction inferences complicates matters: we could replace the contraction of two existential inferences $\exists x P(x)$ as a disjunction $P(t_1) \vee P(t_2)$ of the auxiliary formulas. But as we have seen in section 3.3, the counter-example to $\mathrm{Dp}(T_1) \vee \mathrm{Dp}(T_2) \rightarrow \mathrm{Dp}(merge(T,T))$ shows that the merge operation is stronger than a disjunction. Therefore, this approach does not work.

In a way, the proof rewriting systems from section 2.6.4 used to put quantifier inferences into place performs this strengthening. We will show for each rule that the deep formula of each reduct implies the deep formula of its redex. Then, the validity of the final skolem expansion sequent implies the validity of the extracted skolem expansion sequent.

Lemma 3.4.7. Let $\pi \rhd_c^1 \pi'$ be a contraction shift. Then $\mathrm{Dp}(extract(\pi')) \to \mathrm{Dp}(extract(\pi))$.

Proof. We distinguish between σ being a unary or primary formula when applying $\rhd_c^1$ (see definition 2.6.27).

- Unary rule: let $S : \Pi_T, \Pi_T^*, \Gamma_T \vdash \Delta_T, \Lambda_T, \Lambda_T^*$ be the expansion sequent extracted from the parent proof common to π and π'. Since the expansion trees of the duplicated occurrences of Π may lead to different expansion trees, we denote the second version with a star (*).

  Moreover, the permuted inferences are independent and the extraction algorithm only takes ancestor inferences of a formula occurrence into account. We can therefore safely distinguish between the extraction of the contracted formulas Π and Λ and the formulas Γ and Δ affected by the inference σ.

  Then the extracted expansion proof for π is $\Pi'_T, \Gamma'_T \vdash \Delta'_T, \Lambda'_T$ where Γ'_T and Δ'_T are identical to Γ_T and Δ_T up to the primary formula F of σ. In case σ is a contraction, in both cases the expansion tree for the primary formula is $merge(F_1, F_2)$, in the case of a logical operator $\circ$, it is an expansion tree of the form $\circ F_1$.

  The contracted trees Π'_T and Λ'_T for both π and π' translate to the same merges between trees in Π_T and Π_T^* respectively to merges between trees in Λ_T and Λ_T^*.

  Therefore, with identical expansion sequents, we surely also have $\mathrm{Dp}(extract(\pi')) \to \mathrm{Dp}(extract(\pi))$.

- Binary rule: again let $S_1 : \Pi_T, \Pi_T^*, \Gamma_T \vdash \Delta_T, \Lambda_T, \Lambda_T^*$ and $S_2 : \Sigma_T \vdash \Theta_T$ be the expansion sequents corresponding to the parent proofs common in π and π'. Similar to the unary case, we can use the independence of the operation to argue separately on the contracted formulas and the primary formula of σ. Again, we end up with the same merged expansion trees in Π_T, Λ_T and Π_T^*, Λ_T^* as well as binary node $\circ$ applied on the auxiliary trees in Σ_T and Θ_T, ending up with identical expansion sequents for $extract(\pi)$ and $extract(\pi')$. The implication of the deep formulas follows trivially. The symmetrical case where S_1 and S_2 are flipped works alike.

□

Lemma 3.4.8. Let π' be an LK_{sk} proof obtained from an LK_{sk} proof π by applying one of the rules in definition 2.6.29. Then $\mathrm{Dp}(extract(\pi)) \to \mathrm{Dp}(extract(\pi'))$.

Proof. We distinguish between the outermost symbol of the contracted formula.

- The outermost symbol is a unary operator $\circ$:
  Then the rewrite rule in question looks as follows:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash \Delta, \langle F\rangle^{\ell_1}}{\Gamma \vdash \Delta, \langle G\rangle^{\ell_2}}\,\rho}{\vdots}}{\dfrac{\dfrac{\Gamma' \vdash \Delta', \langle F\rangle^{\ell_1}, \langle G\rangle^{\ell_2}}{\Gamma' \vdash \Delta', \langle G\rangle^{\ell_2}, \langle G\rangle^{\ell_2}}\,\rho'}{\vdots}}{\dfrac{\Gamma^* \vdash \Delta^*, \langle G\rangle^{\ell_2}, \langle G\rangle^{\ell_2}}{\Gamma^* \vdash \Delta^*, \langle G\rangle^{\ell_2}}\,c} \quad \rightsquigarrow \quad \dfrac{\Gamma \vdash \Delta, \langle F\rangle^{\ell_1}}{\dfrac{\vdots}{\dfrac{\dfrac{\Gamma' \vdash \Delta', \langle F\rangle^{\ell_1}, \langle F\rangle^{\ell_1}}{\Gamma' \vdash \Delta', \langle F\rangle^{\ell_1}}\,c}{\dfrac{\Gamma' \vdash \Delta', \langle G\rangle^{\ell_2}}{\vdots}\,\rho'}}}{\Gamma^* \vdash \Delta^*, \langle G\rangle^{\ell_2}}$$

The inferences on ancestors of the context Γ^*, Δ^* are independent of the skipped inference ρ as well as the contractions and ρ'. For these, the extracted expansion sequent $extract(\Gamma^* \vdash \Delta^*)$ is identical for π and π' such that also the deep formulas agree.

Suppose F_T is the expansion tree corresponding to $\langle F\rangle^{\ell_1}$ and F_T^* the one corresponding to $\langle F\rangle^{\ell_2}$. Then the expansion tree for the occurrence of $\langle G\rangle^{\ell_2}$ in the end-sequent of π is $merge(\circ F_T, \circ F_T^*)$. Furthermore the expansion tree for the occurrence of $\langle G\rangle^{\ell_2}$ in the end-sequent of π' is $\circ merge(F_T, F_T^*)$. Now, we can apply lemma 3.3.5: for all operators but the weak quantifiers, $merge(\circ A, \circ B) = \circ merge(A, B)$, such that the deep formulas agree. If $\circ$ is a weak quantifier, the fact that both inferences are homomorphic ensure that it was inferred from the same formula F and instantiation term t, such that we obtain $merge(G +^t F_T, G +^t F_T^*)$ from π and $G +^t merge(F_T, F_T^*)$ from π'.

Furthermore suppose that F and F^* occur positively, the negative case is analogous, where disjunction is replaced by conjunction. Then by unfolding the definition of deep formulas, we see that $\mathrm{Dp}(merge(G +^t F_T, G +^t F_T^*)) = \mathrm{Dp}(F_T) \vee \mathrm{Dp}(F_T^*)$. Now by lemma 3.3.4, we know that $\mathrm{Dp}(merge(F_T, F_T^*)) \to (\mathrm{Dp}(F_T) \vee \mathrm{Dp}(F_T^*))$ such that the implication also holds in this case.

- The outermost symbol is a binary operator:
  Then the rewrite rule looks as follows:

$$\dfrac{\dfrac{\dfrac{\langle F\rangle^{\ell}, \Gamma \vdash \Delta \qquad \langle G\rangle^{\ell}, \Pi \vdash \Lambda}{\langle F \vee G\rangle^{\ell}, \Gamma, \Pi \vdash \Delta, \Lambda}\,\rho}{\vdots}}{\dfrac{\dfrac{\langle F\rangle^{\ell}, \langle F \vee G\rangle^{\ell}, \Gamma^* \vdash \Delta^* \qquad \langle G\rangle^{\ell}, \Pi^* \vdash \Lambda^*}{\langle F \vee G\rangle^{\ell}, \langle F \vee G\rangle^{\ell}, \Gamma^*, \Pi^* \vdash \Delta^*, \Lambda^*}\,\rho'}{\vdots}}{\dfrac{\langle F \vee G\rangle^{\ell}, \langle F \vee G\rangle^{\ell}, \Gamma^+ \vdash \Delta^+}{\langle F \vee G\rangle^{\ell}, \Gamma^+ \vdash \Delta^+}\,c : l} \quad \rightsquigarrow \quad \dfrac{\dfrac{\langle F\rangle^{\ell}, \Gamma \vdash \Delta}{\langle F\rangle^{\ell}, \Gamma, \Pi \vdash \Delta, \Lambda}\,w : *}{\dfrac{\vdots}{\dfrac{\dfrac{\langle F\rangle^{\ell}, \langle F\rangle^{\ell}, \Gamma^* \vdash \Delta^*}{\langle F\rangle^{\ell}, \Gamma^* \vdash \Delta^*}\,c \qquad \langle G\rangle^{\ell}, \Pi^* \vdash \Lambda^*}{\langle F \vee G\rangle^{\ell}, \Gamma^*, \Pi^* \vdash \Delta^*, \Lambda^*}\,\rho'}{\vdots}}{\langle F \vee G\rangle^{\ell}, \Gamma^+ \vdash \Delta^+}$$

  Suppose the expansion sequents of the common parent proofs are $F_T^1, \Gamma_T \vdash \Delta_T$ and $G_T^2, \Pi_T^* \vdash \Lambda_T^*$. Furthermore, suppose the expansion sequent for the additional parent proof is $G_T^1, \Pi_T \vdash \Lambda_T$ and the tree corresponding to the auxiliary occurrence of F in ρ' of π is F_T^2.

  Again, independence allows to have a separate argumentation for the primary formula and a context formula of the end-sequent:

– Primary formula: the expansion tree for the primary formula in π is then $merge(F_T^1 \vee G_T^1, F_T^2 \vee G_T^2) = merge(F_T^1, F_T^2) \vee merge(G_T^1, G_T^2)$ and the expansion tree for the primary formula in π' is $merge(F_T^1, F_T^*) \vee G_T^2$ where F_T^* depends on the ancestorship of the auxiliary F in ρ:

  * F has an ancestor in Γ or Δ: then F_T^* is identical to F_T^2 and we need to check $\neg\mathrm{Dp}(merge(F_T^1, F_T^2) \vee G_T^2) \to \neg\mathrm{Dp}(merge(F_T^1, F_T^2) \vee merge(G_T^1, G_T^2))$ which simplifies to $\neg\mathrm{Dp}(G_T^2) \to \neg\mathrm{Dp}(merge(G_T^1, G_T^2))$. But since the occurrences of G_T^1 and G_T^2 are negative, this is exactly an instance of lemma 3.3.3.
  * F has an ancestor in Π or Λ: then F_T^* comes from weakening and has the shape $\mathbf{W}{+}F$. Since $merge(F_T^1, \mathbf{W}{+}F) = F_T^1$, we need to check $\neg\mathrm{Dp}(F_T^1 \vee G_T^2) \to \neg\mathrm{Dp}(merge(F_T^1, F_T^2) \vee merge(G_T^1, G_T^2))$ which simplifies to $(\neg\mathrm{Dp}(F_T^1) \to \mathrm{Dp}(merge(F_T^1, F_T^2))) \vee (\neg\mathrm{Dp}(G_T^2) \to \neg\mathrm{Dp}(merge(G_T^1, G_T^2)))$. Since both disjuncts are instances of lemma 3.3.3, also this case is covered.

□

Lemma 3.4.9. Let $\pi \rhd_u^1 \pi'$ be a unary rule shift. Then $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ where $S = extract(\pi')$ and $T = extract(\pi)$.

Proof. We distinguish on the type of the inference σ:

- σ is a unary rule:
  Then the general rewrite pattern looks like:
  $$\dfrac{\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash M, G, \Delta}\,\rho}{\Gamma \vdash M, N, \Delta}\,\sigma \qquad \rightsquigarrow \qquad \dfrac{\dfrac{\Gamma \vdash F, G, \Delta}{\Gamma \vdash F, N, \Delta}\,\sigma}{\Gamma \vdash M, N, \Delta}\,\rho$$
  Depending on the actual rule, the polarity and number of occurrences of the formulas F and G in the parent proof may differ. But in each case, when we look at the (indpendent) paths between F and M / G and N in π and π', they are pairwise homomorphic. Then because the extraction of expansion trees depends only on the structure of M and N, the extracted expansion sequents S and T must be the same and therefore $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ holds.

- σ is a binary rule:
  Then the general rewrite pattern looks like:
  $$\dfrac{\dfrac{\Gamma \vdash F, G_1, \Delta}{\Gamma \vdash M, G_1, \Delta}\,\rho \qquad \Pi, G_2 \vdash \Lambda}{\Gamma, \Pi, M, G_1 \vee G_2 \vdash \Delta, \Lambda}\,\sigma \qquad \rightsquigarrow \qquad \dfrac{\Gamma \vdash F, G_1, \Delta \qquad \dfrac{\Pi, G_2 \vdash \Lambda}{\Gamma, \Pi, F, G_1 \vee G_2 \vdash \Delta, \Lambda}\,\sigma}{\Gamma \vdash M, G_1 \vee G_2, \Delta}\,\rho$$
  Again, the paths F to M, G_1 to $G_1 \vee G_2$ and G_2 to $G_1 \vee G_2$ in π are homomorphic to their counterparts in π'. Then again, the extracted expansion sequents do not differ making $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ valid.

□

Lemma 3.4.10. Let $\pi \rhd_b^1 \pi'$ be a binary rule shift. Then $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ where $S = extract(\pi')$ and $T = extract(\pi)$.

Proof. We distinguish on the type of the inference σ:

- σ is a unary rule:
  Then the general pattern is:

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F, G_1 \vdash \Delta_1, \Lambda \qquad \Pi, \Gamma_2, G_2 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, F, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, F, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,c:*}{\Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,\sigma \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F, G_1 \vdash \Delta_1, \Lambda}{\Pi, \Gamma_1, M, G_1 \vee G_2 \vdash \Delta_1, \Lambda}\,\sigma \qquad \Pi, \Gamma_2, G_2 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, M, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Lambda}\,c:*$$

  Similar to $\rhd_u^1$, the paths from F to M, G_1 to $G_1 \vee G_2$ and G_2 to $G_1 \vee G_2$ in π and π' are homomorphic, leading to identical expansion sequents S and T such that $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ holds.

- σ is a binary rule:
  Then the general pattern rewrites

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F_1, G_1 \vdash \Delta_1, \Lambda \qquad \Pi, \Gamma_2, F_2, G_1 \vdash \Delta_2, \Lambda}{\Pi, \Pi, \Gamma_1, \Gamma_2, F_1 \vee F_2, G_1, G_1 \vdash \Delta_1, \Delta_2, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, F_1 \vee F_2, G_1 \vdash \Delta_1, \Delta_2, \Lambda}\,c:* \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda}\,\sigma$$

  to

$$\dfrac{\dfrac{\dfrac{\Pi, \Gamma_1, F_1, G_1 \vdash \Delta_1, \Lambda \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_3, F_1, G_1 \vee G_2 \vdash \Delta_1, \Delta_3, \Lambda}\,\sigma \qquad \dfrac{\Pi, \Gamma_2, F_2, G_1 \vdash \Delta_2, \Lambda \qquad G_2, \Gamma_3 \vdash \Delta_3}{\Pi, \Gamma_1, \Gamma_3, F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_3, \Lambda}\,\sigma}{\Pi, \Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda, \Lambda}\,\rho}{\Pi, \Gamma_1, \Gamma_2, \Gamma_3, F_1 \vee F_2, G_1 \vee G_2 \vdash \Delta_1, \Delta_2, \Delta_3, \Lambda}\,c:*$$

  In this case, the duplicated parent trees lead to additional merges. Suppose the expansion sequents corresponding to parent proofs are $\Pi_T, \Gamma_{1T}, F_{1T}, G_{1T} \vdash \Delta_{1T}, \Lambda_T$ and $G_{2T}, \Gamma_{3T} \vdash \Delta_{3T}$. Let Q_S and Q_T be expansion trees in the extracted expansion sequents S and T of π and π'. We distinguish on the ancestorship of Q_S/Q_T in the parent proofs:

  - The trees correspond to formulas in Π, Λ:
    Then $Q_S = Q_T = merge(P, Q)$ where P, Q are expansion trees in Π_T or Λ_T. The implication $\mathrm{Dp}(Q_S) \to \mathrm{Dp}(Q_T)$ follows trivially.
  - The trees correspond to formulas in $\Gamma_1, \Gamma_2, \Delta_1, \Delta_2$:
    Then $Q_S = Q_T = Q$ where Q is an expansion tree in $\Gamma_{1T}, \Gamma_{2T}, \Delta_{1T}, \Delta_{2T}$. Again the implication $\mathrm{Dp}(Q_S) \to \mathrm{Dp}(Q_T)$ holds trivially.
  - The trees correspond to formulas in Γ_3, Δ_3:
    Then $Q_S = merge(Q, Q)$ and $Q_T = Q$ where Q is an expansion tree in Γ_{3T}, Δ_{3T}. The implication $\mathrm{Dp}(S) \to \mathrm{Dp}(T)$ holds by the idempotence of merge (lemma 3.3.1).

– The trees corresponds to the formula $F_1 \vee F_2$:
Then $Q_S = Q_T = F_{1T} \vee F_{2T}$ making the implication $\mathrm{Dp}(Q_S) \to \mathrm{Dp}(Q_T)$ trivial again.

– The trees corresponds to the formula $G_1 \vee G_2$:
Then $Q_S = G_{1T} \vee merge(G_{2T}, G_{2T})$ and $Q_T = G_{1T} \vee G_{2T}$ and again the implication $\mathrm{Dp}(Q_S) \to \mathrm{Dp}(Q_T)$ holds by the idempotence of merge.

□

Putting all these results together we can lift the soundness of the deskolemization procedure on LK_{sk} trees to a soundness result on the extracted expansion trees, proving the conjecture 3.4.6 stated earlier.

Theorem 3.4.11. Let T be a skolem expansion tree obtained by applying $extract(\pi)$ to a weakly regular LK_{sk} proof π. Then $\mathrm{Dp}(T)$ is a tautology and T is a skolem expansion proof.

Proof. Suppose π is an LK_{sk} proof where quantifier inferences are correctly placed. Then by theorem 2.6.33 there exists a regular LK proof π' corresponding to π such that by the soundness of LK, the end-sequent of π' is valid. The difference between π and π' only lies in the replacement of skolem terms by unique eigenvariables. The uniqueness allows us to reverse this replacement such that we can define a substitution σ mapping the eigenvariables to the skolem terms in π. But then certainly the deep formula $\mathrm{Dp}(extract(\pi))$ is the instance $\mathrm{Dp}(extract(\pi'))\sigma$ and the validity of the general formula carries over.

Suppose π is an LK_{sk} proof where the quantifier inferences are not correctly placed. Then by lemma 2.6.32 there exists a series of applications of $\rhd_c$, $\rhd_{zip}$, $\rhd_u$ and $\rhd_b$ such that the resulting proof has correctly placed quantifier inferences. Let $\pi_i \rhd \pi_{i+1}$ be such a step in the reduction sequence of proofs $\pi, \pi_1, \ldots \pi_n$. Then by lemmas 3.4.7, 3.4.8, 3.4.9 and 3.4.10, we know that $\mathrm{Dp}(extract(\pi_{i+1})) \to \mathrm{Dp}(extract(\pi_i))$. Now the final proof π_n has correctly placed quantifier inferences and therefore we know that $\mathrm{Dp}(\pi_n)$ is valid. Following the implication chain to the original proof π, this validity is preserved. Therefore $\mathrm{Dp}(extract(\pi))$ is valid and $extract(\pi)$ is an expansion proof. □

The impact of this theorem on the proof analysis is that we can directly take the expansion proof obtained from the LK_{sk} proof obtained without performing the elaborate post-processing steps. The only disadvantage is that rewriting via $\rhd_{zip}$ removes sub-proofs such that a directly extracted expansion proof might not be minimal.

Finally, because the deep formula does not contain any quantifiers in the formula structure[9], it can be double-checked for validity by a SAT-solver with an apropriate encoding. Since Satallax is also saturation based, we expect it to work well without encoding.

In the case of $LK_{sk=}$, we do not have an equivalence to theorem 3.4.11 yet. Still, an SMT solver handling equality modulo uninterpreted functions suffices to double-check the validity of the deep formula. Since Satallax is just SAT but not SMT based, we only expect Leo II to handle these problems well.

[9] They may be present in arguments of an atom but can not be pushed to the formula level.

3.5 Extracting Expansion Proofs from LK Proofs with Propositional Cuts

Miller's extraction algorithm for expansion trees is only defined for cut-free proofs. Our motivation for extracting expansion proofs from LK proofs with propositional cuts is twofold: The primary reason is that the CERES method produces proofs with atomic cuts. If we are only interested in the expansion proof, a direct extraction makes the elimination of atomic cuts – which has an exponential time complexity proportional to the size of the proof[10] – superfluous. A similar situation arises when we leave propositional cuts in place since then they are already present in the projections.

The secondary reason is our long-term goal to use expansion proofs even earlier in the CERES procedure. Replacing the projections by expansion proofs, we can define the simulation of the refutation of the characteristic sequent set as operations on expansion proofs. The crucial step there is that ground expansion proofs are not closed under substitution, which is so far unsolved. What still poses problems is that a substitution $\sigma = \{x \leftarrow t\}$ is not restricted to ancestors of formulas in the end-sequent which contain x. Let us assume that the succedent occurrence of the axiom inference $P(x) \vdash P(x)$ is preserved in the end-sequent, but the ancestor occurrence will be used to infer $\forall y\ P(y) \vdash P(x)$. Then the node expanding the quantifier will also contain x which must be replaced by t when applying σ to the proof. But since a branch of an expansion proof can come from a merge of multiple sub-trees, it is unclear if the substituted term is sufficient in all of these sub-trees. In the context of CERES$^\omega$ the situation is even worse since the refutation simulation also inserts inferences on axioms. For the same reason as before it is unclear how to identify the parts connected by an axiom. It is possible that an annotation linking the formulas connected by an introduction rule suffices to define substitution, but we have not succeeded yet.

Nonetheless, as a first step into this direction, let us prove that an expansion tree's property of being an expansion proof is invariant under Gentzen's reductive cut-elimination rules, as long as occurrences of cut formulas are always restricted or quantifier free. For the proof, we will use expansion sequents (see section 2.3).

Since we will be reasoning on parent proofs which contain a cut-formula, we need to adapt the notion of expansion trees. Although the class of LK_{skc} proofs we are considering does not contain any LK quantifier rules with eigenvariables, a restricted formula may still contain quantifiers which are irrelevant to the validity of the theorem. Since an atom node can not represent such a formula, we introduce such a passive node.

Definition 3.5.1. Let F be a formula, then the *passive node* $\mathbf{P}{+}F$ is an expansion tree with $\mathrm{Sh}(\mathbf{P}{+}F) = \mathrm{Dp}(\mathbf{P}{+}F) = F$. The merge of passive nodes is defined as $merge(\mathbf{P}{+}F, T) = merge(T, \mathbf{P}{+}F) = T$ and the expansion tree extraction is extended: let $S_1 \times S_2$ be a product of two sequents, where the formulas in S_1 are ancestors of end-sequent formulas and those in S_2 are ancestors of cut-formulas. Then $extract_p(S_1 \times S_2) = extract(S_1) \times P$ where each expansion tree in P is $\mathbf{P}{+}(F)$ for each corresponding formula occurrence F in S_2. Also

[10] This results directly from the rank reduction rules in Gentzen's algorithm, in particular from the number of contraction rules applied. See also Theorem 5.2.4 in Methods of Cut-Elimination [61].

$extract_p(F \vdash F) = \mathbf{P}{+}(F) \vdash \mathbf{P}{+}(F)$ for an axiom where one of the occurrences of F is passive.

Lemma 3.5.2. Let π be a balanced LK_{skc} proof where all cut formulas are propositional or passive and let T_π be the expansion sequent extracted from π. Furthermore, let ρ be the LK_{sk} proof obtained by applying one of the reductive cut-elimination rules of definition 2.4.4 to π and let T_ρ be the expansion sequent extracted from ρ. Then $\mathrm{Dp}(T_\rho) \to \mathrm{Dp}(T_\pi)$.

Proof. We proceed by induction on the length of the reduction sequence. The base case $\pi = \rho$ is trivial. For the step case we distinguish among the possible rules which can be applied:

1. The cut has two axiom parents:
$$\dfrac{\dfrac{}{A \vdash A} \quad \dfrac{}{A \vdash A}}{A \vdash A}\, cut \qquad \rightsquigarrow \qquad A \vdash A$$
Suppose $T_\pi = T_1 \vdash T_2$ then $T_\rho = T_1 \vdash T_2$ and for both atom and passive nodes, $\mathrm{Dp}(T_1) = \mathrm{Dp}(T_2) = A$.

2. The cut formula is introduced by weakening:
$$\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta, F} \quad \dfrac{(\pi_2)}{\dfrac{\Gamma_2 \vdash \Delta_2}{F, \Gamma_2 \vdash \Delta_2}\, w:l}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut \qquad \rightsquigarrow \qquad \dfrac{\dfrac{(\pi_2)}{\Gamma_2 \vdash \Delta_2}\, w:l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, w:r$$
Suppose $T = extract(\pi_2) = T_{\Gamma_2} \vdash T_{\Delta_2}$ then also $\mathrm{Dp}(extract(\rho)) = \mathrm{Dp}(T_{\Gamma_2} \vdash T_{\Delta_2})$ because for each antecedent tree $\mathbf{W}{+}(F) = \top$ and for each succedent tree $\mathbf{W}{+}(G) = \bot$.

3. The cut formula is introduced by negation:
$$\dfrac{\dfrac{(\pi_1)}{\dfrac{\Gamma_1, A \vdash \Delta_1}{\Gamma_1 \vdash \Delta_1, \neg A}\, \neg:r} \quad \dfrac{(\pi_2)}{\dfrac{\Gamma_2 \vdash A, \Delta_2}{\Gamma_2, \neg A \vdash \Delta_2}\, \neg:l}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut \qquad \rightsquigarrow \qquad \dfrac{\dfrac{(\pi_1)}{\Gamma_2 \vdash \Delta_2, A} \quad \dfrac{(\pi_2)}{\Gamma_1, A \vdash \Delta_1}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$
Then the ancestor relation of end-sequent formulas is untouched and $\mathrm{Dp}(extract(\pi)) = \mathrm{Dp}(extract(\rho))$.

4. The cut formula is introduced by conjunction:
$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash \Delta_2, B}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B}\, \wedge:r \quad \dfrac{(\pi_3)}{\dfrac{\Gamma_3, A \vdash \Delta_3}{\Gamma_3, A \wedge B \vdash \Delta_3}\, \wedge:l}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Gamma_3}\, cut \qquad \rightsquigarrow \qquad \dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_3)}{\Gamma_3, A \vdash \Delta_3}}{\dfrac{\Gamma_1, \Gamma_3 \vdash \Delta_1, \Delta_3}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3}\, w:*}\, cut$$
Suppose $T_\pi = T_{\Gamma_1}, T_{\Gamma_2}, T_{\Gamma_3} \vdash T_{\Delta_1}, T_{\Delta_2}, T_{\Delta_3}$ then $T_\rho = T_{\Gamma_1}, W_1, T_{\Gamma_3} \vdash T_{\Delta_1}, W_2, T_{\Delta_3}$ where W_1 and W_2 are the weakening nodes corresponding to $\mathrm{Sh}(\Gamma_2)$ and $\mathrm{Sh}(\Delta_2)$. Then for the same reasons as in the weakening case, $\mathrm{Dp}(T_\rho) = \mathrm{Dp}(T_{\Gamma_1}, T_{\Gamma_3} \vdash T_{\Delta_1}, T_{\Delta_3})$. But the latter sequent is a sub-sequent of $\mathrm{Dp}(\pi)$ and each subsequent implies its weakend form. Therefore $\mathrm{Dp}(\rho) \to \mathrm{Dp}(\pi)$.

5. The cut formula is introduced by disjunction/implication:
The reasoning is analogous to the conjunction case.

6. The cut formula is introduced by quantification:
   This case can not happen because all cuts are either propositional or passive.

7. The cut is permuted over a unary rule (which is not contraction):

$$\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{\dfrac{\Gamma_2 \vdash A, \Delta_2}{\Gamma_2' \vdash A, \Delta_2'}\,\rho}}{\Gamma_1, \Gamma_2' \vdash \Delta_1, \Delta_2'}\,cut \quad \rightsquigarrow \quad \dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash A, \Delta_2}}{\dfrac{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}{\Gamma_1, \Gamma_2' \vdash \Delta_1, \Delta_2'}\,\rho}\,cut$$

   The inferences are independent and the ancestor relationship unchanged, leading to $T_\pi = T_\rho$ and $\mathrm{Dp}(T_\rho) = \mathrm{Dp}(T_\pi)$.

8. The cut is permuted over a binary rule:

$$\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{\dfrac{(\pi_2)}{A, \Gamma_2 \vdash \Delta_2} \quad \dfrac{(\pi_3)}{\Gamma_3 \vdash \Delta_3}}{A, \Gamma_2', \Gamma_3' \vdash \Delta_2', \Delta_3'}\,\rho}{\Gamma_1, \Gamma_2', \Gamma_3' \vdash \Delta_1, \Delta_2', \Delta_3'}\,cut \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, A} \quad \dfrac{(\pi_2)}{A, \Gamma_2 \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\,cut \quad \dfrac{(\pi_3)}{\Gamma_3 \vdash \Delta_3}}{\Gamma_1, \Gamma_2', \Gamma_3' \vdash \Delta_1, \Delta_2', \Delta_3'}\,\rho$$

   Again, the inferences are independent and the ancestor relationship unchanged, leading to $T_\pi = T_\rho$ and $\mathrm{Dp}(T_\rho) = \mathrm{Dp}(T_\pi)$.

9. The cut is permuted over a contraction rule:
   We show the case for contractions on both cut-formulas where the last inference applied before was a conjunction; the other cases are analogous.

   The pattern

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F^b, B, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash F^{a+b}, A \wedge B, \Delta_1}\,\wedge : r}{\Gamma_1, \Gamma_2 \vdash F, A \wedge B, \Delta_1, \Delta_2}\,(a+b-1) \times c : r \quad \dfrac{\dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A \wedge B, F^n \vdash \Delta_3}\,\wedge : l}{\Gamma_3, A \wedge B, F \vdash \Delta_3}\,(n-1) \times c : l}{\Gamma_1, \Gamma_2 \vdash F, \Delta_1, \Delta_2}\,c : r \quad \dfrac{\Gamma_3, A \wedge B, F \vdash \Delta_3}{\Gamma_3, F \vdash \Delta_3}\,c : l}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3}\,cut$$

   is rewritten to the two proofs π_4 and π_5

$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1}}{\Gamma_1 \vdash F, A, \Delta_1}\,(a-1) \times c : r \quad \dfrac{\dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A \wedge B, F^n \vdash \Delta_3}\,\wedge : l}{\Gamma_3, A \wedge B, F \vdash \Delta_3}\,(n-1) \times c : l}{\Gamma_3, F \vdash \Delta_3}\,c : l}{\Gamma_1, \Gamma_3 \vdash A, \Delta_1, \Delta_3}\,cut$$

(π_4)

   and

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash F^a, A, \Delta_1} \quad \dfrac{(\pi_2)}{\Gamma_2 \vdash F^b, B, \Delta_2}}{\Gamma_1, \Gamma_2 \vdash F^{a+b}, A \wedge B, \Delta_1}\wedge : r}{\Gamma_1, \Gamma_2 \vdash F, A \wedge B, \Delta_1, \Delta_2}(a+b-1) \times c : r \quad \dfrac{\dfrac{(\pi_3)}{\Gamma_3, A, F^n \vdash \Delta_3}}{\Gamma_3, A, F \vdash \Delta_3}(n-1) \times c : l}{\Gamma_1, \Gamma_2, \Gamma_3, A \vdash \Delta_1, \Delta_2, \Delta_3}\, cut$$

which are combined again by cut:

$$\dfrac{\dfrac{\dfrac{(\pi_4)}{\Gamma_1, \Gamma_3 \vdash A, \Delta_1, \Delta_3} \quad \dfrac{(\pi_5)}{\Gamma_1, \Gamma_2, \Gamma_3, A \vdash \Delta_1, \Delta_2, \Delta_3}}{\Gamma_1, \Gamma_3, \Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_3, \Delta_1, \Delta_2, \Delta_3}\, cut}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash \Delta_1, \Delta_2, \Delta_3}\, c : l + c : r$$

The case is similar to the uncontracted conjunction case, just with merges added. Suppose T_F is an expansion tree resulting from $extract(F)$ where F appears in $\Gamma_1, \Gamma_3, \Delta_1, \Delta_3$ within π, then the corresponding tree in ρ is $merge(F, F)$. By lemma 3.3.1, we know that $\mathrm{Dp}(merge(F, F)) \to \mathrm{Dp}(F)$ and $\neg\mathrm{Dp}(merge(F, F)) \to \neg\mathrm{Dp}(F)$, depending on the polarity of F. For occurrences in T_G of a G in Γ_2, Δ_2 the corresponding tree in ρ is identical to T_G. Therefore, $\mathrm{Dp}(T_\rho) \to \mathrm{Dp}(T_\pi)$.

□

Together with theorem 3.4.11, we can now extract an expansion proof directly from the PCNF, skipping the elimination of passive cuts as well as the whole de-skolemization process.

3.6 Definition Elimination

An issue which impedes the readability of sequent calculus is that the context is repeated in each inference. Sometimes even a single large formula is hard to understand. One approach taken in mathematics is the use of definitions to abstract over details which are not required throughout the whole proof. Without definitions, even simple mathematical texts become unreadable. For instance, already the first chapter of "Proofs from the Book" [1] presenting six different proofs of the infinity of primes silently presupposes (amongst others) the definitions of a number being prime, a number being the divisor of another, the group axioms[11] and the power series of the logarithm function. Sometimes, the act of folding or unfolding a definition is relevant to the proof though. Examples for this occur in the same chapter: In proof 4, the authors identify a sum as a geometric series and insert its explicit form to simplify an expression. Further on in proof 5, we are shown properties of open and closed sets by pure definition unfolding. To achieve a similar kind of abstraction, we integrate definition rules to LK_{skc} in the following manner:

[11] Mentioned in the proof for Lagrange's theorem.

Definition 3.6.1 (Definition rewrite rule)**.** A *definition* is a rewrite rule $c_\alpha \to t_\alpha$, where c is a constant different from the logical operators as well as skolem constants and the definiens t is a HOL term with $FV(t) = \emptyset$.

Definition 3.6.2 (Definition Rule)**.** We define the inference rule *def* as:

$$\frac{\Gamma \vdash \Delta, F}{\Gamma \vdash \Delta, G}\, def : R \qquad\qquad \frac{\Gamma \vdash \Delta, G}{\Gamma \vdash \Delta, F}\, undef : R$$

If F rewrites to G within one step using the definition R.

From our mathematical intuition we would expect that definitions are removable. Usually this is guaranteed by ordering the definitions together with the requirement that the definiens only contains simpler definitions. We will abstract from this notion and just require that $\to_{dt(\pi)}$ is convergent, i.e. it is confluent and terminating. The former will assure that definitions unfold to a unique normal form while the latter prevents infinite definition unfolding sequences.

The restriction of the left-hand side in a definition rewrite rule to a constant is less severe than it seems because the definition $taut(x) = x \vee \neg x$ can be formulated as the rule $taut \to \lambda x.x \vee \neg x$. Unfortunately, it is sufficiently strong that, together with the termination requirement, it excludes any recursive definitions, since a rule $c \to f[c]$ cannot terminate. Still we believe that this approach can be extended to recursive definitions by lifting the restriction the constants as left-hand side of a rule and some syntactic restrictions on the definiens instead of convergence.

It is also worth mentioning that the elimination of a single definition inference from a proof has some global effects. For example, if a defined formula is contracted further on in the proof, the contraction can only be applied if both auxiliary formulas are unfolded. But the second formula could have been directly introduced by an axiom or a weakening rule. In a similar manner, a weak quantifier rule can replace occurrences of the term t in different sub-formulas. Again, one sub-formula might have introduced t by a definition rule but the other occurrence of t was already present during the axiom or weakening rule which introduced its respective sub-formula. Examples of both cases can be seen in figure 3.1.

With definitions

$$\dfrac{\dfrac{\dfrac{F \vdash F}{F \vdash G}\, def \qquad \dfrac{A \vdash A}{A \vdash A, G}\, w : r}{F \vee A \vdash A, G, G}\, \vee : l}{F \vee A \vdash A, G}\, c : r$$

$$\dfrac{\dfrac{\dfrac{F(s(0)) \vdash F(s(0))}{F(s(0)) \vdash F(t)}\, def \qquad G(t) \vdash G(t)}{F(s(0)), G(t) \vdash F(t) \wedge G(t)}\, \wedge : r}{F(s(0)), G(t) \vdash (\exists x F(x), G(x))}\, \exists : r$$

Without definitions

$$\dfrac{\dfrac{F \vdash F \qquad \dfrac{A \vdash A}{A \vdash A, F}\, w : r}{F \vee A \vdash A, F, F}\, \vee : l}{F \vee A \vdash A, F}\, c : r$$

$$\dfrac{\dfrac{F(s(0)) \vdash F(s(0)) \qquad G(s(0)) \vdash G(s(0))}{F(s(0)), G(s(0)) \vdash F(s(0)) \wedge G(s(0))}\, \wedge : r}{F(s(0)), G(s(0)) \vdash (\exists x F(x), G(x))}\, \exists : r$$

Figure 3.1: Examples of global effects of definition elimination

To reflect the global nature of definition rules, we now introduce the notion of a definition table. A welcome side-effect is that we do not need to annotate the particular definition used in

a rule, as long the definition table is known.

Definition 3.6.3 (Definition Table)**.** Furthermore, a *definition table* $dt(\pi)$ of a proof π is the rewrite system consisting of all definitions occurring in definition rules within π. We denote the reflexive-transitive closure of a definition table $dt(\pi)$ by $\to_{dt(\pi)}$.

Theorem 3.6.4 (Admissibility of definition rules)**.** Let π be an LK_{skc} proof with definition rules and end-sequent S. Moreover, let us assume that $\to_{dt(\pi)}$ is convergent, i.e. it is confluent and terminating. Then there exists an LK_{skc} proof of the end-sequent S' without definition rules, where S' is obtained by applying $\to_{dt(\pi)}$ to each label and formula in S.

Before giving the proof, we will restate two well-known results in rewriting. If a rewrite system $\to_1 \cup \to_2$ is confluent, it does not follow that $\to_1$ and $\to_2$ are confluent[12]. Moreover, if $\to_1$ and $\to_2$ are terminating, it does not follow that $\to_1 \cup \to_2$ is terminating[13]. This means that we cannot expect to inductively combine the definition tables of a binary rule's parents. What works out though, is that we use a global definition table to remove all definitions together.

Proof. We obtain our result with a transformation $ED(\rho, \to)$, where ρ is an LK_{skc} proof with definitions and $\to$ is a strongly normalizing rewrite system. It recursively applies $\to$ to every label, formula and substitution term of a rule in π. Then $ED(\pi, \to_{dt(\pi)})$ has the desired properties.

We will show this by induction on the structure of ρ. In all cases, the context of an inference stays unchanged, therefore the claim that the context was correctly rewritten follows directly from the induction hypothesis. For the active formulas, we make a case distinction on the last inference $(*)$, using $F\!\downarrow$ as a shorthand notation for the normal form of F with regard to $\to$ as well as using $\ell\!\downarrow$ for $\{x\!\downarrow |\ x \in \ell\}$:

- $(*)$ is an axiom: we directly obtain the rewritten proof since we apply $\to$ to all labels and all formulas

- $(*)$ is a definition rule with auxiliary formula F and primary formula G: w.l.o.g. we assume $F \to G$. Then $F\!\downarrow = G\!\downarrow$, i.e. the parent proof has already the desired form, so we can safely skip the definition inference.

- $(*)$ is a strong quantifier rule inferring $\langle QF\rangle^{\ell}$ from $\langle FX\rangle^{\ell}$ ($Q \in \{\forall, \exists\}$): by IH our auxiliary formula is $\langle (FX)\!\downarrow\rangle^{\ell\downarrow}$. Since each left-hand side of a rewrite rule is a constant there can be no overlap between F and X i.e. $(FX)\!\downarrow = F\!\downarrow X\!\downarrow$. Since X is a variable we this is also equivalent to $\langle F\!\downarrow X\rangle^{\ell\downarrow}$. The eigenvariable condition still holds, since the free variables did not change. Applying (*), we infer $\langle Q(F\!\downarrow)\rangle^{\ell\downarrow}$. Furthermore, because quantifiers are not subject to definitions, we know that $\langle Q(F\!\downarrow)\rangle^{\ell\downarrow} = \langle (QF)\!\downarrow\rangle^{\ell\downarrow}$, which is what is required.

[12] A counter-example: $\{a \to b; a \to c\}$ is not confluent, but $\{a \to b; a \to c\} \cup \{b \to c\}$ is.
[13] A counter-example: $\{a \to b\}$ and $\{b \to a\}$ are terminating, but $\{a \to b\} \cup \{b \to a\}$ is not.

- $(*)$ is a weak quantifier rule inferring $\langle QF\rangle^{\ell}$ from $\langle FT\rangle^{\ell}$ ($Q \in \{\forall, \exists\}$): by IH we know that our auxiliary formula is $\langle (FT)\downarrow\rangle^{\ell\downarrow}$. Again there is no possible overlap between F and T, leading us to $\langle F\downarrow T\downarrow\rangle^{\ell\downarrow}$. We now apply $(*)$ to obtain $\langle Q(F\downarrow)\rangle^{\ell\downarrow}$ which is equivalent to $\langle (QF)\downarrow\rangle^{\ell\downarrow}$, since quantifiers are not changed by the $\rightarrow$ relation.

- $(*)$ is a strong skolem quantifier rule inferring $\langle QF\rangle^{\ell}$ from $\langle F(f(S_1, \ldots, S_n))\rangle^{\ell}$ where $\ell = \{S_1, \ldots, S_n\}$ ($Q \in \{\forall, \exists\}$): Since f is a skolem symbol, it will never be directly rewritten. By the absence of overlaps we obtain the equivalence $F(f(S_1, \ldots, S_n))\downarrow = F\downarrow f(S_1\downarrow, \ldots, S_n\downarrow)$. Since definitions preserve all free variables, the skolem term will be contained at least once after beta normalization.

  Therefore by IH we obtain a formula $\langle F\downarrow(f(S_1\downarrow, \ldots, S_n\downarrow))\rangle^{\ell\downarrow}$. Looking at the definition of a strong skolem quantifier rule, we see that $\{S_1, \ldots, S_n\}\downarrow = \ell\downarrow$ allowing us to apply $(*)$ to infer $\langle QF\downarrow\rangle^{\ell\downarrow}$. Shifting the rewriting over the quantifier again, we obtain $\langle (QF)\downarrow\rangle^{\ell\downarrow}$, which is the formula required.

- $(*)$ is a weak skolem quantifier rule inferring $\langle QF\rangle^{\ell}$ from $\langle FT\rangle^{\ell,T}$ ($Q \in \{\forall, \exists\}$): we can follow the argumentation for weak quantifiers, but have to make sure the labels behave properly. By IH the auxiliary formula is $\langle (FT)\downarrow\rangle^{\ell\downarrow,T\downarrow}$, which is equivalent to $\langle F\downarrow T\downarrow\rangle^{\ell\downarrow,T\downarrow}$. Since the labels agree, we can apply $(*)$, obtaining $\langle QF\downarrow\rangle^{\ell\downarrow}$ and subsequently $\langle (QF)\downarrow\rangle^{\ell\downarrow}$.

- $(*)$ is a unary logical rule inferring $\neg F$ from F: by IH we have $F\downarrow$ as auxiliary formula and apply $(*)$ to infer $\neg F\downarrow$. Since logical operators are unchanged by $\rightarrow$, this is equivalent to $(\neg F)\downarrow$.

- $(*)$ is a binary logical rule inferring $F \circ G$ from F and G: by IH we obtain $F\downarrow$ and $G\downarrow$ as auxiliary formulas. We apply $(*)$ to infer $F\downarrow \circ G\downarrow$, which is equivalent to $(F \circ G)\downarrow$ again.

- $(*)$ is a weakening rule introducing F: since there are no restrictions on the weakening rule, we just introduce $F\downarrow$.

- $(*)$ is a contraction rule with auxiliary formula F: by IH we know that there are two occurrences of $F\downarrow$ in the parent proof, which we can contract again.

- $(*)$ is a cut rule on the formula F: by IH we know that there are suitable occurrences of $F\downarrow$ in the parent proofs, allowing us to apply the cut rule again.

□

We will conclude this section with an overview of some of the challenges when we extend to recursive definitions. So far we strongly relied on the impossibilities of overlaps in an application $s(t)$. It allowed us to disregard β-normalization but it also prevented quantifier inferences on an overlap. Let us consider the definition $P(a) \rightarrow A$ together with the following proof:

$$\dfrac{\dfrac{\dfrac{A \vdash A}{A \vdash P(a)}\,def \qquad P(b) \vdash P(b)}{A, P(a) \to P(b) \vdash P(b)} \to\!: l}{A, \exists X(X(a) \to X(b)) \vdash P(b)}\,\exists : l$$

Every ground instance of $P(x)$ has a unique normal-form, but the rewritten proof

$$\dfrac{A \vdash A \qquad P(b) \vdash P(b)}{A, A \to P(b) \vdash P(b)} \to\!: l$$

lacks a suitable instance for X to apply the $\exists : l$ inference. Given an if-then-else operator, a possible substitution would be $X \leftarrow \lambda x(if\ x = a\ then\ A\ else\ P(x)))$. But as soon as we quantify over a predicate or function variable, solving the equation $x = f$ may be undecidable and therefore cannot be resolved automatically. An alternative might be to use the notions of higher-order rewriting [57, 68, 109] which takes substitution and β-reduction into account.

3.7 Encoding Computations in Arithmetic

3.7.1 Conditionals (If-then-else)

A basic ingredient of computation is branching, in programming usually implemented as the if-then-else statement. Roughly speaking, we would expect that we can prove the equations $if\ (exp_o)\ then\ t_\alpha\ else\ f_\alpha = t_\alpha$ in case the conditional exp is true and $if\ (exp_o)\ then\ t_\alpha\ else\ f_\alpha = f_\alpha$ otherwise. The central question is now, how we evaluate exp.

A possible approach is to use Church numerals and definition by cases defined in section 2.1.1. Alternatively, the projections $\lambda x_\iota \lambda y_\iota\ x$ and $\lambda x_\iota \lambda y_\iota\ y$ may represent true and false, which can be directly used to implement the conditional as $\lambda t_\iota \lambda f_\iota \lambda e_{\iota>\iota>\iota}\ e\ t\ f$.

Practical as this seems, the strong quantifier rules of sequent calculus block many computations. Suppose we would like to prove an induction step as follows:

$$\dfrac{\dfrac{\dfrac{(\pi)}{P(v,w) \vdash \exists y P(v+1,y)}\,\exists : l}{\exists y P(v,y) \vdash \exists y P(v+1,y)} \to\!: r}{\vdash \exists y P(v,y) \to \exists y P(v+1)}\,\forall : r$$

$$\vdash \forall x(\exists y P(x,y) \to \exists y P(x+1,z))$$

The eigenvariables v and w will stay untouched in π which severely restricts the expressivity results mentioned in section 2.1.1. We know that $v+1$ is equivalent to $\lambda x \lambda u.v(x, x(u))$ but since v is the head of the application, the expression can only change in the arguments x and $x(u)$. The same problem applies to the condition e of an if-then-else expression. If no substitution unfolds e to one of the two projections mentioned above, the evaluation can not be made on the computational level.

What is possible though is to reason on the logical level and specify computations as equational theories. Then we can prove the theorem $\forall exp \forall t \forall f((if\ (exp)\ then\ t_\alpha\ else\ f_\alpha = t_\alpha) \vee$

$(if\ (exp)\ then\ t_\alpha\ else\ f_\alpha = t_\alpha))$ by making a case distinction that exp is either true or false. Since we have an axiomatization on the logical level, it is practical to formulate all axioms of Second Order Arithmetic there and also encode if-then-else arithmetically. Using bounded subtraction $\dot{-}$, the expression $(1 \dot{-} (1 \dot{-} e)) * t + (1 \dot{-} e) * f$ fulfills the equations above.

However, having both an equational and a computational definition of an arithmetic function leads to duplication. In the worst case, for each pair of functions f, g where f stands for the arithmetical and g stands for the computational definition, we need to prove the equivalence $\forall x(f(x) = g(x))$. In case of the conditional we can hide the actual representation of the behind a conversion function $\llcorner\lrcorner_{o>\iota}$ for the Boolean type and $\ulcorner\urcorner_{\iota>\iota}$ for the individual type. A suitable axiomatization then ensures that the range of both conversions is the set $\{0, 1\}$, simplifying the conditional to $e * t + (1 \dot{-} e) * f$. The equations $X \to \llcorner\lrcorner X) = 1$, $\neg X \to \llcorner\lrcorner X) = 0$ and $\ulcorner\urcorner x) = 1(\dot{-}(1 \dot{-} x)$ provide this, leading to a unified representation of if-then-else. If hiding the arithmetical content behind a definition $ite_{\iota>\alpha>\alpha>\alpha}$, we can also prove the two lemmas $P \vdash ite(\llcorner P\lrcorner, t, f) = t$ and $\neg P \vdash ite(\llcorner P\lrcorner, t, f) = f$. Outside the lemmas, the actual definition of ite does not matter anymore.

This formulation of the conversion functions is a slight diversion from the original goal: we actually embed the result $\llcorner x < y\lrcorner$ of a function instead of the operator $x \llcorner<\lrcorner y$ itself. It is obvious that a general proof of $\llcorner x < y\lrcorner = x \llcorner<\lrcorner y$ requires the axiom of extensionality. Still, for the given task of evaluating the condition of an if-then-else expression on the logical level, the version presented here suffices.

3.7.2 Recursion

Using second order arithmetic as axiom system has the advantage that we know which terms we will encounter: every individual x is either zero or the successor of another individual. Therefore if we prove the existence of an individual x_ι, we can find it by enumerating all numerals (e.g. by term depth) and will eventually find it, although we do not know when. Therefore when we are talking about a mathematical interpretation, we can certainly use the μ recursion operator to find the smallest number which fulfills the predicate. Using μ instead of primitive recursion circumvents the problem of the unknown bound but we don't need it to express functions which grow non-elementary.

3.8 Reducing the characteristic sequent set during construction

Subsumption has a substantial computational complexity [45]. Moreover, the algorithms in GAPT are not highly optimized, so reduction under higher-order subsumption of a sequent set can already take about 10 minutes for sizes of about 1700 clauses. This means that we can not expect subsumption to terminate on set sizes of 100000 or more in reasonable time. In the following, we will use $subsumed(S)$ to denote a maximal subset of S where for each clause $C \in S$, there is exists no clause $D \in S$ where D is subsumed by C. Since the subsumption relation is a non-strict partial order in which two clauses C and D only subsume each other, if they are equal modulo renaming of variables, the set defined by $subsumed$ is also unique up to variable names.

A possible approach is to fold the subsumption into clause set generation. Using the set product $\times$ and union $\cup$ operation to represent characteristic clause sets, we see two possible optimization points in the product and the addition operation. Unfortunately, the subsumption operation is not homomorphic modulo the sequent product.

Lemma 3.8.1. There exists sequent sets C_1 and C_2 for which the following proposition does not hold:

$$subsumed(C_1 \times C_2) \subseteq subsumed(C_1) \times subsumed(C_2))$$

Proof. Take $C_1 = \{P(x) \vdash;\}$ and $C_2 = \{\vdash Q(x);\}$, then $C_1 \times C_2 = \{P(x) \vdash Q(x)\}$. If we now define the substitutions $\sigma = \{x \mapsto a\}$ and $\tau = \{x \mapsto b\}$, it is clear that S_1 subsumes $S_1\sigma$ and S_2 subsumes $S_2\tau$. But $S_1 \times S_2$ does not subsume $S_1\sigma \times S_2\tau$, since x needs to be mapped to a and b at the same time. □

Even though subsumption commutes with set union, the product of a subsumed set can be less general than the product of the original sets.

Lemma 3.8.2. There exist sequents sets C_1 and C_2 for which the following property does not hold:

$$subsumed(C_1) \times C_2 \text{ does not subsume } subsumed(C_1 \times C_2)$$

Proof. Take $C_1 = \{P(x) \vdash; P(y) \vdash\}$ and $C_2 = \{\vdash Q(x)\}$. Then $\{P(x)\}$ subsumes C_1, but does not subsume $P(y) \vdash Q(x)$, which is contained in $C_1 \times C_2$. □

The reason for the failure is that for the merge $\times$ of sequent sets, the uniqueness of $subsumed(S)$ up to variable renaming is insufficient because it may accidentally unify two variables. A refutation of $subsumed(C_1) \times C_2$ can be extended to one of $subsumed(C_1 \times C_2)$ by adding the renaming to the unifiers. Since this refutation is only found in proofs where all contractions are redundant, the result is not helpful in practice.

3.9 A simple First Order Embedding

We perform lambda lifting [56] to replace lambda expressions by functions. Essentially, we obtain a formula in many-sorted first order logic. A prover supporting many-sorted first-order logic could already work with this input, but without sorts an embedding is necessary to stay sound. As a heuristic, the current implementation uses such an unsound embedding which completely drops the types. If the types can be restored, the first-order refutation is also a R_{al} refutation: factoring can be delayed before resolution is applied (as is done in Robinson's original formulation [81] where it is integrated in the resolution rule).

In GAPT, the function `replaceAbstractions` is responsible for lambda lifting. Used on an expression, it just returns the abstracted term:

```
gapt> val exp = hof"p(^(x:i) (x+x))"
exp: at.logic.gapt.expr.HOLFormula = p(λx x + x): o

gapt> replaceAbstractions(exp)
res15: at.logic.gapt.expr.HOLFormula = p('q_{1}':i>i): o
```

Applied to a list of sequents, we also obtain the information to reconstruct the original formula:

```
gapt> val (map, List(HOLSequent(_,lifted :: _))) = replaceAbstractions(List(HOLSequent() :+ exp ))
map: at.logic.gapt.expr.fol.replaceAbstractions.ConstantsMap = Map(λx x + x -> q_{1})
lifted: at.logic.gapt.expr.HOLFormula = p('q_{1}':i>i): o

gapt> undoReplaceAbstractions(lifted, map)
res25: at.logic.gapt.expr.HOLFormula = p(λx x + x): o
```

3.10 A CERES based LK conversion of Resolution Proofs

This part is based on one of our publications [50]. Here we approach the method in the terms of CERES. Its origin is the first-order proof import function of GAPT which takes an arbitrary formula and returns a resolution refutation (e.g. `Prover9.getRobinsonProof`) or an LK proof (e.g. `Prover9.getLKProof`). A direct interpretation of resolution as cut, factoring as contraction and with instances of clauses as axioms leads to a proof of the empty sequent. Already from the user's point of view this is unsatisfactory since she would expect the input formula in the end-sequent. Moreover, from Gentzen's consistency proof via sequent calculus it is obvious that a proof of the empty sequent is only possible if we admit it as an initial sequent. Consequently, an application of reductive cut-elimination will prune every such formulated import to a single introduction of the empty sequent. For the same reason, the expansion sequent of this proof would be empty, making further exploration impossible.

Using the ideas of CERES, we can construct projections from the input formula to each clause. Then we can continue using the original CERES method and simulate the refutation with the projections. In practice there is one more complication: the skolem symbols are introduced by the theorem prover and need to be inferred from the proof output. Therefore it is less problematic to skolemize the input formula before projecting and restoring the strong quantifier inferences afterwards.

We will first define the clause normal form (CNF) decomposition for positive and negative contexts.

Definition 3.10.1 (Clause normal forms)**.** Let A, A_1, A_2 denote formulas without weak quantifiers and B, B_1, B_2 denote formulas without strong quantifiers and let P denote atoms. Define the mappings $\mathtt{CNF}^+(A)$ and $\mathtt{CNF}^-(B)$ by the following mutual induction:

$$\begin{array}{ll}
\mathtt{CNF}^+(P) = \{\vdash P\} & \mathtt{CNF}^+(A_1 \wedge A_2) = \mathtt{CNF}^+(A_1) \cup \mathtt{CNF}^+(A_2) \\
\mathtt{CNF}^-(P) = \{P \vdash\} & \mathtt{CNF}^+(A_1 \vee A_2) = \mathtt{CNF}^+(A_1) \times \mathtt{CNF}^+(A_2) \\
\mathtt{CNF}^+(\neg B) = \mathtt{CNF}^-(B) & \mathtt{CNF}^-(B_1 \wedge B_2) = \mathtt{CNF}^-(B_1) \times \mathtt{CNF}^-(B_2) \\
\mathtt{CNF}^-(\neg A) = \mathtt{CNF}^+(A) & \mathtt{CNF}^-(B_1 \vee B_2) = \mathtt{CNF}^-(B_1) \cup \mathtt{CNF}^-(B_2) \\
\mathtt{CNF}^+(\forall x.A) = \mathtt{CNF}^+(A) & \mathtt{CNF}^-(\exists x.B) = \mathtt{CNF}^-(B)
\end{array}$$

The case of $\rightarrow$ is defined by combining the cases of $\vee$ and $\neg$.

Now we can state an algorithm proving $F \times \mathcal{C}$ which is the projection of a formula F to one of the clauses $\mathcal{C} \in \mathtt{CNF}^+(F)$:

Definition 3.10.2 (CNF Projection)**.** We define the functions $\mathtt{PCNF}^+(F, V)$ which creates a set of sequents $\{\vdash F \times C | C \in \mathtt{CNF}^+(F)\}$ and $\mathtt{PCNF}^-(F, V)$ which creates a set of sequents

$\{F \vdash \times C | C \in \mathtt{CNF}^-(F)\}$. In both cases, V is a list of instantiation terms (i.e. an LK_{sk} label).

$$
\begin{array}{lcl}
\mathtt{PCNF}^+(P,V) = \mathtt{PCNF}^-(P,V) = \{P \vdash P\} & & \\
\mathtt{PCNF}^+(\neg B, V) & = & \{\text{apply } \neg : r \text{ to } B \vdash \times C | B \vdash \times C \in \mathtt{PCNF}^-(B,V)\} \\
\mathtt{PCNF}^+(A_1 \wedge A_2, V) & = & \{\text{apply } \wedge : l \text{ with } A_2 \text{ to } A_1 \vdash \times C | A_1 \vdash \times C \in \mathtt{PCNF}^+(A_1,V)\} \cup \\
 & & \{\text{apply } \wedge : l \text{ with } A_1 \text{ to } A_2 \vdash \times C | A_2 \vdash \times C \in \mathtt{PCNF}^+(A_2,V)\} \\
\mathtt{PCNF}^+(A_1 \vee A_2, V) & = & \{\text{apply } \vee : l \text{ to } A_1 \vdash \times C_1 \text{ and } A_2 \vdash \times C_2 \\
 & & | A_1 \vdash \times C_1 \in \mathtt{PCNF}^+(A_1,V), A_2 \vdash \times C_2 \in \mathtt{PCNF}^+(A_2,V)\} \\
\mathtt{PCNF}^+(\forall x.A, V) & = & \{\text{apply } \forall : l \text{ to } A\sigma \vdash \times C \\
 & & | A\sigma \vdash \times C \in \mathtt{PCNF}^+(A\sigma, z :: V), \sigma = \{x \leftarrow z\}, z \notin FV(V)\} \\
 & & \\
\mathtt{PCNF}^-(\neg A, V) & = & \{\text{apply } \neg : l \text{ to } \vdash A \times C | \vdash A \times C \in \mathtt{PCNF}^+(A,V)\} \\
\mathtt{PCNF}^-(B_1 \wedge B_2, V) & = & \{\text{apply } \wedge : r \text{ to } \vdash B_1 \times C_1 \text{ and } \vdash B_2 \times C_2 \\
 & & | \vdash B_1 \times C_1 \in \mathtt{PCNF}^-(B_1), \vdash B_2 \times C_2 \in \mathtt{PCNF}^-(B_2,V)\} \\
\mathtt{PCNF}^-(B_1 \vee B_2, V) & = & \{\text{apply } \vee : t \text{ with } B_2 \text{ to } \vdash B_1 \times C | \vdash B_1 \times C \in \mathtt{PCNF}^-(B_1,V)\} \cup \\
 & & \{\text{apply } \vee : r \text{ with } B_1 \text{ to } \vdash B_2 \times C | \vdash B_2 \times C \in \mathtt{PCNF}^-(B_2,V)\} \\
\mathtt{PCNF}^-(\exists x.A, V) & = & \{\text{apply } \exists : r \text{ to } \vdash B\sigma \times C \\
 & & | \vdash B\sigma \times C \in \mathtt{PCNF}^-(B\sigma, z :: V), \sigma = \{x \leftarrow z\}, z \notin FV(V)\}
\end{array}
$$

Then we define $\mathtt{PCNF}(F) = \mathtt{PCNF}^+(F, nil)$.

This procedure can easily be extended to a sequent $\Gamma \vdash \Delta$ by computing $\mathtt{PCNF}(\bigwedge \Gamma \rightarrow \bigvee \Delta)$ and skipping the last $\wedge : l$, $\vee : r$ inferences which lead to the final $\rightarrow : r$. We can apply this procedure to create an LK proof of $S : (\forall x_0(P(x_0) \rightarrow P(s(x_0)))), P(0) \vdash P(s(s(0)))$ via resolution. We compute $\mathtt{CNF}^+(S) = \{ \vdash P(0); P(x) \vdash P(s(x)); P(s(s(0))) \vdash \}$ and $\mathtt{PCNF}^+(S)$.

Another extension uses LK_{skc} to introduce strong quantifiers with skolem terms built from the label V passed to $\mathtt{PCNF}^+/\mathtt{PCNF}^-$:

Definition 3.10.3.
We replace the quantifier rules for $\mathtt{PCNF}^+$ and $\mathtt{PCNF}^+$ with the following:

$$
\begin{array}{lcl}
\mathtt{PCNF}^+(\langle \forall A \rangle^V, V) & = & \{\text{apply } \forall_{sk} : l \text{ to } \langle Az \rangle^{z,V} \vdash \times C \\
 & & | \langle Az \rangle^{z,V} \vdash \times C \in \mathtt{PCNF}^+(A\sigma, z :: V), z \notin FV(V)\} \\
\mathtt{PCNF}^-(\langle \exists B \rangle^V, V) & = & \{\text{apply } \exists : r \text{ to } \vdash \langle Bz \rangle^{z,V} \times C \\
 & & | \vdash \langle Bz \rangle^{z,V} \times C \in \mathtt{PCNF}^-(B\sigma, z :: V), z \notin FV(V)\} \\
\mathtt{PCNF}^-(\langle \forall A \rangle^V, V) & = & \{\text{apply } \forall_{sk} : r \text{ to } \langle Af(\ell_1, \ldots, \ell_n) \rangle^V \vdash \times C \\
 & & | Af(\ell_1, \ldots, \ell_n) \vdash \times C \in \mathtt{PCNF}^+(Af(\ell_1, \ldots, \ell_n), V), V = \ell_1, \ldots, \ell_n\} \\
\mathtt{PCNF}^+(\langle \exists B \rangle^V, V) & = & \{\text{apply } \exists_{sk} : l \text{ to } \vdash \langle Bf(\ell_1, \ldots, \ell_n) \rangle^V \times C \\
 & & | \vdash Bf(\ell_1, \ldots, \ell_n) \times C \in \mathtt{PCNF}^+(Bf(\ell_1, \ldots, \ell_n), V), V = \ell_1, \ldots, \ell_n\}
\end{array}
$$

Each projection in this extended PCNF has strong quantifier rules which are properly placed, because they contain no contractions and the variables z introduced do not appear in the skolem terms between the inference and the root. The extension is not yet implemented though.

Grounded refutation of $\mathtt{CNF}^+(S)$:

$$\dfrac{\dfrac{\vdash P(0) \qquad P(0) \vdash P(s(0))}{\vdash P(s(0))}\,cut \qquad \dfrac{P(s(0)) \vdash P(s(s(0))) \qquad P(s(s(0))) \vdash}{P(s(0)) \vdash}\,cut}{\vdash}\,cut$$

Projections $\mathtt{PCNF}(S)$:

$$\begin{array}{c} P(0) \vdash P(0) \\ (\pi_{P(0)\vdash}) \end{array}$$

$$\begin{array}{c} P(s(s(0))) \vdash P(s(s(0))) \\ (\pi_{\vdash P(s(s(0)))}) \end{array}$$

$$\dfrac{\dfrac{P(u) \vdash P(u) \qquad P(s(u)) \vdash P(s(u))}{P(u) \to P(s(u)), P(u) \vdash P(s(u))}\,{\to}:l}{(\forall x_0(P(x_0) \to P(s(x_0)))), P(u) \vdash P(s(u))}\,\forall:l$$

$$(\pi_{P(u)\vdash P(s(u))}(u))$$

Refutation simulation:

$$\dfrac{\dfrac{\dfrac{(\pi_{P(0)\vdash}) \qquad (\pi_{P(u)\vdash P(s(u))}(0))}{P(0) \vdash P(0) \qquad (\forall x_0(P(x_0) \to P(s(x_0)))), P(0) \vdash P(s(0))}\,cut}{P(0), (\forall x_0(P(x_0) \to P(s(x_0)))) \vdash P(s(0))} \qquad \dfrac{\dfrac{(\pi_{P(u)\vdash P(s(u))}(s(0))) \qquad (\pi_{\vdash P(s(s(0)))})}{(\forall x_0(P(x_0) \to P(s(x_0)))), P(s(0)) \vdash P(s(s(0))) \qquad P(s(s(0))) \vdash P(s(s(0)))}\,cut}{(\forall x_0(P(x_0) \to P(s(x_0)))), P(s(0)) \vdash P(s(s(0)))}\,cut}{P(0), (\forall x_0(P(x_0) \to P(s(x_0)))), (\forall x_0(P(x_0) \to P(s(x_0)))) \vdash P(s(s(0)))}\,cut$$

$$\dfrac{P(0), (\forall x_0(P(x_0) \to P(s(x_0)))), (\forall x_0(P(x_0) \to P(s(x_0)))) \vdash P(s(s(0)))}{(\forall x_0(P(x_0) \to P(s(x_0)))), P(0) \vdash P(s(s(0)))}\,c:l$$

Figure 3.2: Proof import of the sequent $P(0), \forall x(P(x) \to P(s(s(x)))) \vdash P(s(s(0)))$

3.11 Implementation of the techniques in GAPT and ProofTool

The original aim of the General Architecture for Proof Theory (GAPT) [32, 33, 36] was to serve as a testing ground for our methods of cut-elimination [31, 35, 80, 83] and cut-introduction [47]. Over time other applications were added: the proof visualation tool ProofTool [34, 62] provides a flexible user-interface (see also section 3.13). Since an active software project changes rather quickly, we describe version 2.3[14] here, which is current at the time of writing.

GAPT provides fundamental data-structures and algorithms for the language of first-order logic and a simply typed lambda calculus. For example, church numerals (as defined in section 2.1.1) can be implemented in a few lines[15]:

```
def alpha( x: Var, a: LambdaExpression, n: Int,
        acc: LambdaExpression => LambdaExpression ): LambdaExpression =
  n match {
    case 0       => acc( x );
    case n if n > 0 => alpha( x, a, n - 1, e => App( a, acc( e ) ) )
    case _       => throw new Exception( "Church numerals must be positive!" )
  }

def num( n: Int ) = {
  val x = hov"(x:i)"
  val a = hov"(z:i>i)"
  Abs( a, Abs( x, alpha( x, a, n, identity ) ) )
}
```

[14] The github tag https://github.com/gapt/gapt/releases/tag/v2.3 points to the source and the GAPT homepage [33] provides a binary download available as https://logic.at/gapt/downloads/gapt-2.3.tar.gz.

[15] See also the file examples/ChurchNumerals.scala in the GAPT distribution.

```
def plus ( e1: LambdaExpression, e2: LambdaExpression ) =
   BetaReduction.betaNormalize( le"^(x:i>i) => ^(u:i) => (($e1 x) (($e2 x) u))" )

def times( e1: LambdaExpression, e2: LambdaExpression ) =
   BetaReduction.betaNormalize( le"^u ($e2 ($e1 u))" )
```

We define the n-fold iteration function α within the object num and close the term binding the start term `x` and iterator function `a`. The definitions addition and multiplication can be written via the string interpolator `le"..."` which allows us to mix parsing with the objects `e1` and `e2` from the scala environment. For demonstration purposes, both functions beta-normalize their result. Now we can check that $40 * 80$ is indeed $2000 + 2000$:

```
gapt> val fourthousand = times(num(50),num(80))
gapt> fourthousand == plus(num(2000), num(2000))
```

The system also provides several calculi to represent proofs. In the context of this thesis the focus on sequent calculus, resolution and expansion proofs is no surprise. The data-structures are sufficiently flexible to express multiplicative and additive versions of LK rules or to reason intuitionistically. Proofs can be entered manually, using the basic constructors and viewed in prooftool:

```
gapt> val x = Var("x", Ti)
x: at.logic.gapt.expr.Var = x

gapt> val fxx = parseLLKFormula("const F:i>i>o; var x:i;F(x,x)")
fxx: at.logic.gapt.expr.HOLFormula = F(x, x): o

gapt> val afxx = All(x, fxx)
afxx: at.logic.gapt.expr.HOLFormula = ∀x F(x, x)

gapt> val efxy = parseLLKFormula("const F:i>i>o; var x,y:i;(exists y F(x,y))")
efxy: at.logic.gapt.expr.HOLFormula = ∃y F(x, y)

gapt> val eefxy = Ex(x, efxy)
eefxy: at.logic.gapt.expr.HOLFormula = ∃x ∃y F(x, y)

gapt> val p1 = LogicalAxiom (fxx)
p1: at.logic.gapt.proofs.lk.LogicalAxiom =
[p1] F(x, x) ⊢F(x, x) (LogicalAxiom(F(x, x): o))

gapt> val p2 = ForallLeftRule(p1, Ant(0), afxx, x)
p2: at.logic.gapt.proofs.lk.ForallLeftRule =
[p2] ∀x F(x, x) ⊢F(x, x) (ForallLeftRule(p1, Ant(0), F(x, x): o, x, x))
[p1] F(x, x) ⊢F(x, x) (LogicalAxiom(F(x, x): o))

gapt> val p3 = ExistsRightRule(p2, Suc(0), efxy, x)
p3: at.logic.gapt.proofs.lk.ExistsRightRule =
[p3] ∀x F(x, x) ⊢∃y F(x, y) (ExistsRightRule(p2, Suc(0), F(x, y): o, x, y))
[p2] ∀x F(x, x) ⊢F(x, x) (ForallLeftRule(p1, Ant(0), F(x, x): o, x, x))
[p1] F(x, x) ⊢F(x, x) (LogicalAxiom(F(x, x): o))

gapt> val p4 = ExistsRightRule(p3, Suc(0), eefxy, x)
p4: at.logic.gapt.proofs.lk.ExistsRightRule =
[p4] ∀x F(x, x) ⊢∃x ∃y F(x, y) (ExistsRightRule(p3, Suc(0), ∃y F(x, y), x, x))
[p3] ∀x F(x, x) ⊢∃y F(x, y) (ExistsRightRule(p2, Suc(0), F(x, y): o, x, y))
[p2] ∀x F(x, x) ⊢F(x, x) (ForallLeftRule(p1, Ant(0), F(x, x): o, x, x))
[p1] F(x, x) ⊢F(x, x) (LogicalAxiom(F(x, x): o))
```

Using the backend for EProver, we can also find an automatic proof for $\forall x F(x,x) \rightarrow \exists x \exists y F(x,y)$:

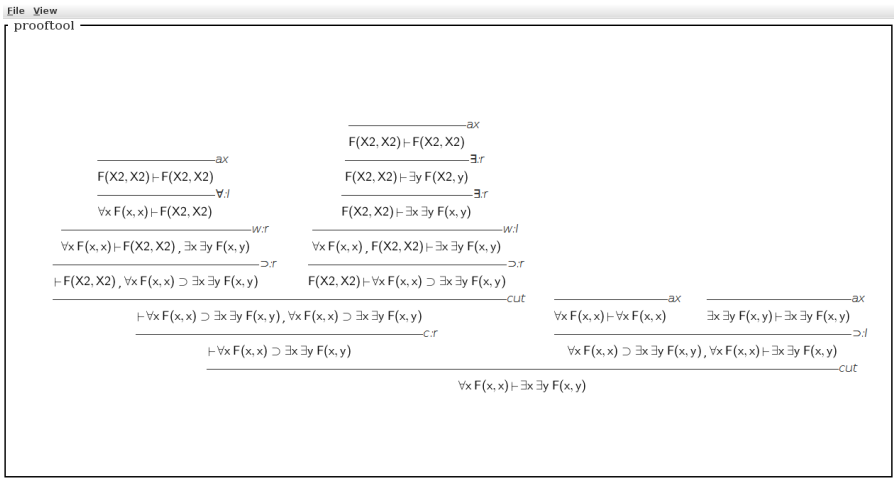
```
gapt> val Some(q) = EProver.getLKProof(Imp(afxx, eefxy))
q: at.logic.gapt.proofs.lk.LKProof =
[p10] ⊢∀x F(x, x) →∃x ∃y F(x, y) (ContractionRightRule(p9, Suc(1), Suc(0)))
```

```
[p9] ⊢∀x F(x, x) →∃x ∃y F(x, y), ∀x F(x, x) →∃x ∃y F(x, y) (CutRule(p4, Suc(0), p8, Ant(0)))
[p8] F(X2, X2) ⊢∀x F(x, x) →∃x ∃y F(x, y) (ImpRightRule(p7, Ant(0), Suc(0)))
[p7] ∀x F(x, x), F(X2, X2) ⊢∃x ∃y F(x, y) (WeakeningLeftRule(p6, ∀x F(x, x)))
[p6] F(X2, X2) ⊢∃x ∃y F(x, y) (ExistsRightRule(p5, Suc(0), ∃y F(x, y), X2, x))
[p5] F(X2, X2) ⊢∃y F(X2, y) (ExistsRightRule(p1, Suc(0), F(X2, y): o, X2, y))
[p4] ⊢F(X2, X2), ∀x F(x, x) →∃x ∃y F(x, y) (ImpRightRule(p3, Ant(0), Suc(1)))
[p3] ∀x F(x, x) ⊢F(X2, X2), ∃x ∃y F(x, y) (WeakeningRightRule(p2, ∃x ∃y F(x, y)))
[p2] ∀x F(x, x) ⊢F(X2, X2) (ForallLeftRule(p1, Ant(0), F(x, x): o, X2, x))
...
```

To obtain the sequent $\forall x F(x,x) \vdash \exists x \exists y F(x,y)$, we have to perform the equivalent of implication elimination in natural deduction. In sequent calculus, this amounts to the introduction of a cut:

```
gapt> val q1 = LogicalAxiom(afxx)
q1: at.logic.gapt.proofs.lk.LogicalAxiom =
[p1] ∀x F(x, x) ⊢∀x F(x, x) (LogicalAxiom(∀x F(x, x)))

gapt> val q2 = LogicalAxiom(eefxy)
q2: at.logic.gapt.proofs.lk.LogicalAxiom =
[p1] ∃x ∃y F(x, y) ⊢∃x ∃y F(x, y) (LogicalAxiom(∃x ∃y F(x, y)))

gapt> val q3 = ImpLeftRule(q1, Suc(0), q2, Ant(0))
q3: at.logic.gapt.proofs.lk.ImpLeftRule =
[p3] ∀x F(x, x) →∃x ∃y F(x, y), ∀x F(x, x) ⊢∃x ∃y F(x, y) (ImpLeftRule(p1, Suc(0), p2, Ant(0)))
[p2] ∃x ∃y F(x, y) ⊢∃x ∃y F(x, y) (LogicalAxiom(∃x ∃y F(x, y)))
[p1] ∀x F(x, x) ⊢∀x F(x, x) (LogicalAxiom(∀x F(x, x)))

gapt> val q4 = CutRule(q, Suc(0), q3, Ant(0))
q4: at.logic.gapt.proofs.lk.CutRule =
[p14] ∀x F(x, x) ⊢∃x ∃y F(x, y) (CutRule(p10, Suc(0), p13, Ant(0)))
[p13] ∀x F(x, x) →∃x ∃y F(x, y), ∀x F(x, x) ⊢∃x ∃y F(x, y) (ImpLeftRule(p11, Suc(0), p12, Ant(0)))
[p12] ∃x ∃y F(x, y) ⊢∃x ∃y F(x, y) (LogicalAxiom(∃x ∃y F(x, y)))
[p11] ∀x F(x, x) ⊢∀x F(x, x) (LogicalAxiom(∀x F(x, x)))
[p10] ⊢∀x F(x, x) →∃x ∃y F(x, y) (ContractionRightRule(p9, Suc(1), Suc(0)))
[p9] ⊢∀x F(x, x) →∃x ∃y F(x, y), ∀x F(x, x) →∃x ∃y F(x, y) (CutRule(p4, Suc(0), p8, Ant(0)))
[p8] F(X2, X2) ⊢∀x F(x, x) →∃x ∃y F(x, y) (ImpRightRule(p7, Ant(0), Suc(0)))
[p7] ∀x F(x, x), F(X2, X2) ⊢∃x ∃y F(x, y) (WeakeningLeftRule(p6, ∀x F(x, x)))
[p6] F(X2, X2) ⊢∃x ∃y F(x, y) (ExistsRightRule(p5, Suc(0), ∃y F(x, y), X2, x))
[p5] F(X2, X2) ⊢∃y F(X2,...

gapt> prooftool(q4)
```

PROOFTOOL, the graphical user-interface of GAPT, shows the resulting proof as:



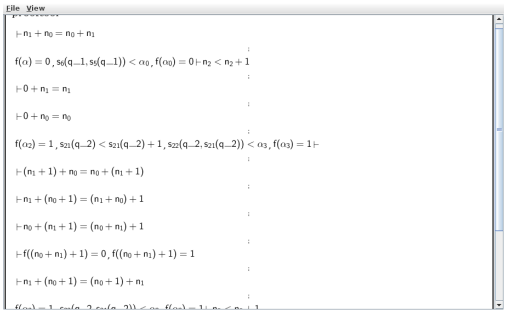
When we eliminate the cuts using Gentzen's method, we obtain a proof isomorphic (modulo names of eigenvariables) to the manual proof `p4` above:

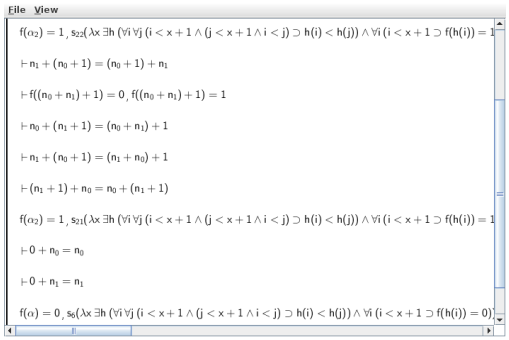
```
gapt> val q5 = ReductiveCutElimination(q4)
q5: at.logic.gapt.proofs.lk.LKProof =
[p4] ∀x F(x, x) ⊢∃x ∃y F(x, y) (ForallLeftRule(p3, Ant(0), F(x, x): o, X2, x))
[p3] F(X2, X2) ⊢∃x ∃y F(x, y) (ExistsRightRule(p2, Suc(0), ∃y F(x, y), X2, x))
[p2] F(X2, X2) ⊢∃y F(X2, y) (ExistsRightRule(p1, Suc(0), F(X2, y): o, X2, y))
[p1] F(X2, X2) ⊢F(X2, X2) (LogicalAxiom(F(X2, X2): o))

gapt> prooftool(q5)
```

With PROOFTOOL we can also visualize clause sets, for instance the ones obtained from the case study (see section 4.8.3), before and after lambda-lifting:

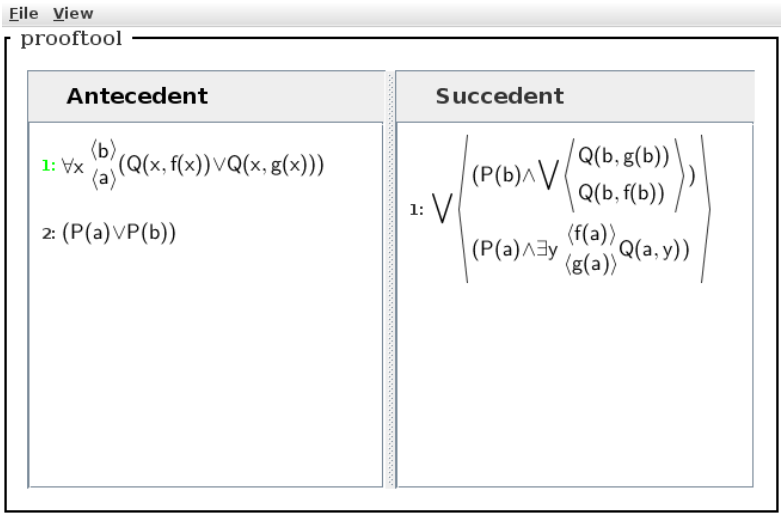
```
gapt> val css_nolabels = nTape2.css.map(_.map(x => x._2) )
css_nolabels: scala.collection.immutable.Set[at.logic.gapt.proofs.Sequent[at.logic.gapt.expr.
    HOLFormula]] =
Set(f(#v('\\alpha': i)) = 0,
s_5(λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
       ∀i (i < x + 1 →f(h(i)) = 0))) <
  s_5(λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
         ∀i (i < x + 1 →f(h(i)) = 0))) +
   1,
s_6(
   λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
        ∀i (i < x + 1 →f(h(i)) = 0)),
   s_5(λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
         ∀i (i < x + 1 →f(h(i)) = 0)))) <
  #v('\\alpha_0': i),
f(#v('\\alpha_0': i)) = 0
⊢
, ⊢0 + #v(n_0: i) < #v(n_1: i) + 1 + #v(n_0: i), ⊢0 + #v(n_1: i) < #v(n_0: i) + 1 + #v(n_1: i), f
    (#v('\\alpha_2': i)) = 1,
s_22(
   λx ∃h (∀i ∀j...
gapt> val (absmap, css_lifted) = replaceAbstractions( css_nolabels.toList )
absmap: at.logic.gapt.expr.fol.replaceAbstractions.ConstantsMap =
Map(λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
    ∀i (i < x + 1 →f(h(i)) = 0)) -> q_{1}, λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h
        (j)) ∧
    ∀i (i < x + 1 →f(h(i)) = 1)) -> q_{2})
css_lifted: List[at.logic.gapt.proofs.HOLSequent] =
List( ⊢#v(n_1: i) + #v(n_0: i) = #v(n_0: i) + #v(n_1: i), f(#v('\\alpha': i)) = 0,
s_6('q_{1}':i>o, s_5('q_{1}')) < #v('\\alpha_0': i),
f(#v('\\alpha_0': i)) = 0
⊢
#v(n_2: i) < #v(n_2: i) + 1, ⊢0 + #v(n_1: i) = #v(n_1: i), ⊢0 + #v(n_0: i) = #v(n_0: i), f(#v('\\
    alpha_2': i)) = 1,
s_21('q_{2}':i>o) < s_21('q_{2}') + 1,
s_22('q_{2}', s_21('q_{2}')) < #v('\\alpha_3': i),
f(#v('\\alpha_3': i)) = 1
⊢
, ⊢#v(n_1: i) + 1 + #v(n_0: i) = #v(n_0: i) + (#v(n_...
gapt> val qs = absmap.toList.map( x => HOLSequent(Nil, Eq(Const(x._2, x._1.exptype),x._1)::Nil) )
qs: List[at.logic.gapt.proofs.Sequent[at.logic.gapt.expr.HOLAtom]] =
List(
⊢
('q_{1}':i>o) =
  (λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
       ∀i (i < x + 1 →f(h(i)) = 0))),
⊢
('q_{2}':i>o) =
  (λx ∃h (∀i ∀j (i < x + 1 ∧(j < x + 1 ∧i < j) →h(i) < h(j)) ∧
       ∀i (i < x + 1 →f(h(i)) = 1))))

gapt> prooftool(css_nolabels)

gapt> prooftool(qs)

gapt> prooftool(css_lifted)
```

The clause sets are displayed as follows:





The last feature of PROOFTOOL to highlight is the viewer for expansion proofs. Weak quantifier nodes are by default drawn folded and can be dynamically expanded by the user. A mouse-click on an expansion displays the witness terms explicitly, a second click shows the expanded formula up to the next layer of weak quantifier nodes. GAPT contains an example proof of the sequent $\forall x(Q(x, f(x)) \vee Q(x, g(x))), P(a) \vee P(b) \vdash \exists x(P(x) \wedge \exists y Q(x, y))$ which has multiple witness terms. The nodes in the antecedent and the $P(b) \wedge (Q(b, g(b)) \vee Q(b, f(b)))$ node in the succedent are fully expanded while the a instance in the succedent shows only the witness terms of the expansion.



3.12 The LLK Input Format

The LLK language emerged from the need to input proofs in higher-order logic in a human readable style. At that time [34], the GAPT system could read legacy proofs in an XML format generated by the Handy LK [96] tool and the predecessor system CERES. The second proof input method implemented was a proof description language for schematic proofs, called

Handy LKS. Both options, formalization in HLK and in HLKS, were deemed unfit. In the first case, the original code written in C++ would have needed a reimplementation to keep GAPT self-contained. This was complicated by the fact that only a grammar in EBNF exists, but no semantics were documented. The second format did not improve too much over the direct use of the GAPT API. The most important drawback of both formats is that only the active formulas of an inference are mentioned. Hence, every inference can be easily evaluated by the user, whereas the context of an inference is hard to track. LCF style [43, 44] proof assistants like Isabelle [75, 78] and Coq [69] share this drawback. The approach taken there is to use an integrated development environment like Isabelle/jEdit [112] and CoqIDE [69], which shows open goals in its own window.

Since the development of an IDE was outside the scope of the thesis, we opted for a language which can be easily read by humans. Although the THF0 fragment of TSTP is flexible enough for our purposes, the explicit application operator makes deeply nested terms hard to read. This becomes already apparent when looking at equational axioms like associativity[16]. The main idea is to adapt the development pattern from working with the popular bussproofs LaTeXpackage. In fact, with an appropriate style file (see appendix D), the LLK format can by directly copy pasted into a prooftree environment.

An actual derivation is written with the full context, but the active formulas are inferred automatically. In the logical rules, weakenings and contractions are inserted when necessary. The order of the inferences is specified as a pre-order traversal of the proof tree i.e. as a stack. This allows us to develop a proof starting from the end-sequent, leaving yet unwritten parts as a hypothesis. Often such a hypothesis has a mathematical meaning and should be properly named. Therefore we added the possibility to define and instantiate subproofs. Since we formalized the arithmetical axioms as an equational theory, we soon noticed emerging patterns: Often, a theory equation is first instantiated and then applied to a subterm within a formula. Since the equality rules in the $LK_{skc=}$ calculus are binary, a series of such inferences quickly grows to the side, obscuring the actual computation. Inspired by deduction modulo [29, 30], we added a unary macro rule for an equational inference with an axiom. In contrast to the original, we perform only one step rewrites. To automatically infer the equation to be used, the theory needs to be declared beforehand. Additional rules like propositional autocompletion were easy to add, since the algorithm was already implemented [31].

An example can be seen in figure 3.3. Suppose we would like to represent lists using the associative symbol $*$ for the list constructor and nil for the empty list. To allow reordering elements, the symbol $*$ is also considered commutative. Using equational logic, it is possible to reverse the list $a * (b * (c * nil))$ with six inferences. We will now show how to formulate the corresponding proof in $LK_{skc=}$ using the LLK language. Since the axiom rules in $LK_{skc=}$ are either tautologies or instances of the reflexivity axiom, the theory axioms need to be added to the antecedent of the end-sequent, which results in $comm \wedge assoc \vdash a * (b * (c * nil)) = c * (b * (a * nil))$.

In order to prove this end-sequent, we first need to define the language. The CONSTDEC and VARDEC commands allow to to declare the types for each symbol. Symbol names are considered globally and the set of variable names must be disjoint from the constant names.

[16]The axiom of associativity is denoted as $((plus@X)@((plus@Y)@Z)) = ((plus@((plus@X)@Y))@Z))$.

```
% language declaration %
\CONSTDEC{a,b,c,nil}{i}
\VARDEC{x,y,z,rest}{i}
\CONSTDEC{∗}{i>i>i}

% names %
\CONSTDEC{THEPROOF, AX, symm,assoc}{o}
\CONSTDEC{FIRSTEQ}{i>o}

% axiom declarations %
\AXIOMDEC{symm}{}{x∗y=y∗x}
\AXIOMDEC{assoc}{}{x∗(y∗z)=(x∗y)∗z}

% a proof reversing the list [abcd] %
\AX{}{c∗(b∗(a∗rest)) = c∗(b∗(a∗rest))}
\WEAKL{AX}{c∗(b∗(a∗rest)) = c∗(b∗(a∗rest))}
\EQAXIOM{b∗(a∗rest) = (b∗a)∗rest}{AX}{c∗((b∗a)∗rest) = c∗(b∗(a∗rest))}
\CONTINUEWITH{FIRSTEQ(rest)}

\CONTINUEFROM{FIRSTEQ(nil)}{AX}{c∗((b∗a)∗nil) = c∗(b∗(a∗nil))}
\EQAXIOM{c∗((b∗a)∗nil) = (c∗(b∗a))∗nil }{AX}{(c∗(b∗a))∗nil = c∗(b∗(a∗nil))}
\EQAXIOM{(b∗a)∗c = c∗ (b∗a)}{AX}{((b∗a)∗c)∗nil = c∗(b∗(a∗nil))}
\EQAXIOM{a∗b=b∗a}{AX}{((a∗b)∗c)∗nil = c∗(b∗(a∗nil))}
\EQAXIOM{(a∗b)∗(c∗nil) = ((a∗b)∗c)∗nil}{AX}{(a∗b)∗(c∗nil) = c∗(b∗(a∗nil))}
\EQAXIOM{a∗(b∗(c∗nil)) = (a∗b)∗(c∗nil)}{AX}{a∗(b∗(c∗nil)) = c∗(b∗(a∗nil))}
\CONTINUEWITH{THEPROOF}
```

Figure 3.3: Example of an LLK proof: Reversing a list of 3 elements

In order to instantiate proofs and axioms easily, their names are represented by atom formulas. Consequently, they need to be declared as well.

In the next step, we define the background theory. Each AXIOMDEC associates a name to the sequent representing the axiom, in this case we use $assoc$ and $comm$. To represent the theory in the proof, we use the special atom AX in the antecedent, which is automatically defined as the conjunction of the universal closure of each axiom. Whenever an axiom is needed, its instance can be derived from AX by a series of conjunction left and universal left rules.

Instead of manually instantiating each axiom, we can use the EQAXIOM macro rule, which does the job for us. The proof in figure 3.4 directly mirrors the equational reasoning via unary equation rules. In each EQAXIOM rule, we provide the instance of the axiom used as an additional parameter, the rest is automatically generated. When we have a look at the unfolded proof (figure 3.5), the binary equation inferences let it grow horizontally to a point, where the proof is unreadable. There are two ways to handle this problem: either we decompose the proof into readable parts or we change the format of the proof. Here, we concentrate on the first solution, the second is addressed in section 3.13.

In the particular example, we will focus on the first equational inference of the proof, since the others are similar. The CONTINUEWITH statement allows us to give this sub-proof the name FIRSTEQ. Later on, we refer to this sub-proof with a CONTINUEFROM statement. Since we might want to reuse this step, instead of the constant nil we use the variable $rest$ throughout the FIRSTEQ proof. Consequently, the parameter $rest$ needs to be passed as an argument to FIRSTEQ when the sub-proof is named and has to be instantiated by referring to FIRSTEQ(nil) later on. We can now continue with the remaining EQAXIOM steps, naming the full proof just THEPROOF.

$$\dfrac{\dfrac{\vdash c*(b*(a*rest))=c*(b*(a*rest))}{AX\vdash c*(b*(a*rest))=c*(b*(a*rest))}\,w:l}{AX\vdash c*((b*a)*rest)=c*(b*(a*rest))}\,EQAX: b*(a*rest)=(b*a)*rest$$

$$(FIRSTEQ(rest))$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(FIRSTEQ(nil))}{AX\vdash c*((b*a)*nil)=c*(b*(a*nil))}}{AX\vdash (c*(b*a))*nil=c*(b*(a*nil))}\,EQAX: c*((b*a)*nil)=(c*(b*a))*nil}{AX\vdash ((b*a)*c)*nil=c*(b*(a*nil))}\,EQAX: (b*a)*c=c*(b*a)}{AX\vdash ((a*b)*c)*nil=c*(b*(a*nil))}\,EQAX: a*b=b*a}{AX\vdash (a*b)*(c*nil)=c*(b*(a*nil))}\,EQAX: (a*b)*(c*nil)=((a*b)*c)*nil}{AX\vdash a*(b*(c*nil))=c*(b*(a*nil))}\,EQAX: a*(b*(c*nil))=(a*b)*(c*nil)$$

$$(THEPROOF)$$

Figure 3.4: The LaTeXrendering of the reverse list LLK input

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{((b*(a*nil))=((b*a)*nil))\vdash((b*(a*nil))=((b*a)*nil))}{(\forall z((b*(a*z))=((b*a)*z)))\vdash((b*(a*nil))=((b*a)*nil))}\,\forall:l}{(\forall y(\forall z((b*(y*z))=((b*y)*z))))\vdash((b*(a*nil))=((b*a)*nil))}\,\forall:l}{(\forall x(\forall y(\forall z((x*(y*z))=((x*y)*z)))))\vdash((b*(a*nil))=((b*a)*nil))}\,def}{assoc\vdash((b*(a*nil))=((b*a)*nil))}\,\wedge:l}{symm\wedge assoc\vdash((b*(a*nil))=((b*a)*nil))}\,def}{AX\vdash((b*(a*nil))=((b*a)*nil))}\qquad \dfrac{\vdash((c*(b*(a*nil)))=(c*(b*(a*nil))))}{AX\vdash((c*(b*(a*nil)))=(c*(b*(a*nil))))}\,w:l\,=:r$$

$$\dfrac{AX,AX\vdash((c*((b*a)*nil))=(c*(b*(a*nil))))}{AX\vdash((c*((b*a)*nil))=(c*(b*(a*nil))))}\,c:l$$

$$(FIRSTEQ(nil))$$

Figure 3.5: Reverse list example: binary equation rules after macro expansion make the proof hardly readable

```
Apostrophe          :=  '''
BSlash              :=  '\'
InfixOp             :=  "=" | "!=" | "<" | "<=" | ">" | ">=" | "+" | "*" | "-" | "/"
UnLogicalOp         :=  "-" | (BSlash ~ "neg" )
BiLogicalOp         :=  "&" | "|" | "->" |
                        (BSlash ~ ("land" | "lor" | "impl" ))
Quantifier          :=  "all" | "exists" |
                        (BSlash ~ ("forall" | "exists"))
NameParticle        :=  (['a'-'z''A'-'Z''0'-'9'] |Apostrophe)+
UpperLowerParticle  :=  (('^' | '_') ~ ('{' ~ NameParticle ~ '}'))?
Name                :=  BSlash ~ (NameParticle ~ UpperLowerParticle ~ (NameParticle |'[' |']' )* )+
Atom                :=  Name |
                        Name ~ '(' ~ (Expr ~ (',' ~ Expr)*)?  ')' |
                        Expr ~ InfixOp ~ Expr
Abs                 :=  '(' ~ BSlash ~ Name ~ "=>" "PExprOrAtom"  ')'
App                 :=  '(' ~ '@' ~Expr+  ')' |
                        '(' ~ BSlash ~ "apply{" ~Expr+ '}'  ')' |
PExprOrAtom             Atom | PExpr | Abs | App
QuantifierExpr      :=  (Quantifier ~ Name)+ ~PExprOrAtom
PExpr               :=  '(' ~ Expr | QuantifierExpr  ')'
Expr                :=  PExprOrAtom |
                        UnLogicalOp ~ PExprOrAtom
                        PExprOrAtom  ~  BiLogicalOp ~ PExprOrAtom
Formula             :=  Expr
```

Figure 3.6: Formula Grammar

```
OneAtomList      :=  '{' ~ Atom ~ '}'
OptAtomList      :=  '{' ~ Atom? ~ '}'
FormulaList      :=  '{' ~ (Formula ~ (',' ~ Formula)*)? ~ '}'
Sequent          :=  FormulaList ~ FormulaList
IntroRule        :=  BSlash ~ "AX" ~ Sequent |
                     BSlash ~ "CONTINUEFROM" ~ OneAtomList ~ Sequent |
                     BSlash ~ "AUTOPROP" ~ OneAtomList ~ Sequent
Uname            :=  "WEAKL" | "WEAKR" | "CONTRL" |
                     "CONTRR" | "ANDL" | "ORR" | "IMPR" |
                     "NEGL" | "NEGR" | "DEF"
URule            :=  Proof ~ BSlash ~ Uname ~ Sequent
Qname            :=  "ALLL" | "ALLR" | "EXL" | "EX"
QRule            :=  Proof ~ BSlash ~ QName ~ OptAtomList ~ Sequent
Bname            :=  "CUT" | "ANDR" | "ORL" | "IMPL" | "EQL" | "EQR"
BRule            :=  Proof ~ Proof ~ BSlash ~ Bname ~ Sequent
NamedAtomList    :=  '{' ~ Atom ~ ':' ~ Atom ~ '}'
EQAxRule         :=  Proof ~ BSlash ~ "EQAXIOM" ~ NamedAtomList ~ Sequent
InstAxRule       :=  Proof ~ BSlash ~ "INSTAXIOM" ~ OneAtomList ~ Sequent
Proof            :=  IntroRule | URule | BRule | QRule
CompletedProof   :=  Proof ~ BSlash ~ "CONTINUEWITH" ~ OneAtomList
```

Figure 3.7: Proof Grammar

3.13 Using Sunburst Trees to navigate large proofs

This section is based on a publication [62] on PROOFTOOL [34] as part of GAPT (see also section 3.11) and focuses on the exploration of large proofs.

The need for visualizing data precedes the invention of computers. Even so, the large data processed by computers made this need more explicit and initiated much research in data visualization and particularly in tree visualization. For example, the traditional disk usage analyzers were all implemented as trees, with directories and files represented by nodes and edges denoting the containment relation. In the last decades, the increase in disk space and the increase in number of files that followed, prompted the design of new tree visualization methods which will be more space efficient. One of the first methods was TreeMap [92] which divides a box into several smaller boxes representing the subtrees. Other algorithms made the nodes implicit by drawing fractals [70,82], added a third dimension [52,63,74] or used hyperbolic and other radial approaches to better group subtrees [51, 60, 97, 115]. Treevis.net [85], a visual bibliography of tree viewers, now contains more than 270 different algorithms.

Sequent calculus proofs are often depicted as trees and in fact, the tree representation was used from the very beginning. Gentzen's representation for sequent calculus proofs can be seen as a variant of an algorithm by Donald Knuth [59]. The child nodes are horizontally aligned in the distance of the width of their respective subtrees with their parent node being aligned centrally between them. The vertical alignment is determined by the distance from the root.

However, although this presentation seems natural, it is not well suited for large proofs as their structure is no longer visible. Even tracing the ancestors of a formula is cumbersome, since the distance between parent inferences can be very large.

Despite the abundant research done in the field of tree visualization and the fact that proofs are normally represented as trees, little was done so far in integrating these advancements into tools for proof visualization. In fact, the first viewer which was integrated into PROOFTOOL was a traditional tree viewer. Proof General [3], which also manages the proof visualization for provers such as Coq [53], supports the traditional tree view as well. Other systems like $L\Omega UI$ [93] and Theorema [114] provide a structural overview in form of a DAG and a tree, respectively. However, their main focus lies on human-readable proofs where the formula level is directly contained in the text. One of the few graphical user interfaces which deviates is IDV [106]. It renders DAG proofs in the TSTP format [101] using the spring layout [27]. This layout turned out to be insufficient for our needs as is discussed in [34]. The reason that many advancements in tree visualization are only slowly reaching the proof theory community may primarily lie with the fact that only few of the provers care about a visual presentation of the generated object, if they generate it at all. Nevertheless, proof visualization is a crucial tool for analyzing large proofs like the ones we encounter in our work. Therefore, we find it important to search for and integrate efficient tree viewers.

We propose some criteria for visualizing sequent calculus proofs and use them to analyze the existing layouts. We argue that Sunburst Trees [97] are the most adequate layout and develop a new viewer for PROOFTOOL, the graphical user interface of the GAPT framework, which is based on them. The viewer allows to display the structure of the whole proof at once, to easily identify similar subproofs, to zoom in to relevant parts and to see the relevant inference details.

3.13.1 Criteria for Visualizing Sequent Calculus Proofs

When using the traditional tree layouts, wide node labels often stretch the width of the tree and deform its structure. One reason for that is that the context formulas of an inference need to be repeated along many branches. Moreover, it can also happen that the main or auxiliary formulas become overly wide themselves. Therefore, it is helpful to completely separate the tree structure from the information about the inference itself. Since sequent calculus and the inference viewer work well on the sequent level, we concentrate on the requirements for the structural layout.

In this section, we compile a small set of requirements which were identified as critical for our proof analysis. We have tried comparing our conclusions with other works concerning the aesthetics of mathematical proofs. Surprisingly, they often stay only on an abstract level. Hardy [46] names *unexpectedness*, *inevitability* and *economy* as aesthetic properties. The first refers to an element of surprise when a conclusion is reached, which has similarities to narratives [22]. The second stands for a detailed, convincing deduction whereas the third means restricting a proof to the minimal steps in order to prove the theorem. Some works deepen these concepts and introduce case studies [77], but we know of no work which details the relation between graphical notations and the aesthetics of a proof. Therefore, the requirements given here are the result of the authors' own involvement with the formalization of mathematical proofs.

1. *Displaying large proofs* is one of the most important factors. Proofs containing thousands of inference steps can become very hard to read. We would like to find the most efficient tree representation. For example, despite the fact that proofs are traditionally denoted as trees, the edges between the nodes play a very small role and are not space efficient. Another aim is to be able to represent the full proof on a single screen in a comprehensive way. This is not only useful for exporting purposes, but for tracking changes after the application of proof transformations, such as substitution, skolemization or cut-elimination.

2. *Distinguishing between different kinds of rules* is important as some rules, like quantifier rules, give information about the content of the proof while others, like contractions, give information about the shapes of proofs. Different coloring of rules is one way to distinguish between them.

3. Without *easy navigation*, one would not be able to follow the logical progress of the proof. The sub-proof relation should always be obvious and easy to navigate.

4. In many cases, *formula ancestor information* is important in order to relate a sequent with the atomic formulas by which it is implied.

5. Proofs have many uses and one would sometimes like to *focus on different aspects of the proof*. Proof complexity, different instantiations, cuts complexity and contractions may all be important for a prospective viewer of a proof.

6. The ability to relate *shape of proofs* and sub-proofs to their content might also be an important factor since it might allow us to detect redundancies and similarities of content.

3.13.2 Choosing the proper tree visualization

One of the most comprehensive bibliographies for research on tree visualization is Treevis.net [85] which contains over 270 different algorithms. Consequently, it is a challenge to pick an adequate algorithm out of the numerous ones which have been published. However, Treevis.net also provides a categorization of the techniques in terms of the criteria of dimensionality (2D, 3D or hybrid), representation (implicit, explicit or hybrid) and alignment (axis-parallel, radial or free). In this section we will first explain our choices with regard to the categories mentioned. We then identify the algorithm satisfying our category requirements and show that it also meets our visualization criteria.

We decided to focus on two-dimensional representations, since there is no general additional structure which could be mapped to the third dimension. Although both explicit and implicit representations of edges would suit our purposes, the later allows us to expect a more compact layout. Consequently, we would like to focus on this case. This is additionally motivated by the fact that sequent calculus proofs often contain a high amount of unary rules, where the edge is then redundant. From the algorithms meeting these requirements, we now excluded those which do not meet criterion 6 from Section 3.13.1. The remaining options were unexpectedly low in number. A reason for that is that the large class of layouts based on TreeMap divides a box into equal sub-parts for each subtree. The problem there is that in a series of identical subproofs connected by binary inferences, each subproof has half the size of the preceding one, making it nigh impossible to recognize their similarity. Fractal layouts have similar problems, whereas grid embeddings [84, 116] do not reflect the similarity of subtrees.

What remained were explicit axis-parallel layouts and implicit radial layouts. The first class consists of improvements on the classical Tidy Tree algorithm [113], whereas the later centers around representing the tree from the root outwards.

Of special appeal to our applications was the Sunburst Tree [97]. It is particularly efficient for *displaying large proofs* due to its radial shape and the fact that it eliminates all edges. One can easily *distinguish different kinds of rules* by setting different colorings. The user can group the rules by their function in the proof and thus separate the proof into parts with logical, equational or quantifier inferences. Together with the branching structure, (sub-)proofs are already distinguishable from each other without referring to the formula level. *Navigation* is similarly simple. A single click into a subproof shrinks the original proof to half its size. At the same time the selected subproof is projected onto the circle around the full proof, giving it sufficient space to see detailed inferences. This kind of stretching can impede the identification of similar structures within the proof. Nonetheless, the combination with the full view allows a comparison on the same level. In order to *focus on different aspects of proofs*, a customization of two parameters is possible. By changing both the coloring scheme as well as the inference width ratio, one can single out instantiations denoted by weak quantifiers, different subproof complexities and other aspects. Another strong point of the Sunburst viewer is its ability to relate content to *proof shape*. This is again achieved by inference coloring and width ratio and by its efficient presentation of a whole proof on a single screen. Finally, a radial layout is also helpful in that it can always be drawn into a square, leaving room on the screen for the inference information.

There is one requirement where the Sunburst viewer falls far behind the traditional tree viewers and this is with keeping *formula ancestor information*. The relationship between a

formula and its ancestor, while easily displayed in the traditional viewer, cannot be represented in Sunburst. This raises another important requirement for a useful proof visualization tool, its ability to support different viewers and the switching between them.

3.13.3 Integration in PROOFTOOL

We have integrated the Sunburst view as an option accessible from the menu. Choosing this option loads the Sunburst view into a separate dialog window. In addition to the proof, which is displayed on the left side, we display also an inference panel. The information in this panel contains details about the inference: its type, its primary and auxiliary formulas, and quantifier instantiation information, if applicable.

In the remainder of this section, we will describe the new interface in terms of the conditions given in Section 3.13.1. To emphasize what is written, we have inserted snapshots of views of actual proofs. The appendix contains information about the names of each proof and of how to load it using our system.

Displaying of large proofs. The traditional Gentzen layout contains abundant and redundant white spaces, not only due to its use edges, but also because it has to create extra horizontal spaces between premises of binary inferences. Therefore, proofs with many binary inferences, even if the formulas are hidden, are too wide to fit on the screen. An example of this can be seen in Figure 3.8.

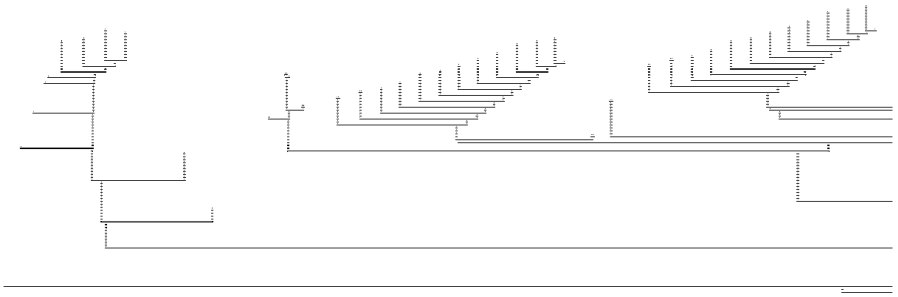


Figure 3.8: A small part of the Gentzen view of a proof with more than 2000 nodes.

Projecting the sequent calculus proof to a circle allows roughly four times more space to render the inferences of a certain level[17]. Since the formulas are hidden, an inference is an easily clickable section of the disc, covering the whole area below its parents. Also, hovering over an inference with the mouse cursor triggers a darkening of its bounds. This is particularly helpful when tracing a formula throughout a proof, as one can then easily identify branching without the need to zoom in.

[17]Let us assume the Gentzen proof to be an isosceles triangle with base length w_1, and height d. If we further assume the window is maximized, we can estimate the ratio $w_1 : d = 16 : 10$. Then the height d is also the radius of the Sunburst tree giving it circumference $w_2 = 2d\pi$. The ratio of the two lengths is then $w_1 : w_2 = \frac{20\pi}{16} \approx 4$.

As an example, Figure 3.9 shows the n-occurrences tape proof from section 4.8 with more than 2000 inferences together with a zoomed in subproof. Comparing the two figures shows that even if we hide all inference information in the Gentzen view, we will still not be able to fit this proof on the screen. In contrast, the Sunburst view allows us to identify the main parts which constitute the proof: the top and bottom side have the same shape, for they contain the same reasoning structure on different terms. Only a small proof, which gives rise to a case distinction, is situated on the left hand side. Just by hovering over or selecting the cut-formulas (colored green), the user can identify the three parts of the proof. The right hand side of Figure 3.9 shows a zoom into one of the proof instances just mentioned.

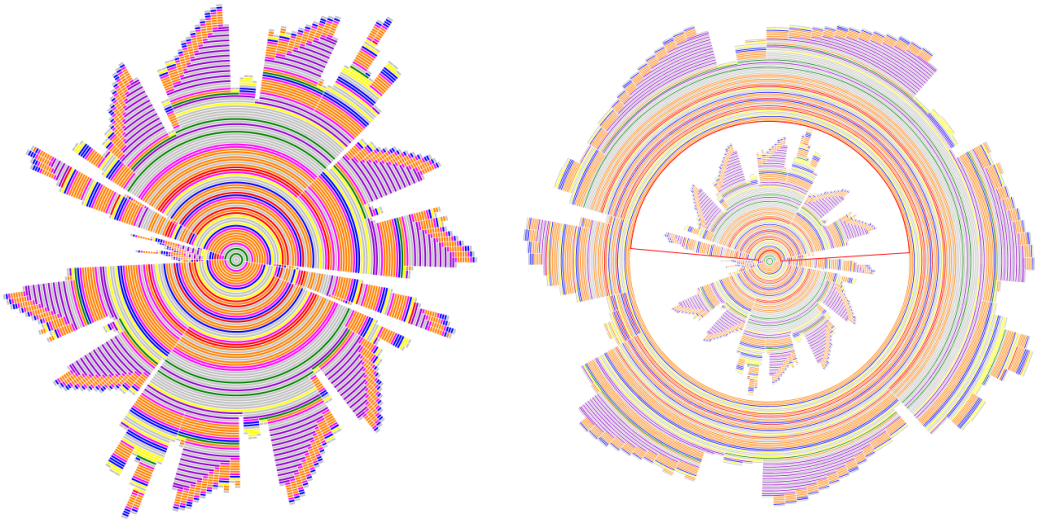


Figure 3.9: Sunburst view of a large proof in full view (left) and zoomed in (right).

Distinguishing between different kinds of rules. This is easily achieved in Sunburst view by coloring the inference depending on the rule type. In order to obtain the highest contrast, we assigned the colors of the rainbow (see Figures 3.9 and 3.10) to groups of rules according to Table 3.1. The relative size of subtrees to each other is adjustable by defining a

Cut	green	Unary Logical Rule	orange	Strong Quantifier Rule	red
Structural Rule	gray	Binary Logical Rule	yellow	Weak Quantifier Rule	blue
Axiom	gray	Equational Rule	violet	anything else	magenta

Table 3.1: Rules coloring schemes.

so called weight. At the moment, the weight of a subtree is just the number of inferences. Depending on the application, one can imagine metrics which prioritize specific rules or even specific inferences.

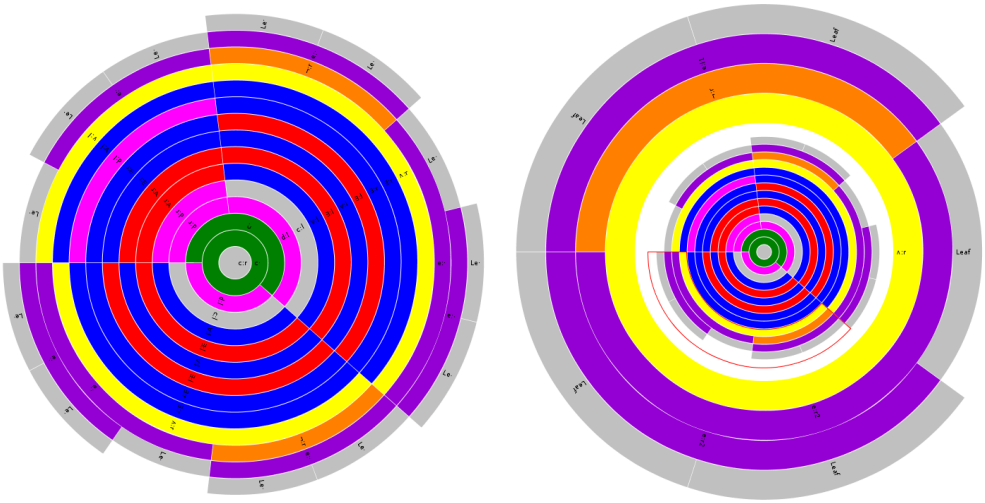


Figure 3.10: A combinatory proof (left) with a zoom into one of its subproofs (right).

Easy navigation. Navigation is very easy in Sunburst, as can be seen in Figure 3.10. A single click on a subproof shrinks the whole proof to half of its size while displaying the subproof on its outer ring. Navigating backward can be done using the nested original proof.

One drawback of this form of navigation is that zooming into a subproof distorts its shape, affecting the user ability to understand the structure of the proof. Zooming into a subproof distorts all inferences in the same way. We therefore believe that this has only a minor affect on understanding the proof structure, since a human user can easily compensate for this fixed distortion.

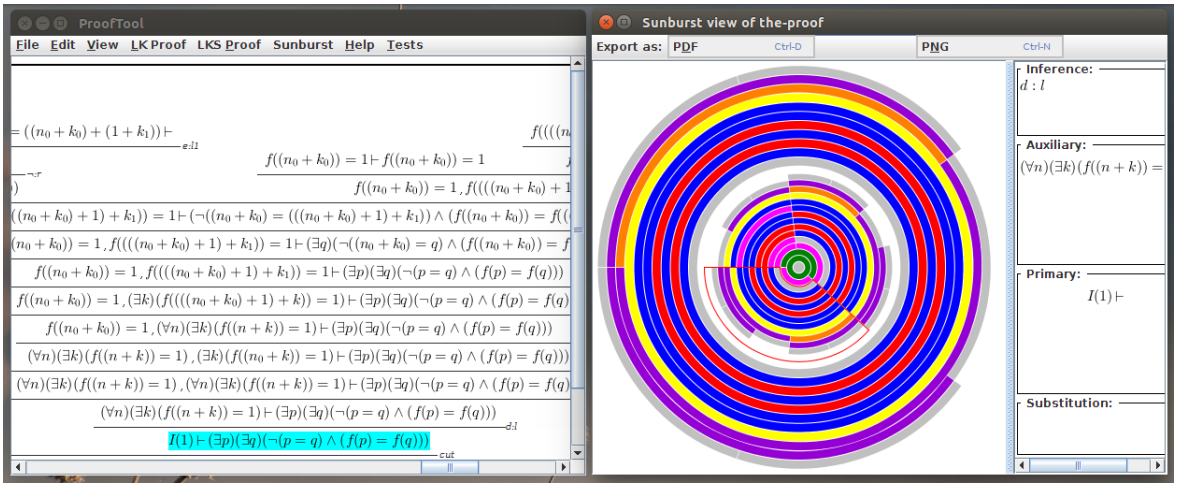


Figure 3.11: Synchronizing the two views.

Formula ancestor information. The sunburst view window is divided into two parts as shown in Figure 3.11. The first part shows the structure of the proof, while the second part gives additional information about the selected node. This includes the inference name, its auxiliary and principal formulas, and the substitution used, if any. But still, this information is not enough to see the ancestor relationship as well as it is possible in the Gentzen layout. The two views complement each other in this aspect as is illustrated in Figure 3.12.

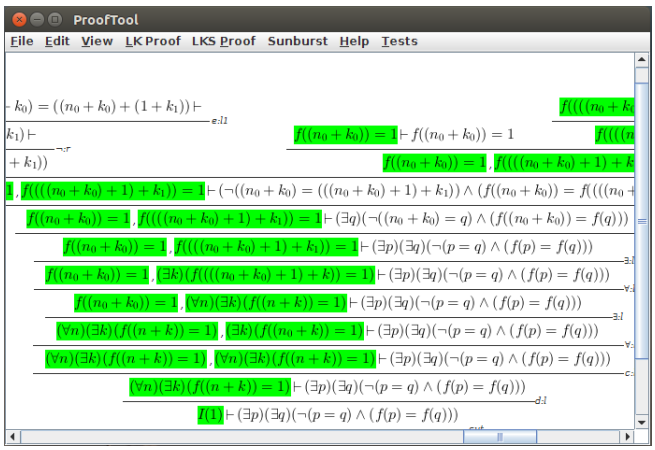


Figure 3.12: The cut-formula ancestors marked green.

Focus on different aspects of the proof. In order to display different aspects of the same proof, one can take advantage of the possibility to customize the colors and width ratio of inferences in Sunburst. We would like to have a set of such pre-defined customizations which will emphasize different aspects, such as sub-proof and cut complexities, variable instantiations and specific rules and inferences. We plan to implement this feature in the near future.

Shape of proofs. In some situations where the formula level is obscured, it is helpful to concentrate on the structure of the proof. In the following we describe two phenomenons we encountered.

Proof transformations often keep the structure intact. Some proof transformations such as elimination of definition rules and skolemization strongly change the proof on a formula level, but only slightly modify the structural layout. Nonetheless, locating an inference in the Gentzen layout with the find function of PROOFTOOL becomes virtually impossible, since the subterms allowing a unique identification of a formula often have changed. The Sunburst view allows to use the proof structure to find the inference. For example, it is not always clear how a skolem term ends up in a weak quantifier inference, since the term might be carried over from a different part of the proof. In the Sunburst view, we can navigate to this inference and use both views for further investigation.

Understanding proof arguments. In the process of formalizing a proof, one might not recognize all the possibilities where the proof can be generalized. In the Sunburst view, structural similarities are easier to spot and can then be checked whether a generalization is indeed possible.

As an example, we can look at subsequent instances of a formalization of Fürstenberg's proof of the infinity of primes [5]. Here the schematic nature of the proof was already taken into account during formalization, but now the induction argument becomes clearly visible (see Figure 3.13).

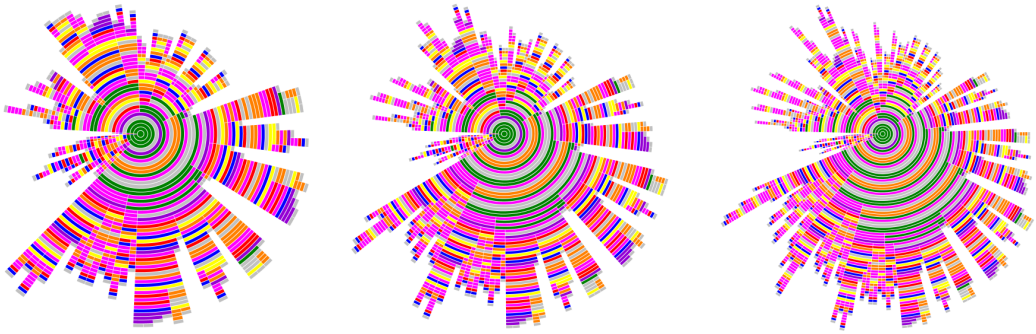


Figure 3.13: Instances 1, 2 and 3 of the formalization of Fürstenberg's proof of the infinity of primes.

3.13.4 Further Directions

In this section, we have explained the issues that standard tree viewers have when faced with large proofs. We have then identified various criteria for a suitable tree visualization and analyzed the available algorithms with respect to them. Our results show that Sunburst Trees seem to be the most adequate structural layout for viewing sequent calculus proofs. The global structure can be better seen than in standard layouts, which makes large proofs readable. We found identifying inferences, navigation, and tracing derivations superior to the Gentzen layout. At the same time, formula or context intensive tasks such as identifying the ancestor relationship are better left to the latter. The integration of the Sunburst viewer alongside the Gentzen viewer in PROOFTOOL demonstrate how well these two complementary layouts interact with each other.

Some improvements are still of interest to us. Foremost, multiple Sunburst trees can be represented by a forest structure. We plan to take advantage of this in two ways. First, larger proofs usually consist of several subproofs solving partial problems. In other words, they can be represented as a forest with links [35] to their subproofs. This division is often explicitly contained in the proof input language[18], but the proof object itself usually does not carry on this information. If all proof transformations are adjusted to carry on the link structure, the result can be divided into a set of proofs, making the meta-structure of the proof visible. Moreover, it

[18]The proof languages hlk, shlk and llk defined in the context of GAPT can be seen as examples for this.

is also possible to display DAGs by converting them into forests. This would enable the viewer to display resolution refutations. The high reuse of clauses in a refutation might fill the forest with many tiny trees, but this is open to experimentation.

A practical improvement is the addition of viewing profiles. By setting different color schemes and inference width ratios for each profile, we can customize the viewer to better display different aspects of proofs, like subproof complexity and instantiations.

The last two planned improvements are on the level of formulas. First, we might increase the readability of large formulas by replacing them with new symbols. In addition, we would like to improve the search facilities in PROOFTOOL. Right now, searching for a specific formula in the Gentzen view mark all occurrences of the formula. We plan to add a similar facility to the Sunburst view. One idea is to put the search results into a list in a new window, thus allowing the user to browse through the search results and jump to the right inference.

4.1 Practical Aspects of the Analysis Process

Even though the process of proof analysis is usually presented as a linear development, it is in fact an iterative process similar to developing a mathematical proof or writing a program. Often, one starts with an idea for proving a particular theorem. Filling in the concrete details might lead to a restatement of the theorem in more suitable terms. In some cases, the theorem even needs some restrictions in order to hold. In others, the original method can not bridge a gap and we might need to change the entire approach. In order to discuss the analysis of our case study, we will also show the development of the proof before we look at the results.

Before we come to that, we give a quick overview of the steps necessary to obtain the atomic-cut normal form and an expansion sequent of our proof (see Figure 4.1). Having formalized an input proof in LK with first-order equality and definitions(1), the proof needs to be transformed into $LK_{skc=}$ to allow the extraction of the characteristic sequent set (2-4). The sequent set is then reduced by tautology elimination, subsumption and condensation. In the case we are attempting to find a first-order refutation, an appropriate embedding has to be performed (5). Now we enter the critical phase of finding an actual refutation (6). This mainly involves experimentation with prover parameters. Another approach further massages the input problem to better suit the prover (see section 3.9). If we fail to obtain a refutation, we have to return to one of the previous steps. Having finally found a refutation, we need to transform the prover output to R_{al} (7), before we create an atomic cut-normal form of the input proof and extract the expansion sequent. The pre/postprocessing steps also include trivial transformations like the adjustment of variable and constant names to the language appropriate for this step. Most of these steps can be automatized, only steps 1, 5 and 6 need human interaction.

1. Proof formalization in LK with definitions

2. Eliminate definitions & expand tautological axioms

3. Conversion to $LK_{skc=}$ with definitions

4. Extract struct, convert to characteristic sequent set & calculate projections

5. Sequent set pre-processing

6. External theorem prover

7. Refutation post-processing

8. Construct proof in ACNF & extract expansion sequent

Steps involving human interaction are denoted in bold face.

Figure 4.1: Analysis Process

4.2 Choosing a Problem

Both, CERES^{ω} and CERES_S, were developed to overcome a drawback of the original CERES method, which was uncovered during the analysis of Fürstenberg's proof of the infinitude of primes [5]: Theorem provers usually did not manage to refute the extracted clause sets. What lead to success in the prime analysis was to use a schematic proof specification which is compiled for specific instances. Even there, only small instances could be solved by theorem provers. Despite enormous improvements in automated theorem proving techniques, the situation is unchanged six years later. Vampire, E-Prover, SPASS and Prover 9 all manage to refute the clause set coming from the instance showing that there are more than two primes, but each of them fails to find a proof for higher instances.

Later on, an (unpublished) attempt of analyzing a second-order formalization of the Fürstenberg proof in CERES^{ω} was stuck in the refutation phase. It was even possible to extract a first-order characteristic sequent set, but it turned out as too big of a challenge to handle automatically. Since the set also resisted manual analysis, the investigations had to stop there.

Therefore, in order to gain better insights into the workings of CERES^{ω}, the proof investigated needed to be simpler. A straightforward way to reduce complexity is to restrict quantification to individual and function variables only. This way, instantiations are only performed on the term level. Even if the terms contain quantifiers, they can not be used on the logical level without a projection, which must come from an instantiation of a predicate variable. For example, a sequent containing the formula $X(\exists x Q(x))$ can only have an existential quantifier rule applied, when X is substituted by the identity projection $\lambda Y\ Y$. As a consequence, during the refutation of the characteristic sequent set, the absence of predicate variables means that the logical and quantifier rules will not be applied after bringing the initial sequents into clause form. In particular, if we expand non-atomic sequent calculus axioms to their axiom form, our initial sequent set is already in clause form. With only the resolution rules of cut and substitution

applicable, the refutations are closer to first-order ones. Therefore, given suitable substitutions for function variables, it was possible to use a first-order theorem prover in case the higher-order provers failed.

To further increase the chances of finding a refutation, we turned to a problem we had studied before [4]: the infinite pigeon hole principle. Ratiu and Trifonov [79] had already managed to extract two different functionals from the infinite pigeon hole principle using A-translations with realizability and the Dialectica interpretation [79]. Therefore we were interested in what functional we would obtain using the CERES^{ω} method.

4.3 The Infinite Pigeon Hole Principle

There exist several variations of the pigeon hole principle. The underlying metaphor is that of a dovecote housing pigeons in its holes. Now, whenever there are more pigeons than holes, at least two pigeons need to share a hole. This observation can also be generalized by stating that for an infinite amount of pigeons, at least one of the (finitely many) holes contains an infinite amount of pigeons. To make this more formal, we define the Infinite Pigeon Hole Principle as follows:

Theorem 4.3.1 (Infinite Pigeon Hole Principle)**.** Given a total function $f : \mathbb{N} \to H$, where H is finite, there exists a hole $h \in H$ s.t. the set of pigeons $P_h = \{p \in \mathbb{N} | f(p) = h\}$ in it is infinite.

Proof. For the sake of contradiction, assume P_h is finite for each hole $h \in H$. Then each hole contains a maximal element c_h for each $h \in H$. Because H is finite, the sum $\sum_{h \in H} c_h$ is also finite. Since all summands are natural numbers, $c_i \leqslant \sum_{h \in H} c_h$ for each $i \in H$. Furthermore $k = 1 + \sum_{h \in H} c_h$ is larger than any c_i and therefore not contained in any P_h with $h \in H$. But since f is a total function, k must be mapped to some $h \in H$ and hence also be contained in one of the P_h, leading to the contradiction. □

The restriction to only two holes can be proved in the same manner.

Theorem 4.3.2 (Infinite Pigeons, Two Holes Principle)**.** Given a total function $f : \mathbb{N} \to \{0, 1\}$, then either $P_0 = \{p \in \mathbb{N} | f(p) = 0\}$ or $P_1 = \{p \in \mathbb{N} | f(p) = 1\}$ is of infinite cardinality.

Proof. Again, let us assume P_0 and P_1 to be finite having c_0 and c_1 as their respective maximal elements. Then both $c_0 < c_0 + c_1 + 1$ and $c_1 < c_0 + c_1 + 1$ hold, therefore $c_0 + c_1 + 1$ is neither contained in P_0 nor P_1. But $f(c_0 + c_1 + 1) = 0$ or $f(c_0 + c_1 + 1) = 1$, meaning that it must be contained either in P_0 or P_1, leading to the contradiction. □

4.4 The n-occurrences Pigeon Hole Principle

Another metaphor for the pigeon hole principle is presented in the PhD thesis of Urban [108]. There, the function f from the previous section represents the tape of a Turing machine. The mapping $\mathbb{N} \to \{0, 1\}$ can then be seen as function from indices of tape cells to a binary alphabet.

Furthermore, the pigeon hole principle itself then states that at least one of the symbols occurs infinitely many times on the tape. An easy way to express this as a formula is to state that there is no upper bound on the occurrences of the symbol s: $\neg\exists x\forall y(x < y \rightarrow f(y) \neq s)$. Pushing the negation inwards, we can also state that for each position x, there is a larger position y containing s.

Definition 4.4.1 (Infinite occurrences)**.** The formula expressing an infinite number of occurrences of the symbol s on the tape f is defined as:

$$I(s) = \forall x\exists y(x < y \wedge f(y) = s)$$

Actually, Urban used the infinite pigeon hole principle for his investigations in (first-order) cut-elimination. It serves as a lemma to prove a simpler statement, namely that there exist two different positions on the tape which have the same symbol. Later on, several formalizations of this proof were used for experiments with the CERES method [4].

To obtain a harder problem, we can generalize the statement in two ways: firstly, we can state that for every natural number n, there exist n different positions on the tape containing the same symbol. Secondly, we focus our interest on extracting the series enumerating the n indices with the same symbol. Mathematically, we claim that there exists a strictly monotonic function h such that $f(h(0)) = f(h(i))$ holds for all $i \leqslant n$.

Theorem 4.4.2 (n-Occurrences Tape Enumeration)**.** For each natural number n, there exists a strictly monotonic function h s.t. $f(h(0)) = f(h(i))$ for all $i \leqslant n$.

Proof. From theorem 4.3.2 we know that at least one, $I(0)$ or $I(1)$ holds. We continue by handling each case separately:

- $I(0)$ holds:
  Now we perform induction on n:
  base case:
  By $I(0)$ we know there exists a position p for which $f(p) = 0$. Therefore we define $h(0) = p$.
  step case:
  By IH we have a function $h'(i)$ for which $f(h'(i)) = 0$ for all $i \leqslant n$. We now define $h(j)$ by a case distinction on j:
  - $j \leqslant n$: This is already covered by h' and we can define $h(j) = h'(j)$.
  - $j = n + 1$: By lemma $I(0)$ we know there exists an index y such that $y > h(n)$ and $f(y) = 0$. We then define $h(n + 1) = y$.

  This gives us $f(h'(j)) = 0$ for all $j \leqslant n + 1$.

- $I(1)$ holds: analogous to the 0 case

□

A stronger statement would be the claim that there exists a function h such that for every n we find at least n different indices in f with the same value. The fixed point of the induction in theorem 4.4.2 would provide such a function, but we can not express it directly. We follow Ratiu et. al. in the claim that this quantifier shift is not provable without the axiom of choice and formalize the unbounded sequence of enumeration functions for simplicity.

4.5 Formalization

One of the hardest steps of the CERES method is to find a resolution refutation of the characteristic sequent set. The reason is that the formalizer has little control over a theorem prover's search strategies. Therefore even in the first-order case, where the completeness of CERES guarantees the existence of a refutation, the input proof needs adjustments. For example, the relation $x \leqslant y$ can be rewritten as $x < y \vee x = y$ or $x < y + 1$. As a result, the formalization process becomes iterative, blurring the line between specification and analysis.

Further on, we will describe the axioms, definitions and structure common to all variants of the n-occurrences tape proof. Later on, we will split the proof into four versions, where version 2 and 3 can be fully analyzed. Version 3 is also fully contained in appendix A. Furthermore we will also replace the use of the induction axiom by iterated application of the induction step and prove specific instances of the n-Tape lemma. The proof versions 4 and 5 are results of these experiments.

4.6 Expectations

The formalization of the first version was dominated by the question on how to specify the function h. A simple way to count n different symbols is to look at $2n + 1$ arbitrary positions on the tape. Summing up the counts for each symbol, we can decide to return the positions of the one occurring at least n times. What poses a problem is that the symbol being counted can change.

In the configuration depicted in figure 4.2, we first look at a single position and find the symbol 0. Therefore the function $h(0) = 0$ is a witness for a monotonous function enumerating one symbol index. Advancing now to look at three positions, the symbol 0 is still in the majority and we just extend h by $h(1) = 1$. Now, when we look at two more positions, we find that there are insufficient occurrences of 0 to count. Our function becomes then $h(0) = 2, h(1) = 3, h(2) = 4$, counting the symbol 1 this time. Trying to count four symbols, we have to return to count occurrences of 0 again. We can extend the mapping from the case counting two occurrences by $h(2) = 5, h(3) = 6$.

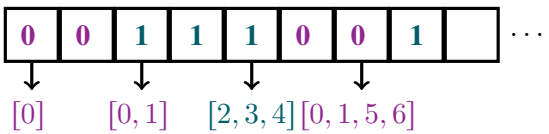


Figure 4.2: Example of a tape with switching counting function

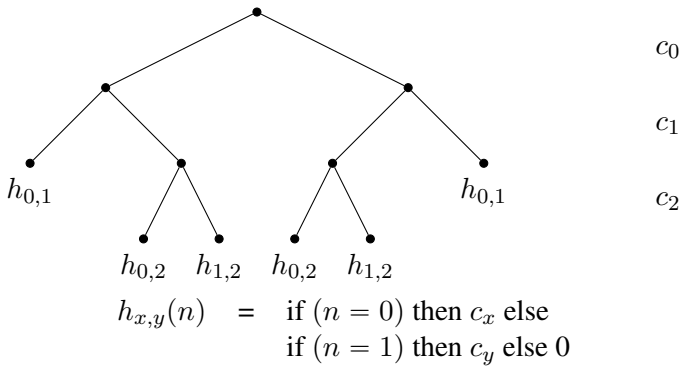


Figure 4.3: A possible decision tree for $h(2)$

To specify h, we could create a tree of nested conditionals expressing all possible configurations of $2n + 1$ positions. For instance, we can consider the tree for three positions (enumerating two symbols) depicted in figure 4.3. Each node at level k branches to the left if the tape at the corresponding position c_k contains 0, otherwise it branches to the right. The leaf nodes now are labeled with the functions returning the corresponding positions. Even though this solution looks satisfying, creating a lambda term which, given an encoded numeral n, expands to a decision tree with depth n of nested if-then-else statements is non-trivial. Our expectation now was that cut-elimination would obtain a function specification similar to this decision tree.

4.6.1 Axioms

The original idea was to use Robinson's Q with a second-order induction axiom. $LK_{skc=}$'s weak quantifier rules implicitly use comprehension, so we already have all the ingredients for higher-order arithmetic. In fact, for our investigations we stayed within second-order arithmetic. The highest occurring type of a quantifier variable is $\iota > \iota$, even though some skolem constants have a more complex type like ι>(ι>o)>(ι>ι)>ι. As it turned out, only using this axiom set is quite restrictive. Already proving the commutativity of addition would take a double induction – if commutativity is applied to a cut-ancestor, the characteristic sequent set will most likely include the proof of commutativity too[1]. If the induction invariant contains higher-order variables which also end up in the characteristic sequent set. Since the general induction axiom is cut-strong, a partial instantiation which still has higher-order variables may also be cut-strong[2]. For this reason, we added more arithmetic theorems with the restrictions that they are reasonably easy to prove in Peano Arithmetic.

From the three approaches for branching discussed in section 3.7.1, we picked the approach of arithmetic encoding. Since the actual reasoning is only used in the sub-proofs about evaluating if-then-else, it is easy to switch methods. Therefore we decided to encode only Boolean expressions and add the transformation to atom encodings when it is required.

The complete list of axioms is presented in table 4.1.

[1] The exception is the case where commutativity is not necessary to refute the characteristic sequent set.

[2] For sure, this is the case for predicate variables which occur directly in the formula structure.

Arithmetic:

Basic Axioms:

$A1$	$\neg(x+1=0)$
$A2$	$x=0 \vee (\exists k x=k+1)$
$A3$	$x+1=y+1 \rightarrow x=y$
$A4$	$x=y \rightarrow x+1=y+1$
$A5$	$0*x=0$
$A6$	$(x+1)*y=(x*y)+y$
$A7$	$\neg(x<0)$
$A8$	$x<y \rightarrow (\exists k(x+k)+1=y)$
$A9$	$x<y \vee x=y \vee y<x$
$A30$	$x+0=x$
$A31$	$0+x=x$

Derived Axioms (from A1 - A9, A30, IND) :

$A10$	$x<y+1 \rightarrow (x<y \vee x=y)$
$A11$	$x=y \vee x<y \rightarrow x<y+1$
$A12$	$x+y=y+x$
$A13$	$x+(y+z)=(x+y)+z$
$A14$	$x*y=y*x$
$A15$	$x*(y*z)=(x*y)*z$
$A16$	$x=y \rightarrow x+z=y+z$
$A17$	$x+z<y+z \rightarrow x<y$
$A36$	$x<y \rightarrow x+z<y+z$
$A18$	$x=0 \vee (\exists y y<x)$
$A19$	$0<x+1$
$A33$	$\neg(x<x)$
$A34$	$x<y \wedge y<z \rightarrow x<z$

Induction, bounded subtraction and encoding of truth values:

IND	$(\forall Y(0) \wedge (\forall n(Y(n) \rightarrow Y(n+1))) \rightarrow (\forall n Y(n)))$
$A20$	$(\forall x(0 \dot{-} x=0))$
$A21$	$(\forall x(x \dot{-} 0=x))$
$A22$	$(\forall x \forall y((x+1) \dot{-} (y+1)=x \dot{-} y))$
$A23$	$(\forall P(P \rightarrow \llcorner P \lrcorner=1))$
$A24$	$(\forall P(\neg P \rightarrow \llcorner P \lrcorner=0))$

Unused Axioms: A4, A11, A14, A15, A16, A18;

Table 4.1: Axioms of the n-occurrences Tape proof

4.6.2 Definitions

Already when we sketched the proof, we started using shortcuts to make the proof readable. Later on, we formally introduced the shortcuts as definition rules. The first formulas which underwent this treatment were the infinity lemma $I(s)$, the tape axiomatization T, monotonicity $MON(h,n)$ of a function restricted up to position n and $NOCC(h,n,s)$, which states that a function counts n occurrences of a specific symbol. What also came up quickly were the formulas C describing the actual formula we were going to prove and its relativization $A(\sigma)$, which removes the quantifier on the symbol σ from C and keeps it as a free variable instead.

For the presentation of the induction invariant, we also introduced $B(n,\sigma)$ by stripping A of the quantification on n[3].

The function definition t comes from proving the induction step. It reuses the function h

[3] $A(\sigma) = \forall n B(n,\sigma)$

$MON(h,n)$	$(\forall i \forall j(i < n+1 \wedge j < n+1 \wedge i < j \rightarrow h(i) < h(j)))$
$NOCC(h,n,s)$	$(\forall i(i < n+1 \rightarrow f(h(i)) = s))$
C	$(\forall n \exists h(MON(h,n) \wedge (\exists s NOCC(h,n,s))))$
$A(s)$	$\forall n \exists h(MON(h,n) \wedge NOCC(h,n,s))$
$B(n,s)$	$\exists h(MON(h,n) \wedge NOCC(h,n,s))$
T	$(\forall n(f(n) = 0 \vee f(n) = 1))$
$I(s)$	$(\forall x \exists y(x < y \wedge f(y) = s))$
$t(h,n,x,y)$	$if\ \llcorner x < n+1 \lrcorner\ then\ h(x)\ else\ y$
$\ulcorner i \urcorner$	$1 \dot{-} i$
$if\ \ulcorner i \urcorner\ then\ x\ else\ y$	$\ulcorner i \urcorner * x + (1 \dot{-} \ulcorner i \urcorner) * y$
$if\ \llcorner X \lrcorner\ then\ x\ else\ y$	$\llcorner X \lrcorner * x + (1 \dot{-} \llcorner X \lrcorner) * y$

Figure 4.4: Defined predicated and functions in the n-occurrences tape proof.

generated in the previous step if possible or picks the next suitable index on the tape.

Another set of definitions are related to the encoding of if-then-else. Even though it was not used in the proof, we prepared to allow expressions of type ι. For this, the condition expression must be mapped into the interval $\{0,1\}$ first. Actually, the definitions only need to be unfolded when proving properties of if-then-else, making the actual formalization transparent within the rest of the proof.

The full list of definitions is given in figure 4.4.

4.6.3 General Input Proof Structure

The proof is divided into the proof of the infinite occurrences lemma, which is then used to prove the main theorem. The respective names of these sub-proofs are $INFLEMMA$ and $MAIN$. Over the development of the proofs, these two components did not substantially change, but the top level proof which uses them did. The development can be seen in figures 4.5, 4.8 and 4.17. What is common in all top level proofs is the cut on the infinite occurrences axiom I, but in the beginning we used the additional lemma $A(0) \vee A(1)$ which turned out to be a major obstacle during the refutation phase.

The proof of the lemma is quite short (see figure 4.6), where the central element is exactly the one given in the mathematical formulation above: in the case distinction on the symbol, we instantiate the new position y as a sum $n_0 + n_1$. The use of commutativity allows the flipping of the roles of n_0 and n_1 during the reasoning on the second symbol. This allows us to introduce the preceding position x from n_0 in the first case and and from n_1 in the second. Since the roles of the variables flip in the different branches, both introductions of x fulfill the eigenvariable condition.

As it turned out, the main proof is independent of the actual symbol σ, which allowed some simplifications in version 3 of the proof. Its main ingredient is an induction on the invariant $B(n,\sigma)$, which unfolds to our claim that there exists a function h which is monotonous up to n and which counts n occurrences of σ. The base case for monotonicity is trivial. Similarly, the base case for one symbol occurrence directly follows from the infinity lemma. Unsurprisingly,

the step case encompasses the largest part of the proof. What was surprising though, was that showing the monotonicity of h is more effort than enumerating $n+1$ occurrences. The reason for that is that the monotonicity step $MONstep$ contains a case distinction on tape positions defined by the constraints $i < j$, $i < n+1$ and $j < n+1$. The n-occurrences step $NOCCstep$ extends the previous function by the case distinction covering the case of $n+1$. The exact statistics of the input proof sizes can be found in table 4.2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(BASE(\sigma))}{I(\sigma)\vdash B(0,\sigma)} \quad \dfrac{(STEP(\sigma))}{I(\sigma)\vdash \forall n(B(n,\sigma)\to B(n+1,\sigma))}}{I(\sigma)\vdash B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma))}\wedge:r \quad \dfrac{A(\sigma)\vdash A(\sigma)}{\forall nB(n,\sigma)\vdash A(\sigma)}\,def}{I(\sigma),(B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma)))\to\forall nB(n,\sigma)\vdash A(\sigma)}\to:l}{I(\sigma),\forall X(X(\sigma)\wedge\forall n(X(n)\to X(n+1))\to\forall nX(n))\vdash A(\sigma)}\forall:l}{I(\sigma),IND\vdash A(\sigma)}\,def}{(MAIN(\sigma))}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{(INFTAPE)}{AX,T\vdash I(0),I(1)} \quad \dfrac{(MAIN(0))}{AX,I(0)\vdash A(0)}}{AX,T\vdash A(0),I(1)}\,cut \quad \dfrac{(MAIN(1))}{AX,I(1)\vdash A(1)}}{AX,T\vdash A(0),A(1)}\,cut}{AX,T\vdash A(0)\vee A(1)}\vee:r \quad \dfrac{\dfrac{(TRIVIAL(0))}{A(0)\vdash C} \quad \dfrac{(TRIVIAL(1))}{A(1)\vdash C}}{A(0)\vee A(1)\vdash C}\vee:l}{AX,T\vdash C}\,cut$$
$$(TAPEPROOF)$$

Figure 4.5: General structure of version 1 of the n-occurrences tape proof

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(LTSUM(n_0,n_1))}{AX\vdash n_0<(n_0+n_1)+1} \quad \dfrac{\dfrac{(LTSUM(n_1,n_0))}{AX\vdash n_1<(n_1+n_0)+1}}{AX\vdash n_1<(n_0+n_1)+1}\,EQAX:A12 \quad \dfrac{\dfrac{\dfrac{f((n_0+n_1)+1)=0\vdash f((n_0+n_1)+1)=0 \quad f((n_0+n_1)+1)=1\vdash f((n_0+n_1)+1)=1}{f((n_0+n_1)+1)=0\vee f((n_0+n_1)+1)=1\vdash f((n_0+n_1)+1)=1,f((n_0+n_1)+1)=0}\vee:l}{(\forall x(f(x)=0\vee f(x)=1))\vdash f((n_0+n_1)+1)=1,f((n_0+n_1)+1)=0}\forall:l}{T\vdash f((n_0+n_1)+1)=1,f((n_0+n_1)+1)=0}\,def}{AX,T\vdash f((n_0+n_1)+1)=1,(n_1<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=0}\wedge:r}{AX,T\vdash(n_0<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=1,(n_1<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=0}\wedge:r}{AX,T\vdash(n_0<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=1,(\exists y((n_1<y)\wedge f(y)=0))}\exists:r}{AX,T\vdash(\exists y((n_0<y)\wedge f(y)=1)),(\exists y((n_1<y)\wedge f(y)=0))}\exists:r}{AX,T\vdash(\exists y((n_0<y)\wedge f(y)=1)),(\forall x\exists y((x<y)\wedge f(y)=0))}\forall:r}{AX,T\vdash(\forall x\exists y((x<y)\wedge f(y)=1)),(\forall x\exists y((x<y)\wedge f(y)=0))}\forall:r}{AX,T\vdash(\forall x\exists y((x<y)\wedge f(y)=1)),I(0)}\,def}{AX,T\vdash I(1),I(0)}\,def$$
$$(INFTAPE)$$

Figure 4.6: The infinite occurrences lemma (without sub-proof LTSUM)

Version independent			
BASE	INFTAPE	MONstep	NOCCstep
169	207	857	456

Version specific			
	TAPEPROOF v2	TAPEPROOF v3	TAPEPROOF v4 (Instance 2 / 3 / 4)
Input proof	3262	1738	3324 / 3342 / 3360
Preprocessed input	3655	1947	3749 / 3803 / 3857
Atomic-cut normal form	5174	5429	–
Characteristic sequent set	14	12	1034 / 1034 / 1034
Preprocessed css	9	7	87 / 87 / 87
Refutation size (dag)	76	52	–
Refutation size (tree)	133	122	–
Reproved deep formula (dag)	1118	1110	–
Reproved deep formula (tree)	1492173	3345509	–

Table 4.2: Statistics of input proof sizes

4.7 Version 1

The first formalization phase happened in parallel to the specification of the LLK language. The first patterns to emerge were axiom instantiation, reasoning modulo a theory equation and lemma instantiation. This allowed to encapsulate the lemmas about if-then-else and other simple arithmetical statements like $\forall x \neg(x + 1 < x)$. The first resulting sequent set left so much room for improvement, that we started to improve the input. It turned out that even though sparsely used, the lemma instantiations substantially increased the sequent set size. Luckily, the inferences could be replaced by direct proof instantiations. The extracted sequent set still had the impressive size of 243154 elements taking about 20 minutes to extract. To eliminate some redundancy, we tried to shrink the sequent set using subsumption, but stopped the algorithm after one week running. To overcome this, we folded the subsumption algorithm into the sequent set generation (see section 3.8).

What finally made a significant difference was to relax the projection computation, leaving cuts on formulas without quantifiers or free higher-order variables intact. After applying subsumption, the final sequent set now had 29 elements.

Next, we could turn our attention to the sequents themselves. The preprocessing step of expanding tautological formula axioms (see section 2.6.3) was responsible that the sequents were already in clause form. Some expressions within a clause were still hard to understand. Especially two skolem contexts, which come from the instantiation of the induction axiom, were obstructing the structure of a clause by their sheer size. After introducing the constants q_1 and q_0 for them, we could obtain a human readable clause set (see figure 4.7).

What was still problematic, though, was that the set contained a high number of the function variables $h0$ and $h1$. Nonetheless we attempted to find a refutation via System on a TPTP using Leo II, Satallax and TPS as backends. Within the time limit of 5 minutes, we could not find a refutation. Leaving a local installation of Leo II running for a week did not yield a result either.

Since also manual attempts on finding instantiations failed, we went back to redesigning the input proof.

$\vdash ((0 + n1) < ((n0 + 1) + n1))$;
$\vdash ((n0 + n1) = (n1 + n0))$;
$\vdash ((n1 + (n0 + 1)) = ((n1 + n0) + 1))$;
$\vdash ((0 + n1) = n1)$;
$\vdash (s_{24}(h0) < (s_{27} + 1)), (s_{26}(h0, 0) < (s_{27} + 1))$;
$\vdash (s_{25}(h0) < s_{24}(h0)), (s_{26}(h0, 0) < (s_{27} + 1))$;
$\vdash (s_{25}(h0) < (s_{27} + 1)), (s_{26}(h0, 0) < (s_{27} + 1))$;
$\vdash (s_{28}(h1) < (s_{31} + 1)), (s_{30}(h1, 1) < (s_{31} + 1))$;
$\vdash (s_{29}(h1) < s_{28}(h1)), (s_{30}(h1, 1) < (s_{31} + 1))$;
$\vdash (s_{29}(h1) < (s_{31} + 1)), (s_{30}(h1, 1) < (s_{31} + 1))$;
$\vdash (f(((n1 + n0) + 1)) = 0), (f(((n1 + n0) + 1)) = 1)$;
$(h0(s_{25}(h0)) < h0(s_{24}(h0))) \vdash (s_{26}(h0, 0) < (s_{27} + 1))$;
$(h1(s_{29}(h1)) < h1(s_{28}(h1))) \vdash (s_{30}(h1, 1) < (s_{31} + 1))$;
$(f(h0(s_{26}(h0, 0))) = 0) \vdash (s_{24}(h0) < (s_{27} + 1))$;
$(f(h0(s_{26}(h0, 0))) = 0) \vdash (s_{25}(h0) < s_{24}(h0))$;
$(f(h0(s_{26}(h0, 0))) = 0) \vdash (s_{25}(h0) < (s_{27} + 1))$;
$(f(h1(s_{30}(h1, 1))) = 1) \vdash (s_{28}(h1) < (s_{31} + 1))$;
$(f(h1(s_{30}(h1, 1))) = 1) \vdash (s_{29}(h1) < s_{28}(h1))$;
$(f(h1(s_{30}(h1, 1))) = 1) \vdash (s_{29}(h1) < (s_{31} + 1))$;
$(h0(s_{25}(h0)) < h0(s_{24}(h0))), (f(h0(s_{26}(h0, 0))) = 0) \vdash$;
$(h1(s_{29}(h1)) < h1(s_{28}(h1))), (f(h1(s_{30}(h1, 1))) = 1) \vdash$;
$(f(\alpha 3) = 1), (s_{21}(q_1, s_{22}(q_1)) < \alpha 5), (f(\alpha 5) = 1), (i0 < (n0 + 1)) \vdash (n0 < (n0 + 1)), (f(s_{23}(q_1, n0, i0)) = 1)$;
$(f(\alpha 0) = 0), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0), (i0 < (n0 + 1)) \vdash (n0 < (n0 + 1)), (f(s_{11}(q_2, n0, i0)) = 0)$;
$(f(\alpha 0) = 0), (s_{10}(q_2) < (s_{10}(q_2) + 1)), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0), (i0 < (n0 + 1)) \vdash (f(s_{11}(q_2, n0, i0)) = 0)$;
$(f(\alpha 3) = 1), (s_{22}(q_1) < (s_{22}(q_1) + 1)), (s_{21}(q_1, s_{22}(q_1)) < \alpha 5), (f(\alpha 5) = 1), (i0 < (n0 + 1)) \vdash (f(s_{23}(q_1, n0, i0)) = 1)$;
$(f(\alpha 3) = 1), (s_{21}(q_1, s_{22}(q_1)) < \alpha 5), (f(\alpha 5) = 1), (i0 < (n0 + 1)), (j1 < (n0 + 1)), (i0 < j1) \vdash (n0 < (n0 + 1)), (s_{23}(q_1, n0, i0) < s_{23}(q_1, n0, j1))$;
$(f(\alpha 0) = 0), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0), (i0 < (n0 + 1)), (j1 < (n0 + 1)), (i0 < j1) \vdash (n0 < (n0 + 1)), (s_{11}(q_2, n0, i0) < s_{11}(q_2, n0, j1))$;
$(f(\alpha 0) = 0), (s_{10}(q_2) < (s_{10}(q_2) + 1)), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0), (i0 < (n0 + 1)), (j1 < (n0 + 1)), (i0 < j1) \vdash (s_{11}(q_2, n0, i0) < s_{11}(q_2, n0, j1))$;
$(f(\alpha 3) = 1), (s_{22}(q_1) < (s_{22}(q_1) + 1)), (s_{21}(q_1, s_{22}(q_1)) < \alpha 5), (f(\alpha 5) = 1), (i0 < (n0 + 1)), (j1 < (n0 + 1)), (i0 < j1) \vdash (s_{23}(q_1, n0, i0) < s_{23}(q_1, n0, j1))$;

with: $q_1 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 1)))))$
$q_2 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 0)))))$

Figure 4.7: Characteristic sequent set of version 1 of the n-occurrences tape proof

4.8 Version 2

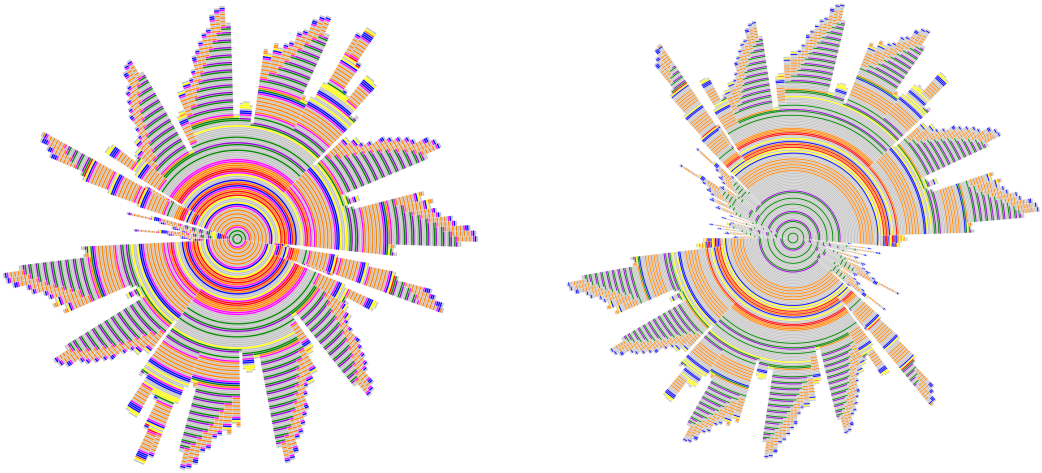
Going back to the proof, we analyzed the remaining cut formulas. The most complex formula was $A(0) \vee A(1)$, one of the top level cuts. Even though Gentzen's reductive elimination method can not move a cut inference over an equational inference or another cut, it could be used in this case. The cut was used to prove $A(\sigma) \vdash C$ for both cases, which could be folded into the $MAIN$ proof such that it proves $AX, T, I(0) \vdash C$ directly instead of $A(\sigma)$(see figure 4.8). The sunburst view shows the complete structure of the proof (see figure 4.9 a).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(BASE(\sigma))}{I(\sigma)\vdash B(0,\sigma)} \quad \dfrac{(STEP(\sigma))}{I(\sigma)\vdash \forall n(B(n,\sigma)\to B(n+1,\sigma))}}{I(\sigma)\vdash B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma))}\wedge:r \quad \dfrac{\dfrac{(TRIVIAL(\sigma))}{A(\sigma)\vdash C}}{\forall nB(n,\sigma)\vdash C}\,def}{I(\sigma),(B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma)))\to\forall nB(n,\sigma)\vdash C}\to:l}{I(\sigma),\forall X(X(\sigma)\wedge\forall n(X(n)\to X(n+1))\to\forall nX(n))\vdash C}\forall:l}{I(\sigma),IND\vdash C}\,def$$

$(MAIN(\sigma))$

$$\dfrac{\dfrac{\dfrac{(INFTAPE)}{T\vdash I(0),I(1)} \quad \dfrac{(MAIN(0))}{I(0)\vdash C}}{T\vdash I(1),C}\,cut \quad \dfrac{(MAIN(1))}{I(1)\vdash C}}{\dfrac{T\vdash C,C}{T\vdash C}\,c:r}\,cut$$

Figure 4.8: General structure of the second version of the n-occurrences tape proof



a) Input Proof b) Atomic-cut normal form

Figure 4.9: Graphical comparison of version 2 of the n-occurrences tape proof before and after CERES

4.8.1 Characteristic Sequent Set

The benefit of this elimination was dramatic: the characteristic sequent set shrank to 9 clauses (see figure 4.10). It still contained higher-order skolem terms, but after introducing the constants q_0 and q_1 for the skolem contexts of the induction rule, there was a good chance to find a refutation. Indeed, Leo II managed to refute the clause set in less than four seconds by embedding the problem into a first-order one. We also managed to directly embed the clause set into first-order (see section 3.9) and load a Prover 9 refutation into GAPT.

This allowed us to transform the input proof into atomic-cut normal form and extract an

$\vdash ((0 + n0) < ((n1 + 1) + n0))$;
$\vdash ((n0 + (n1 + 1)) = ((n0 + n1) + 1))$;
$(f(\alpha 0) = 0), (s_{10}(q_2) < (s_{10}(q_2) + 1)), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0) \vdash$;
$\vdash (f(((n1 + n0) + 1)) = 0), (f(((n1 + n0) + 1)) = 1)$;
$(f(\alpha 0) = 0), (s_9(q_2, s_{10}(q_2)) < \alpha 2), (f(\alpha 2) = 0) \vdash (n0 < (n0 + 1))$;
$(f(\alpha 3) = 1), (s_{26}(q_1) < (s_{26}(q_1) + 1)), (s_{25}(q_1, s_{26}(q_1)) < \alpha 5), (f(\alpha 5) = 1) \vdash$;
$(f(\alpha 3) = 1), (s_{25}(q_1, s_{26}(q_1)) < \alpha 5), (f(\alpha 5) = 1) \vdash (n0 < (n0 + 1))$;
$\vdash ((n0 + n1) = (n1 + n0))$;
$\vdash ((0 + n0) = n0)$;
with
$q_1 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 1)))))$
$q_2 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 0)))))$

Figure 4.10: Characteristic sequent set of version 2 of the n-occurrences tape proof

expansion sequent. From the expansion sequent, we retrieved the instantiation terms for the enumeration function h (see figure 4.11).

Source		*Term*
$TRIVIAL(0)$		$s_{14}(q_1, s_{15})$
$TRIVIAL(0)$		$s_{30}(q_2, s_{15})$
$BASE(0)$	$\lambda x($	$(s_{25}(q_1, s_{26}(q_1)) + 1) + s_9(q_2, s_{10}(q_2)))$
$BASE(1)$	$\lambda x($	$(s_9(q_2, s_{10}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1)))$
$STEP(0)$	$\lambda x($	$\lrcorner x < (s_{10}(q_2) + 1)\lrcorner * s_9(q_2, x)) +$
		$((1 \dot{-} \lrcorner x < (s_{10}(q_2) + 1)\lrcorner) * ((s_{25}(q_1, s_{26}(q_1)) + 1) + s_9(q_2, s_{10}(q_2)))))$
$STEP(1)$	$\lambda x($	$\lrcorner x < (s_{26}(q_1) + 1)\lrcorner * s_{25}(q_1, x2)) +$
		$((1 \dot{-} \lrcorner x2 < (s_{26}(q_1) + 1)\lrcorner) * ((s_9(q_2, s_{10}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1)))))$

$q_1 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 1)))))$
$q_2 = \lambda x(\exists h((\forall i(\forall j(((i < (x + 1)) \wedge ((j < (x + 1)) \wedge (i < j))) \rightarrow (h(i) < h(j))))) \wedge (\forall i((i < (x + 1)) \rightarrow (f(h(i)) = 0)))))$

Figure 4.11: Function instantiation terms in the second version of the n-occurrences tape proof

4.8.2 Analysis

As an orientation, we can visually compare the original proof to its atomic-cut normal form in the sunburst view (see figure 4.9): the dual reasoning on both tape symbols leads to a symmetry which is also present after cut-elimination. The simulation of the resolution refutation is easily recognized by the additional green cut-nodes, purple equational nodes and grey contraction nodes at the center of the tree.

To gain more information, we analyze the extracted expansion proof. Expanding the weak quantifier node for the function h within the occurrence of the formula C in the end-sequent, we find the two instances $s_{14}(q_1, s_{15})$, $s_{30}(q_2, s_{15})$. Both of them are skolem terms which give no insight into the structure of h. Investigating the proof, we quickly find the reason: the instance comes from the sub-proof $TRIVIAL$, which just shows the introduction of the existential quantification over the symbol σ within $A(\sigma)$ such that it becomes C. Since the quantification of h

and n in C are strong occurrences, it introduces the constants s_{14}, s_{30} and s_{15}. So even though $A(\sigma)$ contains a weak quantifier on h, it does not help at all.

We now turn our attention to the two occurrences of the induction axiom in $MAIN(0)$ and $MAIN(1)$, where the invariant $B(n, \sigma)$ contains the other quantifications on h. In figure 4.12, the occurrences of the invariant which contain a weak quantification over h are underlined. One of them is the base case, the other is the $n + 1$ part of the step case.

$$(\underline{B(0,\sigma)} \wedge \forall \mathbf{n}(\overline{B(\mathbf{n},\sigma)} \rightarrow \underline{B(\mathbf{n}+1,\sigma)})) \rightarrow \forall n\, B(n,\sigma)$$

Figure 4.12: Sources of function instances in the induction axiom of the n-occurrences tape proof

If we look into these two instances of h, we notice the same skolem constants occurring all over again: s_9 and s_{25} of type $(\iota > o) > \iota > \iota$ and s_{10} and s_{26} of type $(\iota > o) > \iota$. Apart from finding patterns in the term structure, we need to give a semantic interpretation of these skolem symbols.

Remembering our definitions, the pattern in the term structure is easier to recognize: it is an instance of the if-then-else definition. Reintroducing the definition, the terms look already more readable (see figure 4.13).

Source		*Term*
$BASE(0)$	$h(x) =$	$(s_{25}(q_1, s_{26}(q_1)) + 1) + s_9(q_2, s_{10}(q_2))$
$BASE(1)$	$h(x) =$	$(s_9(q_2, s_{10}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1))$
$STEP(0)$	$h(x) =$	$if\ \llcorner x < (s_{10}(q_2) + 1) \lrcorner\ then\ s_9(q_2, x)\ else\ (s_{25}(q_1, s_{26}(q_1)) + 1) + s_9(q_2, s_{10}(q_2))$
$STEP(1)$	$h(x) =$	$if\ \llcorner x < (s_{26}(q_1) + 1) \lrcorner\ then\ s_{25}(q_1, x)\ else\ (s_9(q_2, s_{10}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1))$

Figure 4.13: Induction instances in the expansion sequent of version 2 of the n-occurrences tape proof

What we can see now is that the base cases are identical modulo commutativity. Likewise the else branches of the step cases are equal modulo associativity and commutativity. This looks mysterious, since we would expect the base case of the function enumerating 0 and of the one enumerating 1 to be different. The solution must lie in the interpretation of the skolem symbols, so we continue with this task.

An idea on how to interpret them comes from the position in the proof where they are introduced. The constants s_9 and s_{10} occur in the $MAIN(0)$ branch of the input proof, whereas the constants s_{25} and s_{26} occur in the $MAIN(1)$ branch, which explains the duplicate occurrence of each type. Investigating further, we find out that s_9 and s_{25} come from the strong occurrence of h in the induction step, which is overlined in figure 4.12.

The source of s_{10} and s_{26} is nearby, in form of the induction variable n, which is written in bold script in the same figure. What is still a bit confusing is the context of the skolem symbols. All of them share the first parameter, which comes from instantiating the invariant within the induction axiom. This also explains why the term q_2 only occurs as an argument of s_9 and s_{10}

whereas q_1 is only an argument of s_{25} and s_{26}. The second parameter is the induction variable n, up to which the function h is already defined in the step.

Since we have a base instance and a step instance of h, we could try to write down an inductive definition of h.

If we look into the base cases and into the "else" branch of the step case, the same term occurs there (modulo associativity and commutativity): $c + d + 1$ with $c = s_9(q_2, s_{10}(q_2))$ and $d = s_{25}(q_1, s_{26}(q_1))$. This looks exactly like the instantiation terms in the lemma (see 4.6) – including the application of commutativity to flip the roles of c and d, depending which symbol we are counting.

In the step case, the situation is similar, but the "then" branch contains a term for which c is an instance in $STEP(0)$ and d is an instance in $STEP(1)$. The reason is that in $STEP(0)$, s_9 is the function from our induction hypothesis which is defined up to $s_{10}(q_2)$. Then c is simply the last index defined for s_9 to which we add $d+1$ to obtain the next index of the tape containing 0.

The situation for $STEP(1)$ is mirrored, but now d is the the function $s_{25}(q_1, x)$ evaluated at the highest defined index $s_{26}(q_1)$.

What makes these explanations complicated, is that we only look at one step where the preceding ones are captured by the skolem functions. Unfolding the induction, we define a series of indices $c_{i+1} + d_{i+1} + 1$ and a series of pairs of functions $h_{0,i+1}$ and $h_{1,i+1}$ where $h_{0,i+1}(i + 1) = c_{i+1} = c_i + d_i + 1$ and $h_{1,i+1}(i + 1) = d_{i+1} = c_i + d_i + 1$. The base case for both functions is both $c_0 + d_0 + 1$. In a way, in $STEP(0)$, the c_i enumerate the indices on the tape where the d_i express the distance of the gaps between c_i and c_{i+1}. For $STEP(1)$, the two just swap roles where d_i represents indices and c_i represents holes. If we put this into proper mathematical notation, we arrive at the function in figure 4.14. A diagram showing the gaps can be seen in figure 4.15.

$$
\begin{array}{ll}
h_{0,0}(0) & c_0 + d_0 + 1 \\
h_{0,n+1}(x) & \begin{cases} h_{0,n}(x) & \text{if } x < n + 1 \\ h_{0,n}(n) + d_{n+1} + 1 & o.w. \end{cases} \\
h_{1,0}(0) & c_0 + d_0 + 1 \\
h_{1,n+1}(x) & \begin{cases} h_{1,n}(x) & \text{if } x < n + 1 \\ h_{1,n}(n) + c_{n+1} + 1 & o.w. \end{cases}
\end{array}
$$

Figure 4.14: Mathematical version of the function extracted from version 2 of the n-occurrences tape proof

Now, the classical nature of the proof comes into play. We defined two witness functions, but only one of them needs to provide the correct indices on the tape. Moreover, we are still away from a computational interpretation of the proof, since the exact position is hidden by the skolem functions. A way to salvage something is to manually provide a possible interpretation of h_0 and h_1, based on the analysis done so far. For the function not used for recursion will just express the distance. To distinguish it from the function counting indices, we call it g and define $g_1(i) = h_0(i + 1) - h_0(i) - 1$ while counting zeroes and $g_0(i) = h_1(i + 1) - h_1(i) - 1$ while counting ones.

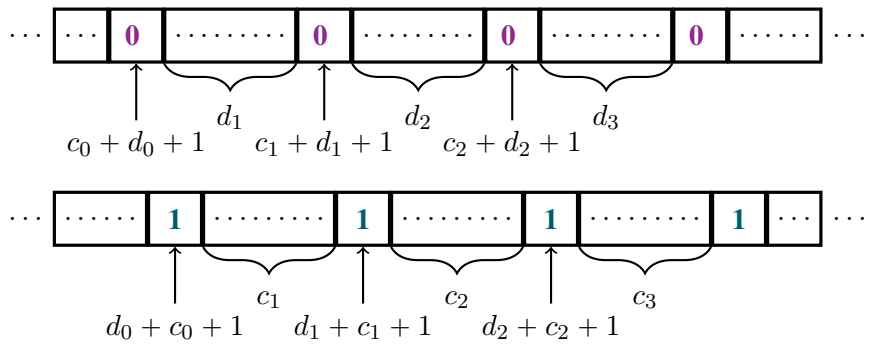


Figure 4.15: Interpretation of the skolem functions on the tape

Now only the implementation of the actual function is missing. Since our lemma proves the existence of an index, we can enumerate indices until we have found a fitting one. Borrowing the μ operator from recursion theory, we can fill in the gaps and arrive at the definition in figure 4.16.

Case 0:

$$h_{0,0}(0) \quad g_1(0) + 1$$

$$h_{0,n+1}(x) \quad \begin{cases} h_{0,n}(x) & \text{if } x \leqslant n \\ h_{0,n}(n) + g_1(n) + 1 & \text{if } x > n \end{cases}$$

$$g_1(n) \quad \begin{cases} (\mu y(0 < y \wedge f(y) = 0)) - 1 & \text{if } n = 0 \\ (\mu y(h_{0,n}(n) < y \wedge f(y) = 0)) - h_{0,n}(n) - 1 & \text{if } n > 0 \end{cases}$$

Case 1:

$$h_{1,0}(0) \quad g_0(0) + 1$$

$$h_{1,n+1}(x) \quad \begin{cases} h_{1,n}(x) & \text{if } x \leqslant n \\ h_{1,n}(n) + g_0(n) + 1 & \text{if } x > n \end{cases}$$

$$g_0(n) \quad \begin{cases} (\mu y(0 < y \wedge f(y) = 1)) - 1 & \text{if } n = 0 \\ (\mu y(h_{1,n}(n) < y \wedge f(y) = 1)) - h_{1,n}(n) - 1 & \text{if } n > 0 \end{cases}$$

Figure 4.16: Operational interpretation of the extracted function from version 2 of the n-occurrences tape proof

4.8.3 Running the Experiments in GAPT

Each of the versions has its own Scala object `nTape2` – `nTape4` providing those steps in figure 4.1 which apply. Since we managed to complete the analysis of version 2, all the steps are available.

The following script is also available in the gapt distribution in the file `examples/ntape/ntape2.script`. It calls the appropriate methods of the `nTape2` object, which again collect various cli commands. The source code with the API calls is contained in appendix E, a help text with the API documentation can be generated using the `help` command.

```
gapt> help(nTape2)
```

In the end, it also prints the information used to create tables 4.2 and 4.11.

```
gapt> //Step 1: proof database with proofs, definitions and completed
gapt> //    input proof
gapt> nTape2.proofdb.Definitions

gapt> prooftool( nTape2.input_proof )

gapt> //Step 2: prepocessed input proof
gapt> nTape2.preprocessed_input_proof

gapt> //Step 3: conversion to LKskc=
gapt> nTape2.lksk_proof

gapt> //Step 4: extract characteristic sequent set and projections
gapt> prooftool( nTape2.css )

gapt> nTape2.projections

gapt> //Step 5: prepreocess the css applying subsumption,
gapt> //    replacing lambda abstractions by fresh
gapt> //    function terms and mapping hol types to fol
gapt> nTape2.fol_css

gapt> //Step 6: refute fol clause set
gapt> prooftool( nTape2.fol_refutation )

gapt> //Step 7: convert fol refutation to Ral=, reintroducing
gapt> //    lambda expressions and hol types
gapt> nTape2.ral_refutation

gapt> //Step 8: create proof in atomic-cut normal-form by
gapt> //    simulating the refutation
gapt> prooftool( nTape2.acnf )

gapt> //Step 9: extract expansion proof (datastructure includes atomic cuts in
gapt>       antecedent ) and show expansion proof without atomic cuts
gapt> nTape2.expansion_proof

gapt> prooftool( nTape2.expansion_proof.expansionSequent )

gapt> // Print statistics and extracted witness terms
gapt> nTape2.printStatistics
```

The integration of Leo II is not that tight, since GAPT is still unable to parse its output. Therefore we export the problem to the TPTP THF fragment.

```
gapt> nTape2.export_thf("ntape2leo.tptp")
```

A manual call to Leo II shows it is able to find a proof quickly.

```
[marty@phedre gapt]$ time leo ntape2leo.tptp -po 0 -t 600
real  0m3.515s
user  0m3.495s
sys   0m0.020s
```

4.9 Version 3

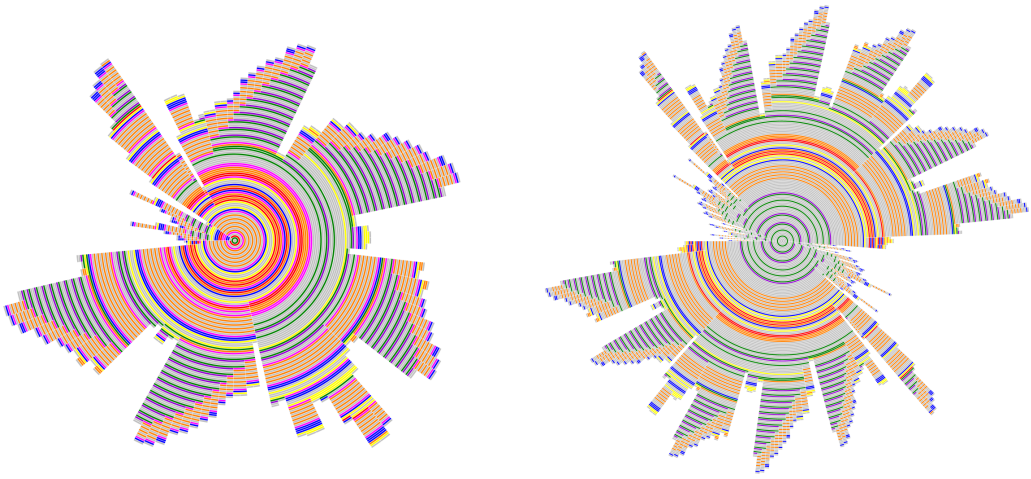
After the successful analysis of version 2, we saw a simple way to improve the input. Instead of instantiating $MAIN$ with each symbol, we could infer $\exists s I(s)$ from both instances and contract them. Then, only one cut on $\exists s I(s)$ is necessary, which nearly halves the input proof size. An immediate effect of this is that the induction axiom is only used for a label once. Instead of two instances the skolem context now contains a more general label with the free variable σ. As

a consequence, the terms q_1 and q_2 in the first-order clause set vanish and are replaced by the function $q(\sigma)$, which also shrinks to the clause set to 7 elements (see figure 4.19).

What does not change too much is the structure of the atomic-cut normal form of the proof. The reason is that during the resolution refutation, $q(\sigma)$ is instantiated as $q(0)$ and $q(1)$. The two instances play exactly the same role the terms q_1 and q_2 do in version 2 of the proof. In fact, $q(0)$ is identical to q_1 and $q(1)$ is identical to q_2 after unfolding the definitions of q, q_1 and q_2. Equipped with this knowledge, it is no surprise that the extracted function terms are also the same as that of version 2 (see figure 4.20).

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(INFTAPE)}{AX, T \vdash I(0), I(1)}}{AX, T \vdash \exists s I(s), I(1)}\exists : r}{AX, T \vdash \exists s I(s), \exists s I(s)}\exists : r}{AX, T \vdash \exists s I(s)}c : r \qquad \dfrac{(MAIN(\sigma))}{AX, I(\sigma) \vdash C}}{\dfrac{AX, \exists s I(s) \vdash C}{AX, AX, T \vdash C}\exists : l}cut$$
$$\dfrac{AX, AX, T \vdash C}{AX, T \vdash C}c : l$$
$$(TAPEPROOF)$$

Figure 4.17: General structure of the third version of the n-occurrences tape proof



a) Input Proof b) Atomic-cut normal form

Figure 4.18: Graphical comparison of version 3 of the n-occurrences tape proof before and after CERES

$\vdash ((0 + n0) < ((n1 + 1) + n0))$;
$\vdash ((n0 + (n1 + 1)) = ((n0 + n1) + 1))$;
$\vdash (f(((n1 + n0) + 1)) = 0), (f(((n1 + n0) + 1)) = 1)$;
$(f(\alpha 0) = \sigma 0), (s_9(q(\sigma 0), s_{10}(q(\sigma 0))) < \alpha 2), (f(\alpha 2) = \sigma 0) \vdash (n0 < (n0 + 1))$;
$(f(\alpha 0) = \sigma 0), (s_{10}(q(\sigma 0)) < (s_{10}(q(\sigma 0)) + 1)), (s_9(q(\sigma 0), s_{10}(q(\sigma 0))) < \alpha 2), (f(\alpha 2) = \sigma 0) \vdash$;
$\vdash ((n0 + n1) = (n1 + n0))$;
$\vdash ((0 + n0) = n0)$;

Figure 4.19: Characteristic sequent set of version 3 of the n-occurrences tape proof

$\lambda x((s_9(q_1, s_{10}(q_1)) + 1) + s_9(q_2, s_{10}(q_2)))$
$\lambda x((s_{25}(q_2, s_{26}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1)))$
$\lambda x((\llcorner x < (s_{10}(q_2) + 1)\lrcorner * s_9(q_2, x)) + ((1 \dot{-} \llcorner x < (s_{10}(q_2) + 1)\lrcorner) * ((s_9(q_1, s_{10}(q_1)) + 1) + s_9(q_2, s_{10}(q_2)))))$
$\lambda x((\llcorner x < (s_{26}(q_1) + 1)\lrcorner * s_{25}(q_1, x)) + ((1 \dot{-} \llcorner x < (s_{26}(q_1) + 1)\lrcorner) * ((s_{25}(q_2, s_{26}(q_2)) + 1) + s_{25}(q_1, s_{26}(q_1)))))$

Figure 4.20: Function instances of version 3 of the n-occurrences tape proof

4.9.1 Analysis

To understand the difference between versions 2 and 3 of the n-occurrences proof we might have a look at the shifting rule for quantifiers when performing reductive cut-elimination. Figure 4.21 shows the pattern which occurs in version 3. Shifting the cut over the contraction duplicates the cut rule, leading to the pattern from version 2. In any case, the eigenvariables α and β are subsequently replaced by the terms s and t.

When we come back to the CERES method, we can compare the characteristic sequent sets of these transformations. To simplify matters, we assume here that F is an atom and was introduced by an axiom rule (i.e. not by a weakening rule). Then the characteristic sequent set of π_1 contains at least one element of the form $\mathcal{C} \times \vdash F(s)$. If the sets introducing $F(s)$ and $F(t)$ were joined by a binary rule – which in case of the n-occurrences tape proof is the clause $\vdash f(x) = 0, f(x) = 1$ – it will be of the form $\mathcal{C}' \times \vdash F(s), F(t)$. Similarly, the characteristic sequent set of $\pi(\alpha)$ contains a clause $\mathcal{D} \times F(\alpha) \vdash$. Resolving twice with the clause above leads to the clause $\mathcal{C}' \times \mathcal{D}\{\alpha \leftarrow s\} \times \mathcal{D}\{\alpha \leftarrow t\}$. This means that either way, using an LK proof transformation or resolution, the instantiation and duplication happens.

Certainly, the resolvent clause does not need to be used for the actual refutation, but there is some intuition why it often happens in practice. Since the characteristic sequent set characterizes the cut formulas, a clause will not be used in the refutation, whenever the cut-formula it contributes to is redundant. Since we were rearranging the main lemma of the n-occurrences tape proof, it is highly unlikely it is redundant. Therefore we just moved some effort from the human input to the theorem prover.

One interesting observation is that in our particular case, Prover 9 still finds a refutation in less than a second, while Leo II returns an SZS status Unknown after a timeout of ten minutes.

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, F(s), F(t)}}{\Gamma_1 \vdash \Delta_1, \exists x F(x), F(t)}\exists : r}{\Gamma_1 \vdash \Delta_1, \exists x F(x), \exists x F(x)}\exists : r}{\Gamma_1 \vdash \Delta_1, \exists x F(x)}\, c : r \qquad \dfrac{(\pi_2(\alpha))}{\Gamma_2, F(\alpha) \vdash \Delta_2}\exists : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, F(s), F(t)}}{\Gamma_1 \vdash \Delta_1, \exists x F(x), F(t)}\exists : r}{\Gamma_1 \vdash \Delta_1, \exists x F(x), \exists x F(x)}\exists : r \qquad \dfrac{(\pi_2(\alpha))}{\Gamma_2, F(\alpha) \vdash \Delta_2}\exists : l}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, \exists x F(x)}\, cut \qquad \dfrac{(\pi_2(\beta))}{\Gamma_2, F(\beta) \vdash \Delta_2}\exists : l}{\dfrac{\Gamma_1, \Gamma_2, \Gamma_2 \vdash \Delta_1, \Delta_2, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, c : *}\, cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, F(s), F(t)}}{\Gamma_1 \vdash \Delta_1, \exists x F(x), F(t)}\exists : r \qquad \dfrac{(\pi_2(t))}{\Gamma_2, F(t) \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, \exists x F(x)}\, cut \qquad \dfrac{(\pi_2(\beta))}{\Gamma_2, F(\beta) \vdash \Delta_2}\exists : l}{\dfrac{\Gamma_1, \Gamma_2, \Gamma_2 \vdash \Delta_1, \Delta_2, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, c : *}\, cut$$

$$\Downarrow$$

$$\dfrac{\dfrac{\dfrac{(\pi_1)}{\Gamma_1 \vdash \Delta_1, F(s), F(t)} \qquad \dfrac{(\pi_2(t))}{\Gamma_2, F(t) \vdash \Delta_2}}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, F(s)}\, cut \qquad \dfrac{(\pi_2(s))}{\Gamma_2, F(s) \vdash \Delta_2}}{\dfrac{\Gamma_1, \Gamma_2, \Gamma_2 \vdash \Delta_1, \Delta_2, \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}\, c : *}\, cut$$

Figure 4.21: Reductive cut-elimination of a contracted existential quantifier

4.9.2 Running the Experiments in GAPT

Since version 2 and 3 of the n-occurrences proof are so similar, all the commands mentioned in section 4.8.3 still apply. The full script is also available in the gapt distribution in the file `examples/ntape/ntape3.script`. Again, the `printStatistics` methods outputs the information used to create tables 4.2 and 4.20.

```
gapt> // Print statistics and extracted witness terms
gapt> nTape3.printStatistics
```

4.10 Version 4

Since the induction axiom hides the unfolding of the enumeration function h and we found no possibility to encode an unfolding of the if-then-else structure up to a variable n, we aimed for a lower goal. How would the instantiation terms look like, if we only proved our theorem for a specific instance? First of all, the main formula C needed to be replaced by a relativization $D(n) \equiv \exists h(MON(h,n) \wedge \exists \sigma NOCC(h,n,\sigma))$. Now, for the $MAIN(\sigma)$ proof, instead of using the proofs $BASE(\sigma)$ and $STEP(\sigma)$ to prove the induction axiom, we cut on the conjunction of $B(0,\sigma)$ and $\forall n(B(n,\sigma) \rightarrow B(n+1,\sigma))$. Then we could use n contractions of the induction step to show $B((\ldots(0+1)\ldots+1),\sigma)$. Below the proof of $MAIN(\sigma)$, nothing changed and we could use the same reasoning as in the n-occurrences proof version 2.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{B(0,\sigma)\vdash B(0,\sigma)}\quad \dfrac{\dfrac{\dfrac{}{B(0+1,\sigma)\vdash B(0+1,\sigma)}\quad \dfrac{}{B((0+1)+1,\sigma))\vdash D((0+1)+1)}}{B(0+1,\sigma),B(0+1,\sigma)\to B((0+1)+1,\sigma))\vdash D((0+1)+1)}\to:l}{B(0+1,\sigma),\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}\forall:l}{B(0,\sigma),B(0,\sigma)\to B(0+1,\sigma),\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}\to:l}{B(0,\sigma),\forall n(B(n,\sigma)\to B(n+1,\sigma)),\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}\forall:l}{B(0,\sigma),\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}c:l$$

$(TRIVIAL)$

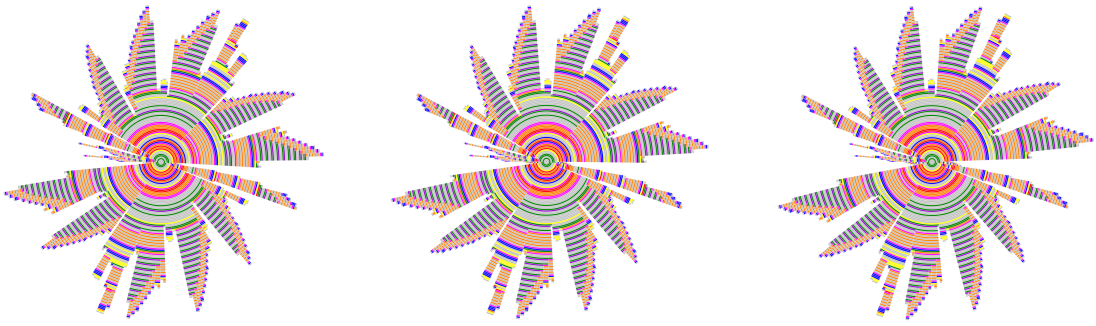
$(STEPS)$

$$\dfrac{\dfrac{\dfrac{}{I(\sigma)\vdash B(0,\sigma)}(BASE(\sigma))\quad \dfrac{}{I(\sigma)\vdash\forall n(B(n,\sigma)\to B(n+1,\sigma))}(STEP(\sigma))}{I(\sigma)\vdash B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma))}\wedge:r\quad \dfrac{\dfrac{}{B(0,\sigma),\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}(STEPS)}{B(0,\sigma)\wedge\forall n(B(n,\sigma)\to B(n+1,\sigma))\vdash D((0+1)+1)}\wedge:l}{I(\sigma)\vdash D((0+1)+1)}cut$$

$(MAIN(\sigma))$

$$\dfrac{\dfrac{\dfrac{}{T\vdash I(0),I(1)}(INFTAPE)\quad \dfrac{}{I(0)\vdash D((0+1)+1)}(MAIN(0))}{T\vdash I(1),D((0+1)+1)}cut\quad \dfrac{}{I(1)\vdash D((0+1)+1)}(MAIN(1))}{T\vdash D((0+1)+1),D((0+1)+1)}cut$$
$$\dfrac{T\vdash D((0+1)+1),D((0+1)+1)}{T\vdash D((0+1)+1)}c:r$$

Figure 4.22: Structure of the instance $(0+1)+1$ of the fourth version of the n-occurrences tape proof



a) $(0+1)+1$ b) $((0+1)+1)+1$ c) $(((0+1)+1)+1)+1$

Figure 4.23: Graphical comparison of instances of version 4 of the n-occurrences tape proof

4.10.1 Analysis

We investigated the instances numbers 2, 3 and 4 for our analysis, which correspond to the upper bound n of the enumeration function h instantiated with the terms $0 + 1$, $(0 + 1) + 1$ and $((0 + 1) + 1) + 1$. Instance 1 only uses the induction base and does not contain a higher-order cut.

Interestingly, in each case, the clause set set size was 1034. After applying subsumption these sets together with tautology elimination, we ended up with 75 clauses in both cases. Comparing the clause sets of the three instances, we observed that the only difference in the reduced clause sets was the instance constant.

Unfortunately, similar to version 1 of the proof, the formula of the induction step here is a cut ancestor, resulting in free function variables in the clause set. An example for such a clause is $f(h(s_{35}(h, 1))) = 1 \vdash s_{33}(h) < ((0 + 1) + 1)$ – in total 34 clauses contain a function variable. Again, the recursion pattern is clearly visible, but no higher-order prover managed to find a refutation of this clause set. Similarly, we failed at finding an instantiation to first-order by hand.

4.10.2 Running the Experiments in GAPT

Since we did not manage to perform the full analysis of version 4, we can only execute the steps up to the export of the characteristic sequent set to the TPTP THF fragment. Again, the `printStatistics` method outputs the information used in table 4.2.

```
gapt> // Print statistics and extracted witness terms
gapt> for (i <- 2 to 4) nTape4(i).printStatistics
```

We can investigate the clauses with function variables in prooftool:

```
gapt> prooftool( nTape4(2).preprocessed_css_hol_clauses )
```

The actual THF export is done calling `export_thf` again.

```
gapt> for (i <- 2 to 4) nTape4(i).export_thf(s"ntape4-thf-${i}.tptp")
```

4.11 Version 5

The reason why the sequent set was the same in version 4 can be explained by the additional cut on the induction step. The cut-formula is quantified on n, therefore the clause set is also general in n, even though we only need specific instances. The next version of the proof therefore used explicit instances of the induction step instead of the general one (see figure 4.24 for the modified proof structure). This meant, that instead of one cut, we now needed to introduce $n + 1$ cuts, to prove the instance for the numeral n. Using the optimization done in version 3, we now obtained 55 clauses for the instance $(0 + 1)$.

Since the theorem provers still failed, we continued with the simplification of the proof. In the original proof, removing arithmetization the if-then-else did not make a significant difference. With the additional cut on the induction step, this changed. Replacing the arithmetic proof of if-then-else by the equational axiomatization, the sequent set shrunk to 214 elements, which could be reduced to 26. As expected, the size of the sequent set started to grow with the size of

$$\dfrac{\dfrac{(BASE(\sigma))}{AX, I(\sigma) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))} \quad \dfrac{\dfrac{(STEP'(0,\sigma))}{AX, I(\sigma), (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma))) \vdash (\exists h(MON(h,0+1) \wedge NOCC(h,0+1,\sigma)))}}{AX, I(\sigma), AX, I(\sigma) \vdash (\exists h(MON(h,0+1) \wedge NOCC(h,0+1,\sigma)))}\, c:l}{AX, I(\sigma) \vdash (\exists h(MON(h,0+1) \wedge NOCC(h,0+1,\sigma)))}\, cut \quad \dfrac{(TRIVIAL(0+1))}{(\exists h(MON(h,0+1) \wedge NOCC(h,0+1,\sigma))) \vdash D(0+1)}\, cut$$

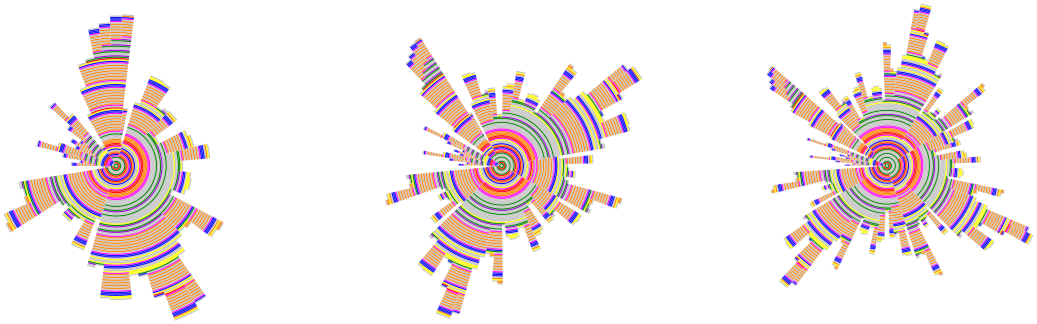
$$\dfrac{AX, I(\sigma) \vdash D(0+1)}{(MAIN(\sigma))}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{(INFTAPE)}{T \vdash I(0), I(1)}\, \exists:r}{AX, T \vdash (\exists x I(x)), I(1)}\, \exists:r}{AX, T \vdash (\exists x I(x)), (\exists x I(x))}\, c:r}{AX, T \vdash (\exists x I(x))} \quad \dfrac{(MAIN(\sigma))}{AX, I(\sigma) \vdash D(0+1)}\, \exists:l}{AX, (\exists x I(x)) \vdash D(0+1)}\, cut$$

$$\dfrac{AX, AX, T \vdash D(0+1)}{AX, T \vdash D(0+1)}\, c:l$$

$$(TAPEPROOF)$$

Figure 4.24: Structure of the instance $(0+1)+1$ of the fifth version of the n-occurrences tape proof



a) $(0+1)+1$ b) $((0+1)+1)+1$ c) $(((0+1)+1)+1)+1$

Figure 4.25: Graphical comparison of instances of version 5 of the n-occurrences tape proof

the instantiation numeral. Still, in all cases, no higher-order theorem prover managed to refute our problem. Figure 4.26 contains the full statistics for instances 2 to 4.

	Arithmetic if-then-else	Axiomatized if-then-else		
	Instance 2	Instance 2	Instance 3	Instance 4
Input proof	1760	900	1383	1866
Preprocessed input	1967	1004	1528	2052
Characteristic sequent set	522	71	122	173
Preprocessed css	55	26	30	54
Clauses with HOL content	18	15	25	35

Figure 4.26: Proof and characteristic sequent set sizes for version 5 of the n-occurrences tape proof

4.11.1 Running the Experiments in GAPT

Similar to version 4, we can only export the generated sequent sets to TPTP THF. Again, the `printStatistics` method outputs the information used in table 4.2 and `help(nTape5)` provides help and the API of the experiment.

```
gapt> // Print statistics and extracted witness terms
gapt> for (i <- 2 to 4) nTape5(i).printStatistics
```

We can investigate the clauses with function variables in prooftool:

```
gapt> prooftool( nTape5(2).preprocessed_css_hol_clauses )
```

The actual THF export is done calling `export_thf` again.

```
gapt> for (i <- 2 to 4) nTape5(i).export_thf(s"ntape5-thf-${i}.tptp")
```

4.12 Experiments with the If-then-else axiomatization

The failure to derive instances of the instantiated n-Tape problem sets leads to the question for the source. To eliminate other influences, we simplify the problem and claim the existence of a function which is specified by a finite amount of function values. Again we are looking at variations of the same problem, all of which are contained in figure 4.27.

To start simply, we represent an enumeration of the sequence $(1, 0)$ as $\exists h(h(0) = s(0) \wedge h(s(0)) = 0$ and try to prove it from our axiomatization of if-then-else. Out of Leo II, Satallax and agsyHOL, only the latter manages to prove the theorem. The initial expectation was that the prover finds an instantiation similar to $h(x) = if\ x = 0\ then\ s(0)\ else\ 0$. Accordingly, also Leo II and Satallax find a proof when we add this witness to our axiom set.

Extending the sequence to $(1, 0, 0)$ and $(1, 0, 1)$ we are in a similar situation: the witness can be described with the use of a single if-then-else conditional. Nonetheless, it is necessary to add axioms which ascertain that the domain does not collapse to one element. Without this requirement, conditions like $s(0) \neq 0$ allowing to evaluate the else-branch can not be proved for the problems $p1$ and $p2$, which have three approximation points. Just to be sure the axiomatization would work for nested if-then-else constructs, we provided a second witness W_3 for problem $p2$, which Leo II and Satallax accepted without problems.

In a way the bad performance of the theorem provers is no surprise: the condition X in the if-then-else axiomatization is a free variable. The clauses $X \rightarrow if\ (\ then\ X\ else\ , s, t) =$

F_1 $\forall X \forall u \forall v (X \to ite(X, u, v) = u)$
F_2 $\forall X \forall u \forall v (\neg X \to ite(X, u, v) = v)$
F_3 $\forall x (s(x) \neq 0)$
F_4 $\forall x (s(x) \neq x)$
W_1 $\forall x (h(x) = ite((x = 0), s(0), 0))$
W_2 $\forall x (h(x) = ite((x = s(0)), 0, s(0)))$
W_3 $\forall x (h(x) = ite((x = 0), s(0), ite((x = s(0)), 0, s(0))))$
G_1 $\exists H (H(0) = s(0) \wedge H(s(0) = 0))$
G_2 $\exists H (H(0) = s(0) \wedge H(s(0)) = 0 \wedge H(s(s(0))) = 0))$
G_3 $\exists H (H(0) = s(0) \wedge H(s(0)) = 0 \wedge H(s(s(0))) = s(0)))$

Short	Name	Conjecture		Provable by Leo II	Satallax	agsyHOL
p0	(1,0) sequence	F_1, F_2	$\vdash G_1$			x
w0	(1,0) sequence + witness W_1	F_1, F_2, W_1	$\vdash G_1$	x	x	x
	(1,0,0) sequence	F_1, F_2	$\vdash G_2$			
p1	(1,0,0) sequence + arith	F_1, F_2, F_3, F_4	$\vdash G_2$			
	(1,0,0) sequence + witness W_1	F_1, F_2, W_1	$\vdash G_1$			
w1	(1,0,0) sequence + witness W_1 + arith	F_1, F_2, F_3, W_1	$\vdash G_2$	x	x	x
	(1,0,1) sequence	F_1, F_2	$\vdash G_3$			
p2	(1,0,1) sequence + arith	F_1, F_2, F_3, F_4	$\vdash G_3$			
w2	(1,0,1) sequence + witness W_2 + arith	F_1, F_2, F_3, F_4, W_2	$\vdash G_3$	x	x	x
w2b	(1,0,1) sequence + witness W_3 + arith	F_1, F_2, F_3, F_4, W_2	$\vdash G_3$	x	x	x

Figure 4.27: Variants of if-then-else experiments

s and $\neg X \to if\ (\ then\ X\ else\ , s, t) = t$ directly encode a cut on X such that unification with arbitrary negative/positive literals will happen. Moreover, it seems that equality is usually handled in a special way such that the unification with $x = 0$ never happens. This problem even remains when we instantiate the if-then-else axiomatization to $x = y \to if\ (\ then\ x\ else\ = y, s, t) = s$. An obvious solution would be to use the TPTP $\$ite$ construct for conditionals (see the definition of thf_conditional in the TPTP Syntax BNF [99]), but unfortunately it is unsupported by Leo II and Satallax.

4.12.1 Running the Experiments in GAPT

This time, we only generate clause sets and export them to TPTP THF. As usual, a call to `help(nTape6)` opens the API documentation with a short description. The method `export` has an optional parameter for a directory, where the problem set is written - the default is the current directory. The files themselves are called `ntape-6-${i}-without-witness.tptp` and `ntape-6-${i}-with-witness.tptp` for i from 0 to 2. Within the system, the methods `p0` to `p2` and `w0` to `w2` provide the corresponding problems.

```
gapt> nTape6.P2
gapt> nTape6.W2
gapt> nTape6.export()
```

The call to Leo II and Satallax is done as usual, with a 10 minute timeout:

```
for I in ntape6-*tptp; do leo $I -t 600 > $I.leo ;done
for I in ntape6-*tptp; do satallax -t 600 $I> $I.satallax ;done
```

The generated input and output files can also be downloaded from the personal webpage of the author[4]. An even better download option is the NUN section on the website of the TPTP problem library[5]. The problems NUN24, NUN25 and NUN26 correspond to the three functions defined here.

4.13 Comparison to A-Translation with modified Realizability

4.13.1 The method

The infinite pigeonhole principle was also formalized by Ratiu and Trifonov [79]. In this case, the computational information was extracted twice. The first method uses a combination of refined versions of A-Translation and modified Realizability [17,38] and a second time by functional interpretation [42,90]. Since in comparison to our analysis with cut-elimination, the two methods have similar results, we focus on A-translation, which performed better for the investigated proof of the infinite pigeon hole principle.

The context of both methods is Negative Arithmetic (NA^ω) which is the fragment of Heyting Arithmetic with finite types (HA^ω) which only uses implication and universal quantification in its formula language. The language of object terms is Gödel's System $\mathcal{T}$ which is a lambda calculus with the simple types of booleans B, natural numbers N and lists L of natural numbers. Complex types are constructed via type arrows and a pairing operator. It also adds the respective type constructors (tt, ff, 0, S, $[]$, $::$, $<,>$) as well as pair projections, a conditional operator $\mathcal{C}$ and the operators $\mathcal{R}_N, \mathcal{R}_L$ for primitive recursion on naturals and lists. The latter are defined as additional rewrite rules for beta-reduction.

$$
\begin{array}{ll}
\mathcal{R}_N\ 0\ s\ t \quad \to s & \mathcal{R}_L\ nil\ s\ t \quad \to s \\
\mathcal{R}_N\ (S0)\ s\ t \to t\ n\ (\mathcal{R}_N\ s\ t) & \mathcal{R}_L\ (n :: l)\ s\ t \to t\ n\ l\ (\mathcal{R}_L\ l\ s\ t) \\
\mathcal{C}\ tt\ t_1\ t_2 \quad \to t_1 & \\
\mathcal{C}\ ff\ t_1\ t_2 \quad \to t_2 &
\end{array}
$$

Atoms are created by lifting decidable boolean terms to atoms via the $at(\,)$ predicate.[6] Alternatively, terms are applied to predicate variables to form uninterpreted atoms. Formulas are inductively created using the $\bot$, $\to$ and $\forall$ operators. The classical negation operator $\tilde{\neg}A$ is defined as $A \to \bot$, the classical existential quantifier $\tilde{\exists} x A$ is defined as the dual $\tilde{\neg}\forall x \tilde{\neg} A$ of universal quantification. The pattern $A_1 \tilde{\wedge} \ldots \tilde{\wedge} A_n \to B$ stands for a series of implications $A_1 \to \ldots A_n \to B$.

The language of proof terms annotates each term with the conclusion of the corresponding natural deduction rules. Its axiom rules allow the introduction of an assumption variable u^A and of arithmetical truth $at(tt)$. The introduction rules of implication and universal quantification are

[4] https://logic.at/staff/riener/if-then-else.tgz

[5] http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Problems&Domain=NUN

[6] Even though $at(\,)$ lifts the type B to the formula level, boolean terms are not formulas themselves.

realized via lambda abstraction with an eigenvariable condition for the latter. The elimination rules are realized by application. Additionally it contains (schematic) induction rules on the base types B, N and L.

The role of modified A-Translation is to rewrite a classically valid formula such that modified realizability can be applied. It applies Gödel's negative translation which amounts to double negation of atoms in NA^ω. Further simplifications are achieved by distinguishing computationally relevant formulas (those ending in $\bot$) from irrelevant ones. Using this notion, goals are atoms including $\bot$, implications proving another goal from a relevant or decidable assumption and universal quantification over irrelevant goals. Furthermore, definite formulas are atoms including $\bot$, implications with a relevant assumption, implications with an irrelevant conclusion and universal quantification over definite formulas.

Formulas to be realized have the form $\forall x(\overline{D} \to \overline{H} \to \tilde{\exists}\overline{G})$, with $\overline{D}$ atoms, $\overline{G}$ goals and $\overline{H}$ arbitrary. There, arithmetical falsity ($at(ff)$) replaces logical falsity ($\bot$) in goals and definite formulas. Denoting this substitution A^F for a formula A, the remaining occurrences of $\bot$ are replaced by $\exists y\overline{G}^F$. The formula in consideration then has the form $\forall x(\overline{D}^F \to \overline{H}[\bot \leftarrow \exists y\overline{G}(x,y)^F] \to \tilde{\exists}\overline{G}^F)$. As required by realizability, it belongs to the Π^0_2 fragment of the arithmetical hierarchy.

We now turn to modified realizability: the notion $|A|^r$ means that r realizes the formula A. To every predicate variable P of type $\sigma_1 > (\ldots(\sigma_n > o))$ we assign a corresponding type α_P and a fresh variable P^r of type $\alpha_P > (\sigma_1 > (\ldots(\sigma_n > o)))$. Then we can define realizability as follows:

$$
\begin{array}{ll}
|P(\overline{s})|^r & = \begin{cases} P^r(r,s) & \text{if } P \text{ is a predicate variable of type } \alpha_P \\ P(s) & \text{if } P \text{ is a predicate constant} \end{cases} \\
|\forall x A|^r & = \forall x |A|^{rx} \\
|A \to B|^r & = \forall x(|A|^x \to |B|^{rx})
\end{array}
$$

When extracting a realizer from a proof term, the logical rules are directly interpreted in the term language. The induction rules map to primitive recursion, whereas the conditional rules map to arithmetical choice with the condition of $n = 0$ for naturals and $l = []$ for lists.

$$
\begin{array}{ll}
[\![u^A]\!] & x_u^{\tau(A)} \\
[\![(\lambda u^A M)^{A\to B}]\!] & \lambda x_u^{\tau(A)} [\![M]\!] \\
[\![(M^{A\to B} N)]\!] & [\![M]\!][\![N]\!] \\
[\![(\lambda x^\rho M)^{\forall x\, A}]\!] & \lambda x^\rho [\![M]\!] \\
[\![(M^{\forall x\, A} r)]\!] & [\![M]\!]\mathrm{r}
\end{array}
\qquad
\begin{array}{ll}
[\![Ind_{N,A}]\!] & \mathcal{R}_N \\
[\![Ind_{L,A}]\!] & \mathcal{R}_{L(\rho)} \\
[\![Cases_{n,A}]\!] & \mathcal{C}_{n,\sigma} \\
[\![Cases_{l,A}]\!] & \mathcal{C}_{l,\sigma}
\end{array}
$$

The relation between a the proof term for a formula and its realizer is made explicit the following soundness theorem:

Theorem 4.13.1 (Soundness of Proof Extraction by A-Translation with modified Realizability [17])**.**
Assume that M is a derivation of B. Then there is a derivation $|B|^{[\![M]\!]}$ from the assumptions $\{|C|^{x_u^{\tau(C)}} | u^C \in \text{ free assumptions of } M\}$.

In other words, if M is a proof of the formula B, there exists a proof of B realized by the term extracted from M which uses the same assumptions as M.

4.13.2 Formulation of the Infinite Pigeonhole Principle

The formulation of the infinite pigeonhole principle IPH is quite close to the INFTAPE lemma used in this work:

$$\forall r \forall f(\forall n f(n) < r \to \tilde{\exists} q \forall n \tilde{\exists} k(n \leqslant k \tilde{\wedge} f(k) = q)) \qquad \text{IPH}$$

$$\forall x(f(x) = 0 \vee f(x) = 1) \to \exists q \forall x \exists y(x < y \wedge f(y) = q) \qquad \text{INFTAPE}$$

The function f represents an infinite sequence of r different colors. The sequence directly corresponds to the tape f but we investigate only the instance $r = S(S(0))$. The theorem IPH does not fulfill the syntactic requirements for realizability, therefore it is used to prove the consequence UPH which does.

$$\begin{array}{ll}
\forall r \forall f(\forall n \tilde{\neg} \tilde{\neg} f(n) < r \to \forall n \tilde{\exists} l G(f,n,l)) & \text{UPH} \\
|l| = n \tilde{\wedge} & \\
\quad (\forall m(S(m) < n \to l_{S(m)} < l_m)) \tilde{\wedge} & \\
\quad (\forall m(S(m) < n \to f(l_m) = f(l_{S(m)}))) & G'(f,n,l) \\
 & \\
\forall x(f(x) = 0 \vee f(x) = 1) \to \forall n \exists h G(h,n) & \text{TAPEPROOF} \\
MON(h,n) \wedge \exists s \forall i(i < n+1 \to f(h(i)) = s)) & \text{G(h,n)} \\
\forall i \forall j(i < n+1 \wedge (j < n+1 \wedge i < j) \to h(i) < h(j)) & \text{MON(h,n)}
\end{array}$$

The modified goal of UPH claims the existence of a descending list l of length n which contains the indices of the same color. Again, there are strong similarities to our TAPEPROOF theorem. The enumeration function h plays the same role as the list l. Both have a monotonicity requirement with the TAPEPROOF version being the transitive closure of the one in UPH. Moreover, the n-occurrences statement in the TAPEPROOF differs from the same colors statement in UPH only by making the color explicit.

4.13.3 Proof and Extracted Programs

The proof of IPH is an induction on r, where the base case is trivial because the condition $\forall n f(n) < 0$ is a contradiction. The step case is proved indirectly by assuming the induction hypothesis ($\forall r \forall f(\forall n f(n) < r \to \tilde{\exists} q \forall n \tilde{\exists} k(n \leqslant k \tilde{\wedge} f(k) = q))$) and the restriction to $r+1$ colors ($\forall n f(n) < S(r)$) hold, but the conclusion of the step is false ($\forall q \tilde{\neg}(\forall n \tilde{\exists} k(n \leqslant k \tilde{\wedge} f(k) = q))$). Now either the color r appears an infinite amount of times, then this leads straight to contradiction with the negated conclusion. Alternatively, the color r appears a finite amount of times. Then there exists an index i, from which on f does not contain the color r. We then create a function $f'(n) = f(max(n,i))$ which repeats the symbol $f(i)$ in the range 0 to i and agrees with f for indices greater than i. But the induction hypothesis also applies to f' from which we conclude that a symbol of color less than r repeats an infinite amount of times which again leads to a contradiction.

The proof of UPH uses IPH as an assumption and proves the existence of a list l of decreasing indices such that $f(l_m) = q$ for all $m < n$ and the infinitely occurring color q, again by induction

on the length n of the list. For length 0, the witness is the empty list, for length 1 the instance $n = 0$ of IPH suffices. For the induction step IPH is instantiated with $n = l_0$, the highest index of the list provided by the induction hypothesis. Also since for all elements in the list $f(l_i) = q$, q is replaced by $f(l_{i+1})$ which removes q, including the existential quantifier on q.

The extracted programs conform to continuation passing style, a technqiue in functional programming which takes a program - the continuation - as an additional parameter which is executed instead of returning control to the caller. This is mirrored in the weakening of tautologies of the form $A \to \bot$ by replacing $\bot$ by arbitrary formulas. The pattern also surfaces when the higher-order functions associated to IPH are passed functions associated with UPH. In particular, UPH provides an implementation for the check that the coloring is finite (FC) and the function to construct the infinite monochromatic series (IMS). The sequence extraction function SE ties the knot of the continuation and accumulates the actual implementation of the function h in $TAPEPROOF$.

IPH:

$$
\begin{array}{ll}
IPH(r, f, FC, IMS) & := if\ r = 0\ then\ FC(0, \square)\ else\ IMS(r-1, IS) \\
IS(n, SE) & := IPH(r-1, \lambda x.f(max(n,x)), FC_n, \lambda x \lambda y.IMS_n(SE, x, y)) \\
FC_n(n', \bot) & := FC(max(n,n'), if\ f(max(n,n')) \neq r-1\ then \bot\ else\ SE(max(n,n')) \\
IMS_n(SE, q, IS') & := IMS(q, IS'(k, \lambda x.SE(max(n,x))))
\end{array}
$$

UPH:

$$
\begin{array}{lcl}
UPH(r, f, n) & := & IPH(r, f, FC, IMS) \\
FC(n, \bot) & := & if\ f(n) < r\ then \bot\ else\ \square \\
IMS(q, IS) & := & FS(n, \lambda l.l) \\
FS(n, P) & := & if\ n = 0\ then\ nil\ else \\
 & & if\ n = 1\ then\ IS(0, \lambda k.P(k :: nil))\ else\ FS(n-1, P') \\
P'(k :: l) & := & IS(k+1, \lambda k'.P(k' :: k :: l))
\end{array}
$$

Figure 4.28: Realizers for IPH and UPH obtained by A-Translation (slightly modified to accomodate for additional arguments from helper functions)

The minlog system [18, 89] implements both the A-Translation with modified realizability and the Dialectica method. The example file `pigeonhole.scm` in the directory `minlog examples/classical/combinatorics` of the distribution contains the formalizations discussed here. Executing the test successfully extracts lists of indices for all possible input configurations of two symbols and tapes of length four.

4.14 Comparison

As mentioned above, the n-Tape lemma INFTAPE is essentially the IPH lemma restricted to two symbols. The statement of the theorem TAPEPROOF differs from UPH in two ways: TAPEPROOF does not require a double negation translation and it uses functions of type $N > N$

to represent the sequence instead of the list data-type L. The refined A-Translation does not permit existential quantification on a function as a goal, whereas the CERES$^\omega$ method does not have such restrictions. But this freedom comes with a price: data-types are essential to the successful proof extraction from UPH, because System $\mathcal{T}$ contains recursion operators to interpret induction rules for each data-type. If the induction axiom is used instead of a rule, the invariant $\exists h G(h, n)$ must be skolemized in the induction step $\exists h G(h, n) \rightarrow \exists h G(h, n + 1)$. But then the interpretation of a skolem function is only axiomatized by the context in which it appears. Even worse, the term language might be insufficient to express the skolem function. This is exactly the case for s_9 and s_{25} in the n-Tape 2 formalization of TAPEPROOF, where we added μ-recursion to express the interpretations.

Another important lesson is to reduce the expressivity of the condition in the if-then-else operator like it is done with $\mathcal{C}$. There only arithmetical expressions are allowed, not arbitrary formulas like in the n-Tape formulation. The effect only surfaces in the instantiated example n-Tape 6, where the cut-strong axiomatization of if-then-else ended up in the characteristic sequent set and which subsequently sent each automated higher-order prover into a hopeless synthetisation attempt for that conditional.

4.15 Discussion

Looking at every version of the n-occurrences tape proof, we identified three vital elements for extracting a meaningful function from the proof. First, disregarding purely propositional cut-formulas without higher-order variables reduced the clause set to a treatable size. The second ingredient was a successful embedding of the problem into a first-order clause set. The third key was the manual interpretation of the skolem symbols. Unfortunately, this part cannot be fully automatized, but it might be worth to study the effect of specific axioms. In this particular case, where we encounter a combination of the induction axiom with a weak function quantifier in the invariant, the idea to interpret the skolem function as a fixed point is close by. Nonetheless, some conditions have to be met to allow this interpretation. A straightforward one is the fact that the lemma folded into the function is a Π^0_2 formula. Only because a Π^0_2 formula can be seen as a function itself, the fixed-point interpretation is meaningful.

Similar to the observation in the introduction that empirically, the clause set of first-order CERES can be solved by any first-order prover in System on TPTP or not at all, the sequent sets of CERES$^\omega$ seem to be solvable by higher-order theorem provers when they can be mapped and solved by first-order prover or not at all. In particular the recursion pattern reflected by term $h(s(h))$ with a skolem function s selecting the next input of h based on itself that occurred in early sequent sets seems to be out of reach in principle. The prover would need to syntesize possibly nested conditionals, which is futile without built-in support for if-then-else in the prover. To the best of our knowledge, no higher-order prover supports the `$ite` construct of TPTP.

Furthermore, we overestimated the performance of higher-order provers in the fragment with first-order equality. In the case of Leo II, the integration of E prover seems handle most equality reasoning. But then it is clear that this leads to a two phase process where the higher-order instantiations are found first and only then a most of the the equational part is solved. In the case of Satallax, the SAT backend is completely unaware of equality. We suspect that this is the

reason why Satallax does not find a proof of the essentially multisorted first-order deep formulas of `nTape2` and `nTape3`. Interestingly, the only prover which can reprove the deep formulas is Isabelle 2016 by encoding the problem into SMT and solving it with Z3.

5.1 Summary

In this thesis we introduced $\mathrm{CERES}^{\omega}_{=}$ which adds first-order equality to the calculi used in CERES^{ω} but removes quantifier inferences from the resolution calculus, effectively removing the need for labels in there. Furthermore we could replace the computationally expensive post-processing steps of the full CERES^{ω} method and extract expansion proofs directly from the PCNF. This result carries over $\mathrm{CERES}^{\omega}_{=}$ because its PCNF can be translated to a version with equality axioms instead of rules.

We also extended the GAPT framework by an implementation of the new calculus, including support for Prover9 proofs, the LLK proof input language, and interactive viewers for tree-like proofs and expansion sequents. In the meantime, the Ebner, Hetzl, Reis, Wolfsteiner and Zivota added, amongst other things, support for cut-introduction, an internal superposition prover, expansion trees with cut and backends for a multitude of theorem provers including minlog, veriT, E Prover, SPASS and Vampire [36].

Consequently, we managed to formalize a proof of the infinite pigeon hole principle. In our variant, the proof proceeds using the finite pigeon hole principle as a lemma, which is subsequently eliminated by $\mathrm{CERES}^{\omega}_{=}$. The key to finding the required refutation of the characteristic sequent set was the replacement of cut-free projections by ones which may include cuts on propositional formulas. This allows the higher-order theorem prover Leo II to successfully embed the problem into first-order logic and solve it with eProver. Additionally our manual first-order translation could be refuted by Prover 9.

As we hoped, expansion proofs allowed us to investigate the instantiation terms of the function quantifier we were interested in. Inside, we found two inductively defined functions, covering the cases of counting an infinite occurrence of each possible symbol. In contrast to the competing methods, the presence of skolem symbols makes the interpretation of the term as program impossible. A manual analysis of the proof interprets the relevant skolem terms as fixed points of the formula representing the finite pigeon hole principle. This interpretation is ad-hoc, but can be justified by the proof structure. Nonetheless, since the Skolem functions are now only declaratively specified, they destroy the interpretation of the function term as a program. In programming terms, this would amount to a call to a function for which we know pre- and postconditions but where no actual implementation is given. A term with these holes can not be executable. Our particular case is even worse because the interpretation of the skolem function is a fixed point computation, which is not realizable in simply typed lambda calculus.

Since we conjectured the fixed point construction to be unavoidable, we turned our attention

to instances of the problem by finitely repeating the induction step. Contrary to our intuition, the problem became harder since where we had a skolem function before, the relevant term now contained a free function variable. Even despite the interesting observation that the characteristic sequent sets for the instances 2 to 4 reduce to the same set modulo the parameter, the problem was out of reach of the theorem provers.

As we had expected a decision tree enumerating the - now finite - sub-sequences as possible instance, we investigated if such a tree would constructed in a simpler case. For this we specified a function by a low number of supporting points, added the axioms for if-then-else and claimed that such a function exists. Only one of the theorem provers found the decision tree for a function with two supporting points, none of them managed to find the nesting of if-then-else terms necessary to solve instances with more than two supporting points. The reason is that unification does not find the conditions $x = 0$, $x = s(0)$, etc. since the equalities themselves do not occur in the input problem. This means that without special treatment of if-then-else in the prover, this problem cannot be solved. To track the progress in this matter, we submitted the problem to the TPTP library. The problems NUN24, NUN25 and NUN26 are exactly those examples we investigated in section 4.12.

5.2 Future Work

5.2.1 Open Problems

There are still unsolved problems with the CERES^{ω} (and subsequently the $\mathrm{CERES}^{\omega}_{=}$) method. The foremost one is the lack of a (relative) completeness result, for which it is sufficient to show the relative completeness of R_{al} with regard to Andrew's $\mathcal{R}$ on which it is based. The main difficulties there are the different quantifier rules due to the lack of labeling in $\mathcal{R}$ and the integration of contraction (the *Simp* rule) into R_{al}'s cut rule, essentially restricting it to a contraction on atom formulas. In the case of $\mathrm{CERES}^{\omega}_{=}$, the problem is simplified due to the minor role of labels in it. Still, more work has to be done to complete the proof.

The second open problem lies in the manual interpretation of Skolem functions. In the particular proof investigated, all the points of the function which were calculated in the previous step of the induction become eigenvariables. This significantly simplifies the manual interpretation of the Skolem function. If it turned out that this property is essential for manual interpretation, the results of our analysis would be hard to transfer to other problems. We also believe that the fact that the cut formula of the highest complexity belongs to the Π^0_2 class also plays a role because these formulas can be translated to functions. Again, these phenomena deserve further investigation.

The third open problem concerns the validity preservation of expansion trees under proof rewrite systems like $\rhd_c$. The important ingredients for for lemmas 3.4.7, 3.4.8, 3.4.9 and 3.4.10 are the independence of the permuted operations and the preservation of homomorphic paths. It is quite intuitive that a rewrite rule that preserves paths such that they are homomorphic should have the same expansion tree. Nevertheless, a general theorem of this kind is still missing.

The last open problem is to completely remove the need to compute projections and use expansion proofs instead. The first-order case has been successfully solved by Lolic [64] but the

higher-order case is complicated by the simulation of steps on the leaves of a projection. Such a simulation requires a unique axiom introducing the literal corresponding to the formula resolved on. But in an expansion tree, this axiom might have been merged with other axioms with potentially different labels. Consequently, the simulation of quantifier rules might be unsound for a wrong axiom. Intuitively, because only contractions lead to merges and contractions only appear as the factoring part of the simulation of a resolution inference, it is directly followed by cut. That this is sufficient to prevent unsound leaf simulations needs yet to be proved.

5.2.2 Beyond $\mathrm{CERES}^{\omega}_{=}$

Going further than $\mathrm{CERES}^{\omega}_{=}$ has multiple meanings. First, we can aim for even stronger systems like Gödel's system $\mathcal{T}$, which adds tuples and primitive recursion to simply typed lambda calculus. An even stronger system which is still decidable would be Girard's system $\mathcal{F}$ which also adds type polymorphism [41]. Apart from the technical difficulties of establishing the method, the main problem there is that no automated theorem provers exist for these term languages.

Therefore a second direction to develop the method is to abandon automated theorem proving in favor of interactive theorem proving. In principle, the essential restriction on the calculus used to refute the characteristic sequent set is that there is a translation to sequent calculus which does not apply the cut rule on quantified formulas. This does not hold for natural deduction, but it might be possible to sufficently automatize a tableaux calculus in Isabelle/HOL to be able to find actual refutations. The reason for using HOL is that it is Isabelle's theory which enjoys the most automatation. In contrast to Coq, Isabelle creates an explicit proof object, but it is completely unclear how to extract the proof of the abstraction level we would need.

The third direction is somehow the contraposition to the first one. If we can not automatically refute the sequent sets we obtain right now, we can look for (nontrivial) classes where there exist efficient provers. SMT solvers are increasingly exploring fields like quantifier instantiation [21, 25, 26]. Looking at these classes we could investigate which cuts they introduce and what kind of proofs are possible in these classes.

A central observation there is that refuting the characteristic sequent set essentially amounts to finding a proof in atomic-cut normal form of all the cut formulas at the same time. This direct relation of the cut formulas to the hardness of refuting the characteristic sequent set imposes severe limits on the scalability of method.

Therefore the fourth direction to go on is to work out methods to divide-and-conquer the problem posed by the characteristic sequent set. For instance it is clear that cuts in different branches of the proof can be dealt with independently. Using the original CERES method, this would not change the characteristic sequent set, because the atomic cuts used to simulate resolution cannot be removed via reductive cut-elimination without elimination of the cuts which were skipped. Since we extended the method to ignore propositional cuts, repeated application of the CERES method can now be performed. It is still open how well this influences the scalability of the method.

Direction five is motivated by the observation that the construction of the projections bears a striking resemblance to Maehara's method of constructing an interpolant [65, 105]. Both have the same case distinction for the introduction rule and recombine their components in two ways. The equivalent of the union of projection sets would be the creation of a conjunction and the

equivalent of creation of proofs which merge of all pairs of end-sequents would be a disjunction. Obviously, the cut-formulas do not share the language of the end-sequent but it might be fruitful to investigate other proof theoretic interpolation methods [73] and transfer them to cut-elimination.

Bibliography

[1] Martin Aigner and Günter M. Ziegler. *Proofs from the Book*. Springer Berlin Heidelberg, 2006.

[2] Peter B. Andrews. Resolution in type theory. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 487–507. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[3] David Aspinall, Christoph Lüth, and Daniel Winterstein. A framework for interactive proof. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings*, volume 4573 of *Lecture Notes in Computer Science*, pages 161–175. Springer, 2007.

[4] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. Proof transformation by CERES. In Jonathan M. Borwein and William M. Farmer, editors, *Mathematical Knowledge Management, 5th International Conference, MKM 2006, Wokingham, UK, August 11-12, 2006, Proceedings*, volume 4108 of *Lecture Notes in Computer Science*, pages 82–93. Springer, 2006.

[5] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr. CERES: An analysis of Fürstenberg's proof of the infinity of primes. *Theoretical Computer Science*, 403(2-3):160–175, 2008.

[6] Matthias Baaz and Alexander Leitsch. On skolemization and proof complexity. *Fundamenta Informaticae*, 20(4):353–379, 1994.

[7] Matthias Baaz and Alexander Leitsch. Cut-elimination and redundancy-elimination by resolution. *Journal of Symbolic Computation*, 29(2):149–177, 2000.

[8] Matthias Baaz and Alexander Leitsch. Towards a clausal analysis of cut-elimination. *Journal of Symbolic Computation*, 41(3-4):381–410, 2006.

[9] Christoph Benzmüller. Higher-order automated theorem provers. In David Delahaye and Bruno Woltzenlogel Paleo, editors, *All about Proofs, Proof for All*, Mathematical Logic and Foundations, pages 171–214. College Publications, London, UK, 2015.

[10] Christoph Benzmüller, Chad Brown, and Michael Kohlhase. Cut-simulation and impredicativity. *Logical Methods in Computer Science*, 5(1:6):1–21, 2009.

[11] Christoph Benzmüller and Michael Kohlhase. LEO – a higher-order theorem prover. In Claude Kirchner and Hélène Kirchner, editors, *Automated Deduction - CADE-15, 15th International Conference on Automated Deduction, Lindau, Germany, July 5-10, 1998, Proceedings*, number 1421 in LNCS, pages 139–143. Springer, 1998.

[12] Christoph Benzmüller and Dale Miller. Automation of higher-order logic. In Dov M. Gabbay, Jörg H. Siekmann, and John Woods, editors, *Handbook of the History of Logic, Volume 9 — Computational Logic*, pages 215–254. North Holland, Elsevier, 2014.

[13] Christoph Benzmüller, Lawrence C. Paulson, Nik Sultana, and Frank Theiß. The higher-order prover LEO-II. *Journal of Automated Reasoning*, 55(4):389–404, 2015.

[14] Christoph Benzmüller, Florian Rabe, and Geoff Sutcliffe. THF0 – the core of the TPTP language for classical higher-order logic. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *LNCS*, pages 491–506. Springer, 2008.

[15] Christoph Benzmüller and Nik Sultana. Update report: LEO-II version 1.5. *CoRR*, abs/1303.3761, 2013.

[16] Christoph Benzmüller, Frank Theiss, Lawrence Paulson, and Arnaud Fietzke. LEO-II - a cooperative automatic theorem prover for higher-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *LNCS*, pages 162–170. Springer, 2008.

[17] Ulrich Berger, Wilfried Buchholz, and Helmut Schwichtenberg. Refined program extraction form classical proofs. *Annals of Pure and Applied Logic*, 114(1-3):3–25, 2002.

[18] Ulrich Berger, Kenji Miyamoto, Helmut Schwichtenberg, and Monika Seisenberger. Minlog - A tool for program extraction supporting algebras and coalgebras. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Algebra and Coalgebra in Computer Science - 4th International Conference, CALCO 2011, Winchester, UK, August 30 - September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 393–399. Springer, 2011.

[19] Mirjana Borisavljević. Two measures for proving gentzen's hauptsatz without mix. *Archive for Mathematical Logic*, 42(4):371–387, 2003.

[20] Chad E. Brown. Satallax: An automatic higher-order prover. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 111–117. Springer, 2012.

[21] Chad E. Brown. Reducing higher-order theorem proving to a sequence of SAT problems. *Journal of Automated Reasoning*, 51(1):57–77, 2013.

[22] Alan J. Cain. Deus ex machina and the aesthetics of proof. *The Mathematical Intelligencer*, 32(3):7–11, 2010.

[23] Casc competition 2016 results, website. http://www.cs.miami.edu/~tptp/CASC/J8/WWWFiles/DivisionSummary1.html.

[24] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940.

[25] Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient E-matching for SMT solvers. In Frank Pfenning, editor, *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer, 2007.

[26] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. *Journal of the ACM*, 52(3):365–473, 2005.

[27] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry*, 4(5):235–282, 1994.

[28] Daniel J. Dougherty. Higher-order unification via combinators. *Theoretical Computer Science*, 114(2):273–298, 1993.

[29] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *Journal of Automated Reasoning*, 31(1):33–72, 2003.

[30] Gilles Dowek and Benjamin Werner. Proof normalization modulo. *Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

[31] Cvetan Dunchev. *Automation of cut-elimination in proof schemata*. PhD thesis, Vienna University of Technology, 2012.

[32] Cvetan Dunchev, Gabriel Ebner, Stefan Hetzl, Tomer Libal, Bernhard Mallinger, Sebastian Szivota, Giselle Reis, Martin Riener, Mikheil Rukhaia, Daniel Weller, and Bruno Woltzenlogel-Paleo. Generic Architecture for Proofs – github development page. https://github.com/gapt/gapt.

[33] Cvetan Dunchev, Gabriel Ebner, Stefan Hetzl, Tomer Libal, Bernhard Mallinger, Sebastian Szivota, Giselle Reis, Martin Riener, Mikheil Rukhaia, Daniel Weller, and Bruno Woltzenlogel-Paleo. Generic Architecture for Proofs – webpage. http://www.logic.at/gapt.

[34] Cvetan Dunchev, Alexander Leitsch, Tomer Libal, Martin Riener, Mikheil Rukhaia, Daniel Weller, and Bruno Woltzenlogel-Paleo. PROOFTOOL: a GUI for the GAPT framework. In Cezary Kaliszyk and Christoph Lüth, editors, *UITP*, volume 118 of *EPTCS*, pages 1–14, 2013.

[35] Cvetan Dunchev, Alexander Leitsch, Mikheil Rukhaia, and Daniel Weller. CERES for first-order schemata. *The Computing Research Repository*, abs/1303.4257, 2013.

[36] Gabriel Ebner, Stefan Hetzl, Giselle Reis, Martin Riener, Simon Wolfsteiner, and Sebastian Zivota. System description: GAPT 2.0. In Olivetti and Tiwari [76], pages 293–301.

[37] Michael Färber and Chad E. Brown. Internal guidance for satallax. In Olivetti and Tiwari [76], pages 349–361.

[38] Harvey Friedman. Classically and intuitionistically provably recursive functions. In Gert H. Müller and Dana S. Scott, editors, *Higher Set Theory: Proceedings, Oberwolfach, Germany, April 13–23, 1977*, pages 21–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978.

[39] Gerhard Gentzen. Untersuchungen über das logische Schließen. I, II. *Mathematische Zeitschrift*, 39(1):176–210, 405–431, December 1935.

[40] Jean-Yves Girard. *Proof theory and logical complexity*, volume 1 of *Studies in Proof Theory*. Bibliopolis, Napoli, Italy, 1987.

[41] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, New York, NY, USA, 1989.

[42] Kurt Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica*, 12(3-4):280–287, 1958.

[43] Mike Gordon. From LCF to HOL: a short history. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 169–186. The MIT Press, 2000.

[44] Mike Gordon, Robin Milner, Lockwood Morris, Malcolm Newey, and Christopher Wadsworth. A metalanguage for interactive proof in LCF. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL, pages 119–130, New York, NY, USA, 1978. ACM.

[45] Georg Gottlob and Alexander Leitsch. On the efficiency of subsumption algorithms. *Journal of the ACM*, 32(2):280–295, April 1985.

[46] Godfrey Harold Hardy. *A mathematician's apology*. Canto. Cambridge University Press, Cambridge, UK, 1940.

[47] Stefan Hetzl, Alexander Leitsch, Giselle Reis, Janos Tapolczai, and Daniel Weller. Introducing quantified cuts in logic with equality. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning - 7th International Joint Conference, IJCAR 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 19-22, 2014. Proceedings*, volume 8562 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2014.

[48] Stefan Hetzl, Alexander Leitsch, and Daniel Weller. CERES in higher-order logic. *Annals of Pure and Applied Logic*, 162(12):1001–1034, 2011.

[49] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel-Paleo. Herbrand sequent extraction. In Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors, *AISC/MKM/Calculemus*, volume 5144 of *Lecture Notes in Computer Science*, pages 462–477. Springer, 2008.

[50] Stefan Hetzl, Tomer Libal, Martin Riener, and Mikheil Rukhaia. Understanding Resolution Proofs through Herbrand's Theorem. In Didier Galmiche and Dominique Larchey-Wendling, editors, *Automated Reasoning with Analytic Tableaux and Related Methods (Tableaux 2013)*, volume 8123 of *Lecture Notes in Computer Science*, pages 157–171, 2013.

[51] Yozo Hida, John O. Lamping, and Ramana B. Rao. Tree visualization system and method based upon a compressed half-plane model of hyperbolic geometry, 2005. Patent Number US 6901555 B2.

[52] Mao Lin Huang, Quang Vinh Nguyen, Wei Lai, and Xiaodi Huang. Three-dimensional EncCon tree. In Ebad Banissi, Muhammad Sarfraz, and Natasha Dejdumrong, editors, *CGIV'07: Proceedings of the Computer Graphics, Imaging and Visualisation*, pages 429–433. IEEE Computer Society, 2007.

[53] Gérard Huet, Gilles Kahn, and Christine Paulin-Mohring. The Coq Proof Assistant : A Tutorial : Version 6.1. Rapport de recherche RT-0204, INRIA, August 1997. Projet COQ.

[54] Gérard P. Huet. The undecidability of unification in third order logic. *Information and Control*, 22(3):257–267, 1973.

[55] Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theoretical Computer Science*, 1(1):27–57, 1975.

[56] Thomas Johnsson. Lambda lifting: Treansforming programs to recursive equations. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings*, volume 201 of *Lecture Notes in Computer Science*, pages 190–203. Springer, 1985.

[57] Jean-Pierre Jouannaud. Higher-order rewriting: Framework, confluence and termination. In Aart Middeldorp, Vincent van Oostrom, Femke van Raamsdonk, and Roel C. de Vrijer, editors, *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to*

Jan Willem Klop, on the Occasion of His 60th Birthday, volume 3838 of *Lecture Notes in Computer Science*, pages 224–250. Springer, 2005.

[58] Gerwin Klein, Tobias Nipkow, and Larry Paulson. The archive of formal proofs. http://afp.sourceforge.net.

[59] Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(1):14–25, 1971.

[60] John Lamping, Ramana Rao, and Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In Irvin R. Katz, Robert Mack, Linn Marks, Mary Beth Rosson, and Jakob Nielsen, editors, *CHI'95: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 401–408. ACM Press/Addison-Wesley Publishing Co., 1995.

[61] Alexander Leitsch and Matthias Baaz. *Methods of Cut-Elmination*, volume 34 of *Trends in Logic*. Springer-Verlag New York, Inc., New York, NY, USA, 2011.

[62] Tomer Libal, Martin Riener, and Mikheil Rukhaia. Advanced proof viewing in ProofTool. In Christoph Benzmüller and Bruno Woltzenlogel Paleo, editors, *Proceedings Eleventh Workshop on User Interfaces for Theorem Provers, UITP 2014, Vienna, Austria, 17th July 2014.*, volume 167 of *EPTCS*, pages 35–47, 2014.

[63] Lars Linsen and Sabine Behrendt. Linked treemap: A 3D treemap-nodelink layout for visualizing hierarchical structures. *Computational Statistics*, 26(4):679–697, 2011.

[64] Anela Lolic. Herbrand sequents and the skolem-free ceres method. Master's thesis, Vienna University of Technology, 2015.

[65] Shoji Maehara. On the interpolation theorem of Craig. *Sûgaku*, 12(4):235–237, 1960.

[66] Harry G. Mairson. A simple proof of a theorem of Statman. *Theoretical Computer Science*, 103(2):387–394, 1992.

[67] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, April 1982.

[68] Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998.

[69] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[70] John Meier and Clifford A. Reiter. Fractal representations of Cayley graphs. *Computers and Graphics*, 20(1):163–170, 1996.

[71] Dale A Miller. *Proofs in higher-order logic*. PhD thesis, University of Pennsylvania, 1984.

[72] Dale A. Miller. A compact representation of proofs. *Studia Logica*, 46(4):347–370, 1987.

[73] Nobuyoshi Motohashi. Interpolation theorem and characterization theorem. *Annals of the Japan Association for Philosophy of Science*, 4(2):85–150, 1972.

[74] Tamara Munzner. H3: laying out large directed graphs in 3D hyperbolic space. In John Dill and Nahum D. Gershon, editors, *InfoVis'97: Proceedings of the IEEE Symposium on Information Visualization*, pages 2–10. IEEE Computer Society, 1997.

[75] Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: A Proof Assistant for Higher-order Logic*. Springer-Verlag, Berlin, Heidelberg, 2002.

[76] Nicola Olivetti and Ashish Tiwari, editors. *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*. Springer, 2016.

[77] Grace D Paterson. The aesthetics of mathematical proofs. Master's thesis, University of Alberta, 2013.

[78] Lawrence C Paulson. *Isabelle: A generic theorem prover*, volume 828 of *Lecture Notes in Computer Science*. Springer Verlag Berlin Heidelberg, 1994.

[79] Diana Ratiu and Trifon Trifonov. Exploring the computational content of the infinite pigeonhole principle. *Journal of Logic and Computation*, 22(2):329–350, 2012.

[80] Giselle N. Reis. *Cut-elimination by resolution in intuitionistic logic*. PhD thesis, University of Technology Vienna, 2014.

[81] John A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[82] James Rosindell and Luke J. Harmon. OneZoom: A fractal explorer for the Tree of Life. *Public Library of Science Biology*, 10(10):1–5, 2012.

[83] Mikheil Rukhaia. *CERES in proof schemata*. PhD thesis, Vienna University of Technology, 2012.

[84] Adrian Rusu and Confesor Santiago. A practical algorithm for planar straight-line grid drawings of general trees with linear area and arbitrary aspect ratio. In Ebad Banissi, Remo Aslak Burkhard, Georges Grinstein, Urska Cvek, Marjan Trutschl, Liz Stuart, Theodor G. Wyeld, Gennady Andrienko, Jason Dykes, Mikael Jern, Dennis Groth, and Anna Ursyn, editors, *IV'07: Proceedings of the International Conference on Information Visualisation*, pages 743–750. IEEE Computer Society, 2007.

[85] Hans-Jörg Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, Nov 2011.

[86] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15(2-3):111–126, 2002.

[87] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer, 2013.

[88] Helmut Schwichtenberg. A direct proof of the equivalence between Brouwer's fan theorem and König's lemma with a uniqueness hypothesis. *Journal of Universal Computer Science*, 11(12):2086–2095, 2005.

[89] Helmut Schwichtenberg. Minlog. In Freek Wiedijk, editor, *The Seventeen Provers of the World, Foreword by Dana S. Scott*, volume 3600 of *Lecture Notes in Computer Science*, pages 151–157. Springer, 2006.

[90] Helmut Schwichtenberg. Dialectica interpretation of well-founded induction. *Mathematical Logic Quarterly*, 54(3):229–239, 2008.

[91] Kurt Schütte. Syntactical and semantical properties of simple type theory. *Journal of Symbolic Logic*, 25(4):305–326, 12 1960.

[92] Ben Shneiderman. Tree visualization with tree-maps: A 2-d space-filling approach. *ACM Transactions on Graphics*, 11:92–99, 1991.

[93] Jörg Siekmann, Stephan Hess, Christoph Benzmüller, Lassaad Cheikhrouhou, Detlef Fehrer, Armin Fiedler, Helmut Horacek, Michael Kohlhase, Karsten Konrad, Andreas Meier, Erica Melis, and Volker Sorge. $\mathcal{LOUI}$: A distributed graphical user interface for the interactive proof system ΩMEGA. In *Proceedings of the International Workshop "User Interfaces for Theorem Provers 1998 (UITP'98)*, pages 130–138, Eindhoven, Netherlands, 1998.

[94] Wayne Snyder and Jean H. Gallier. Higher-order unification revisited: Complete sets of transformations. *Journal of Symbolic Computation*, 8(1/2):101–140, 1989.

[95] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Inc., New York, NY, USA, 2006.

[96] Hendrik Spohr, Stefan Hetzl, and Daniel Weller. HLK website. http://logic.at/hlk.

[97] John Stasko and Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In Jock D. Mackinlay, Steven F. Roth, and Daniel A. Keim, editors, *InfoVis'00: Proceedings of the IEEE Symposium on Information Visualization*, pages 57–65. IEEE Computer Society, 2000.

[98] Richard Statman. The typed lambda-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979.

[99] Geoff Sutcliffe. TPTP syntax BNF. http://www.cs.miami.edu/~tptp/TPTP/SyntaxBNF.html.

[100] Geoff Sutcliffe. TPTP, TSTP, CASC, etc. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *Proceedings of the 2nd International Computer Science Symposium in Russia*, number 4649 in Lecture Notes in Computer Science, pages 7–23. Springer-Verlag, 2007.

[101] Geoff Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.

[102] Geoff Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In Edmund M. Clarke and Andrei Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.

[103] William W. Tait. Normal derivability in classical logic. In Jon Barwise, editor, *The Syntax and Semantics of Infinitary Languages*, pages 204–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 1968.

[104] Gaisi Takeuti. On a generalized logic calculus. *Japanese Journal of Mathematics*, 23:39–96, 1953.

[105] Gaisi Takeuti. *Proof Theory*. North-Holland, Elsevier, Amsterdam, 1975.

[106] Steven Trac, Yury Puzis, and Geoff Sutcliffe. An interactive derivation viewer. In Serge Autexier and Christoph Benzmüller, editors, *Proceedings of the 7th Workshop on User Interfaces for Theorem Provers, UITP 2006, Seattle, Washington, USA*, volume 174(2) of *Electronic Notes in Theoretical Computer Science*, pages 109 – 123, 2007. Proceedings of the 7th Workshop on User Interfaces for Theorem Provers (UITP 2006).

[107] Andrzej et. al. Trybulec. Mizar home page. http://www.mizar.org.

[108] Christian Urban. *Classical logic and computation*. PhD thesis, University of Cambridge, October 2000.

[109] Femke van Raamsdonk. Higher-order rewriting. In Paliath Narendran and Michaël Rusinowitch, editors, *Rewriting Techniques and Applications, 10th International Conference, RTA-99, Trento, Italy, July 2-4, 1999, Proceedings*, volume 1631 of *Lecture Notes in Computer Science*, pages 220–239. Springer, 1999.

[110] Jan von Plato. A proof of gentzen's hauptsatz without multicut. *Archive for Mathematical Logic*, 40(1):9–18, 2001.

[111] Daniel Weller. *CERES in Higher-Order Logic*. PhD thesis, Vienna University of Technology, 2010.

[112] Makarius Wenzel. Isabelle/jedit - A prover IDE within the PIDE framework. In Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors, *Intelligent Computer Mathematics - 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*, volume 7362 of *Lecture Notes in Computer Science*, pages 468–471. Springer, 2012.

[113] Charles Wetherell and Alfred Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, SE-5(5):514–520, 1979.

[114] W. Windsteiger. Theorema 2.0: A Graphical User Interface for a Mathematical Assistant System. In Cezary Kaliszyk and Christoph Lueth, editors, *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, Bremen, Germany, July 11th 2012*, volume 118 of *Electronic Proceedings in Theoretical Computer Science*, pages 72–82. Open Publishing Association, 2012. doi 10.4204/EPTCS.118.5.

[115] Jing Yang, Matthew O. Ward, and Elke A. Rundensteiner. InterRing: An interactive tool for visually navigating and manipulating hierarchical structures. In Pak Chung Wong and Keith Andrews, editors, *InfoVis'02: Proceedings of the IEEE Symposium on Information Visualization*, pages 77–84. IEEE Computer Society, 2002.

[116] Hee Yong Youn and Adit D. Singh. Near optimal embedding of binary tree architecture in VLSI. In *ICDCS'88: Proceedings of the International Conference on Distributed Computing Systems*, pages 86–93. IEEE Computer Society, 1988.

A.1 Type Declarations

var	$i, i_1, i_2, j, j_1, k, n, s, x, x_1, y, y_1, z, n_0, n_1$	$:\iota$
var	$\alpha, \beta, \delta, \gamma, \kappa, \nu, \sigma$	$:\iota$
const	f	$:\iota > \iota$
const	AX, T	$:o$
const	$ARITH_1, ARITH_2$	$:o$
const	$ARITH_3$	$:\iota > \iota > o$
const	$ARITH_4$	$:\iota > o$
const	$ARITH_5$	$:\iota > \iota > \iota > o$
const	$BIF0, BIF1$	$:o > \iota > \iota > o$
const	$IIF0, IIF1$	$:\iota > \iota > \iota > o$
const	$C, MONstep$	$:o$
const	$IF0, IF1$	$:\iota > \iota > o$
const	$MONstep_{a1}, MONstep_{a2}, MONstep_{a3}, MONstep_{a4'}, MONstep_{a4}$	$:o$
const	$MONstep_{a5}, MONstep_{a6}, MONstep_{a7}$	$:o$
const	$NOCCstep, NOCCstep_a, NOCCstep_b, INFTAPE, TAPEPROOF, \pi_1$	$:o$
const	$BASE, STEP, MAIN, TRIVIAL$	$:\iota > o$
const	MON	$:(\iota > \iota) > \iota > o$
const	$NOCC$	$:(\iota > \iota) > \iota > \iota > o$
const	t	$:(\iota > \iota) > \iota > \iota > \iota > \iota$
const	A, I	$:\iota > o$
const	ite	$:\iota > \iota > \iota > \iota$
const	be	$:o > \iota$
const	ie	$:\iota > \iota$
var	P, X	$:o$
var	Y	$:\iota > o$
var	h, h', h_1	$:\iota > \iota$
const	$0, 1$	$:\iota$
const	$*, +, bm$	$:\iota > \iota > \iota$
const	$<$	$:\iota > \iota > o$
const	$A1, A2, A3, A4, A5, A6, A7, A8, A9, A10$	$:o$
const	$A11, A12, A13, A14, A15, A16, A17, A18, A19, A20$	$:o$
const	$A21, A22, A23, A24, A30, A31, A32, A33, A34, A35, A36, IND$	$:o$

A.2 Definitions

$$
\begin{array}{ll}
MON(h,n) & \equiv (\forall i \forall j (i < n+1 \wedge j < n+1 \wedge i < j \rightarrow h(i) < h(j))) \\
NOCC(h,n,s) & \equiv (\forall i (i < n+1 \rightarrow f(h(i)) = s)) \\
C & \equiv (\forall n \exists h (MON(h,n) \wedge (\exists s NOCC(h,n,s)))) \\
A(x) & \equiv (\forall n \exists h (MON(h,n) \wedge NOCC(h,n,x))) \\
T & \equiv (\forall n (f(n) = 0 \vee f(n) = 1)) \\
I(s) & \equiv (\forall x \exists y (x < y \wedge f(y) = s)) \\
t(h,n,x,y) & \equiv if \ \llcorner x < n+1 \lrcorner \ then \ h(x) \ else \ y \\
\ulcorner i \urcorner & \equiv 1 \dot{-} i \\
if \ \ulcorner i \urcorner \ then \ x \ else \ y & \equiv \ulcorner i \urcorner * x + (1 \dot{-} \ulcorner i \urcorner) * y \\
if \ \llcorner X \lrcorner \ then \ x \ else \ y & \equiv \llcorner X \lrcorner * x + (1 \dot{-} \llcorner X \lrcorner) * y
\end{array}
$$

A.3 Theory Axioms

Arithmetic:

Basic Axioms:

$$
\begin{array}{ll}
A1 & \vdash \neg(x+1=0) \\
A2 & \vdash x = 0 \vee (\exists k x = k+1) \\
A3 & \vdash x+1 = y+1 \rightarrow x = y \\
A4 & \vdash x = y \rightarrow x+1 = y+1 \\
A5 & \vdash 0 * x = 0 \\
A6 & \vdash (x+1) * y = (x * y) + y \\
A7 & \vdash \neg(x < 0) \\
A8 & \vdash x < y \rightarrow (\exists k (x+k) + 1 = y) \\
A9 & \vdash x < y \vee x = y \vee y < x \\
A30 & \vdash x + 0 = x \\
A31 & \vdash 0 + x = x
\end{array}
$$

Derived Axioms:

$$
\begin{array}{ll}
A10 & \vdash x < y+1 \rightarrow (x < y \vee x = y) \\
A11 & \vdash x = y \vee x < y \rightarrow x < y+1 \\
A12 & \vdash x + y = y + x \\
A13 & \vdash x + (y+z) = (x+y) + z \\
A14 & \vdash x * y = y * x \\
A15 & \vdash x * (y * z) = (x * y) * z \\
A16 & \vdash x = y \rightarrow x + z = y + z \\
A17 & \vdash x + z < y + z \rightarrow x < y \\
A36 & \vdash x < y \rightarrow x + z < y + z \\
A18 & \vdash x = 0 \vee (\exists y y < x) \\
A19 & \vdash 0 < x + 1 \\
A32 & \vdash x < y \rightarrow x + 1 < y + 1 \\
A33 & \vdash \neg(x < x) \\
A34 & \vdash x < y \wedge y < z \rightarrow x < z \\
A35 & \vdash x + 1 < y + 1 \rightarrow x < y
\end{array}
$$

Induction, bounded subtraction and encoding of truth values:

$$
\begin{array}{ll}
IND & \vdash (\forall Y(0) \wedge (\forall n (Y(n) \rightarrow Y(n+1))) \rightarrow (\forall n Y(n))) \\
A20 & \vdash (\forall x (0 \dot{-} x = 0)) \\
A21 & \vdash (\forall x (x \dot{-} 0 = x)) \\
A22 & \vdash (\forall x \forall y ((x+1) \dot{-} (y+1) = x \dot{-} y)) \\
A23 & \vdash (\forall P (P \rightarrow \llcorner P \lrcorner = 1)) \\
A24 & \vdash (\forall P (\neg P \rightarrow \llcorner P \lrcorner = 0))
\end{array}
$$

Unused: A4, A11, A14, A15, A16, A18;

Proof of C

$(INFTAPE)$

$AX, T \vdash I(0), I(1)$

$\exists : r$ — $AX, T \vdash (\exists s I(s)), I(1)$

$\exists : r$ — $AX, T \vdash (\exists s I(s)), (\exists s I(s))$

$c : r$ — $AX, T \vdash (\exists s I(s))$

$(MAIN(\sigma))$

$AX, I(\sigma) \vdash C$

$\exists : l$ — $AX, (existss I(s)) \vdash C$

cut — $AX, AX, T \vdash C$

$c : l$ — $AX, T \vdash C$

$(TAPEPROOF)$

Main

$(BASE(\sigma))$

$AX, I(\sigma) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$(STEP(\sigma))$

$AX, I(\sigma), (\exists h(MON(h,n) \wedge NOCC(h,n,\sigma))) \vdash (\exists h(MON(h,n+1) \wedge NOCC(h,n+1,\sigma)))$

$\rightarrow : r$ — $AX, I(\sigma) \vdash (\exists h(MON(h,n) \wedge NOCC(h,n,\sigma))) \rightarrow (\exists h(MON(h,n+1) \wedge NOCC(h,n+1,\sigma)))$

$\forall : r$ — $AX, I(\sigma) \vdash (\forall n((\exists h(MON(h,n) \wedge NOCC(h,n,\sigma))) \rightarrow (\exists h(MON(h,n+1) \wedge NOCC(h,n+1,\sigma)))))$

$\wedge : r$ — $AX, I(\sigma) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma))) \wedge (\forall n((\exists h(MON(h,n) \wedge NOCC(h,n,\sigma))) \rightarrow (\exists h(MON(h,n+1) \wedge NOCC(h,n+1,\sigma)))))$

$NOCC(h',n_0,\sigma) \vdash NOCC(h',n_0,\sigma)$

$\exists : r$ — $NOCC(h',n_0,\sigma) \vdash (\exists s NOCC(h',n_0,s))$

$MON(h',n_0) \vdash MON(h',n_0)$

$\wedge : r$ — $MON(h',n_0), NOCC(h',n_0,\sigma) \vdash MON(h',n_0) \wedge (\exists s NOCC(h',n_0,s))$

$\wedge : l$ — $MON(h',n_0) \wedge NOCC(h',n_0,\sigma) \vdash MON(h',n_0) \wedge (\exists s NOCC(h',n_0,s))$

$\exists : r$ — $MON(h',n_0) \wedge NOCC(h',n_0,\sigma) \vdash (\exists h(MON(h,n_0) \wedge (\exists s NOCC(h,n_0,s))))$

$\exists : l$ — $(\exists h(MON(h,n_0) \wedge NOCC(h,n_0,\sigma))) \vdash (\exists h(MON(h,n_0) \wedge (\exists s NOCC(h,n_0,s))))$

$\forall : l$ — $(alln(\exists h(MON(h,n) \wedge NOCC(h,n,\sigma)))) \vdash (\exists h(MON(h,n_0) \wedge (\exists s NOCC(h,n_0,s))))$

$\forall : r$ — $(\forall n(\exists h(MON(h,n) \wedge NOCC(h,n,\sigma)))) \vdash (\forall n(\exists h(MON(h,n) \wedge (\exists s NOCC(h,n,s)))))$

def — $(\forall n(\exists h(MON(h,n) \wedge NOCC(h,n,\sigma)))) \vdash C$

$\rightarrow : l$ — $AX, I(\sigma), (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma))) \wedge (\forall n((\exists h(MON(h,n) \wedge NOCC(h,n,\sigma))) \rightarrow (\exists h(MON(h,n+1) \wedge NOCC(h,n+1,\sigma))))) \rightarrow (\forall n(\exists h(MON(h,n) \wedge NOCC(h,n,\sigma)))) \vdash C$

$AXIOM : IND : sub(Y = (\lambda x(\exists h((MON(h,x) \wedge NOCC(h,x,\sigma))))))$ — $AX, I(\sigma) \vdash C$

$(MAIN(\sigma))$

Induction Base

$(\nu+\kappa)+1=0 \vdash ((\nu+\kappa)+1=0)$

$\neg : l$ — $(\nu+\kappa)+1=0, \neg((\nu+\kappa)+1=0) \vdash$

$AXIOM : A1$ — $AX, (\nu+\kappa)+1=0 \vdash$

$EQAX : A13 : \nu+(\kappa+1) = (\nu+\kappa)+1$ — $AX, \nu+(\kappa+1)=0 \vdash$

$EQAX : A12 : 1+\kappa = \kappa+1$ — $AX, \nu+(1+\kappa)=0 \vdash$

$EQAX : A13 : \nu+(1+\kappa) = (\nu+1)+\kappa$ — $AX, (\nu+1)+\kappa=0 \vdash$

$\delta = \nu+1 \vdash \delta = \nu+1$

$=: l$ — $AX, \delta+\kappa=0, \delta=\nu+1 \vdash$

$\exists : l$ — $AX, \delta+\kappa=0, (\exists k(\delta=k+1)) \vdash$

$\delta = 0 \vdash \delta = 0$

$\vee : l$ — $AX, \delta+\kappa=0, \delta=0 \vee (\exists k(\delta=k+1)) \vdash \delta=0$

$AXIOM : A2$ — $AX, \delta+\kappa=0 \vdash \delta=0$

$(\delta+\kappa)+1=0+1 \vdash (\delta+\kappa)+1=0+1$

$\rightarrow : l$ — $AX, (\delta+\kappa)+1=0+1, (\delta+\kappa)+1=0+1 \rightarrow \delta+\kappa=0 \vdash \delta=0$

$AXIOM : A3$ — $AX, (\delta+\kappa)+1=0+1 \vdash \delta=0$

$\exists : l$ — $AX, (\exists k((\delta+k)+1=0+1)) \vdash \delta=0$

$\delta < 0+1 \vdash \delta < 0+1$

$\rightarrow : l$ — $AX, \delta<0+1, \delta<0+1 \rightarrow (\exists k((\delta+k)+1=0+1)) \vdash \delta=0$

$AXIOM : A8$ — $AX, \delta<0+1 \vdash \delta=0$

(π_1)

(π_1)

$AX, \delta<0+1 \vdash \delta=0$

$\gamma<0 \vdash \gamma<0$

$\neg : l$ — $\neg\gamma<0, \gamma<0 \vdash$

$AXIOM : A7$ — $AX, \gamma<0 \vdash$

$=: l$ — $AX, \delta<0+1, \gamma<\delta \vdash$

$w : l$ — $AX, \gamma<0+1, \delta<0+1, \gamma<\delta \vdash$

$w : r$ — $AX, \gamma<0+1, \delta<0+1, \gamma<\delta \vdash \alpha<\alpha$

$\wedge : l$ — $AX, \gamma<0+1, \delta<0+1 \wedge \gamma<\delta \vdash \alpha<\alpha$

$\wedge : l$ — $AX, \gamma<0+1 \wedge \delta<0+1 \wedge \gamma<\delta \vdash \alpha<\alpha$

$\rightarrow : r$ — $AX \vdash \gamma<0+1 \wedge \delta<0+1 \wedge \gamma<\delta \rightarrow \alpha<\alpha$

$\forall : r$ — $AX \vdash (\forall j(\gamma<0+1 \wedge j<0+1 \wedge \gamma<j \rightarrow \alpha<\alpha))$

$\forall : r$ — $AX \vdash (\forall i \forall j(i<0+1 \wedge j<0+1 \wedge i<j \rightarrow \alpha<\alpha))$

def — $AX \vdash MON((\lambda x(\alpha)), 0)$

$f(\alpha)=\sigma \vdash f(\alpha)=\sigma$

$w : l$ — $f(\alpha)=\sigma, \beta<0+1 \vdash f(\alpha)=\sigma$

$\rightarrow : r$ — $f(\alpha)=\sigma \vdash \beta<0+1 \rightarrow f(\alpha)=\sigma$

$\forall : r$ — $f(\alpha)=\sigma \vdash (\forall i(i<0+1 \rightarrow f(\alpha)=\sigma))$

def — $f(\alpha)=\sigma \vdash NOCC((\lambda x(\alpha)), 0, \sigma)$

$\wedge : r$ — $AX, f(\alpha)=\sigma \vdash MON((\lambda x(\alpha)), 0) \wedge NOCC((\lambda x(\alpha)), 0, \sigma)$

$\exists : r$ — $AX, f(\alpha)=\sigma \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$w : l$ — $AX, 0<\alpha, f(\alpha)=\sigma \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$\wedge : l$ — $AX, 0<\alpha \wedge f(\alpha)=\sigma \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$\exists : l$ — $AX, (\exists y(0<y \wedge f(y)=\sigma)) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$\forall : l$ — $AX, (\forall x \exists y(x<y \wedge f(y)=\sigma)) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

def — $AX, I(\sigma) \vdash (\exists h(MON(h,0) \wedge NOCC(h,0,\sigma)))$

$(BASE(\sigma))$

Induction Step

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AX, h'(n) < \alpha, MON(h', n) \vdash i_1 < (n+1)+1 \wedge j_1 < (n+1)+1 \wedge i_1 < j_1 \rightarrow t(h', n, i_1, \alpha) < t(h', n, j_1, \alpha)}\,(MONstep)}{AX, h'(n) < \alpha, MON(h', n) \vdash (\forall j(i_1 < (n+1)+1 \wedge j < (n+1)+1 \wedge i_1 < j \rightarrow t(h', n, i_1, \alpha) < t(h', n, j, \alpha)))}\,\forall : r}{AX, h'(n) < \alpha, MON(h', n) \vdash (\forall i \forall j(i < (n+1)+1 \wedge j < (n+1)+1 \wedge i < j \rightarrow t(h', n, i, \alpha) < t(h', n, j, \alpha)))}\,\forall : r}{AX, h'(n) < \alpha, MON(h', n) \vdash MON((\lambda x(t(h', n, x, \alpha))), n+1)}\,def \quad \dfrac{}{AX, h'(n) < \alpha, f(\alpha) = \sigma, NOCC(h', n, \sigma) \vdash NOCC((\lambda x(t(h', n, x, \alpha))), n+1, \sigma)}\,(NOCCstep)}{AX, h'(n) < \alpha, f(\alpha) = \sigma, MON(h', n), NOCC(h', n, \sigma) \vdash MON((\lambda x(t(h', n, x, \alpha))), n+1) \wedge NOCC((\lambda x(t(h', n, x, \alpha))), n+1, \sigma)}\,\wedge : r}{AX, h'(n) < \alpha, f(\alpha) = \sigma, MON(h', n), NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\exists : r}{AX, h'(n) < \alpha \wedge f(\alpha) = \sigma, MON(h', n), NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\wedge : l}{AX, (\exists y(h'(n) < y \wedge f(y) = \sigma)), MON(h', n), NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\exists : l}{AX, (\forall x \exists y(x < y \wedge f(y) = \sigma)), MON(h', n), NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\forall : l}{AX, I(\sigma), MON(h', n), NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,def}{AX, I(\sigma), MON(h', n) \wedge NOCC(h', n, \sigma) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\wedge : l}{AX, I(\sigma), (\exists h(MON(h, n) \wedge NOCC(h, n, \sigma))) \vdash (\exists h(MON(h, n+1) \wedge NOCC(h, n+1, \sigma)))}\,\exists : l$$

$(STEP(\sigma))$

4 MON step

$$\dfrac{\dfrac{\dfrac{\dfrac{}{i_1 < n+1 \vdash i_1 < n+1} \quad \dfrac{\dfrac{}{j_1 < (n+1) \vdash j_1 < n+1} \quad \dfrac{}{i_1 < j_1 \vdash i_1 < j_1}\,\wedge : r}{j_1 < (n+1), i_1 < j_1 \vdash j_1 < n+1 \wedge i_1 < j_1}\,\wedge : r}{j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash i_1 < n+1 \wedge j_1 < n+1 \wedge i_1 < j_1} \quad \dfrac{}{h'(i_1) < h'(j_1) \vdash h'(i_1) < h'(j_1)}\,\rightarrow : l}{i_1 < n+1 \wedge j_1 < n+1 \wedge i_1 < j_1 \rightarrow h'(i_1) < h'(j_1), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}\,w : l}{AX, i_1 < n+1 \wedge j_1 < n+1 \wedge i_1 < j_1 \rightarrow h'(i_1) < h'(j_1), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}$$

$(MONstep_{a4'})$

$$\dfrac{\dfrac{\dfrac{}{AX \vdash (j_1 < n+1) \rightarrow if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1)}\,(BIF1(j_1 < n+1, h'(j_1), \alpha)) \quad \dfrac{}{j_1 < n+1 \vdash j_1 < n+1} \quad \dfrac{}{if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1) \vdash if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1)}}{j_1 < n+1, (j_1 < n+1) \rightarrow if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1) \vdash if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1)}\,\rightarrow : l}{AX, j_1 < n+1 \vdash if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = h'(j_1)}\,cut \quad \dfrac{\dfrac{\dfrac{}{AX, i_1 < n+1 \wedge j_1 < n+1 \wedge i_1 < j_1 \rightarrow h'(i_1) < h'(j_1), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}\,(MONstep_{a4'})}{AX, (\forall j(i_1 < n+1 \wedge j < n+1 \wedge i_1 < j \rightarrow h'(i_1) < h'(j))), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}\,\forall : l}{AX, (\forall i \forall j(i < n+1 \wedge j < n+1 \wedge i < j \rightarrow h'(i) < h'(j))), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}\,\forall : l}{AX, MON(h', n), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < h'(j_1)}\,def}{AX, MON(h', n), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}\,=: r$$

$(MONstep_{a4})$

$$\dfrac{\dfrac{\dfrac{}{AX \vdash \neg n+1 < n+1 \rightarrow if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}\,(BIF0(n+1 < n+1, h'(j_1), \alpha)) \quad \dfrac{\dfrac{\dfrac{}{n+1 < n+1 \vdash n+1 < n+1}\,\neg : l}{\neg n+1 < n+1, n+1 < n+1 \vdash}\,\neg : r}{\neg n+1 < n+1 \vdash \neg n+1 < n+1} \quad \dfrac{}{if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha \vdash if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}}{\neg n+1 < n+1, \neg n+1 < n+1 \rightarrow if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha \vdash if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}\,\rightarrow : l}{AX, \neg n+1 < n+1 \vdash if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}\,cut}{AX \vdash if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}\,AXIOM : A33$$

$(MONstep_{a6})$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AX \vdash (\forall n(n < n+1))}\,(ARITH_1) \quad \dfrac{\dfrac{}{n < n+1 \vdash n < n+1}}{(\forall n(n < n+1)) \vdash n < n+1}\,\forall : l}{AX \vdash n < n+1}\,cut \quad \dfrac{}{i_1 < n \vdash i_1 < n}}{i_1 < n+1 \vdash i_1 < n+1} \quad \dfrac{}{AX, i_1 < n \vdash n < n+1 \wedge i_1 < n}\,\wedge : r}{AX, i_1 < n+1, i_1 < n \vdash i_1 < n+1 \wedge n < n+1 \wedge i_1 < n}\,\wedge : r \quad \dfrac{\dfrac{}{h'(i_1) < h'(n) \vdash h'(i_1) < h'(n)} \quad \dfrac{}{h'(n) < \alpha \vdash h'(n) < \alpha}\,\wedge : r}{h'(n) < \alpha, h'(i_1) < h'(n) \vdash h'(i_1) < h'(n) \wedge h'(n) < \alpha} \quad \dfrac{}{h'(i_1) < \alpha \vdash h'(i_1) < \alpha}\,\rightarrow : l}{h'(n) < \alpha, h'(i_1) < h'(n), h'(i_1) < h'(n) \wedge h'(n) < \alpha \rightarrow h'(i_1) < \alpha \vdash h'(i_1) < \alpha}\,\rightarrow : l \quad AXIOM : TRANS_{LT}}{AX, h'(n) < \alpha, h'(i_1) < h'(n) \vdash h'(i_1) < \alpha}\,\rightarrow : l}{AX, h'(n) < \alpha, i_1 < n+1 \wedge n < n+1 \wedge i_1 < n \rightarrow h'(i_1) < h'(n), i_1 < n+1, i_1 < n \vdash h'(i_1) < \alpha}\,w : l}{AX, h'(n) < \alpha, i_1 < n+1 \wedge n < n+1 \wedge i_1 < n \rightarrow h'(i_1) < h'(n), i_1 < n+1, i_1 < n, i_1 < j_1 \vdash h'(i_1) < \alpha}$$

$(MONstep_{a7})$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AX, h'(n) < \alpha, i_1 < n+1 \wedge n < n+1 \wedge i_1 < n \rightarrow h'(i_1) < h'(n), i_1 < j_1, i_1 < n+1, i_1 < n \vdash h'(i_1) < \alpha}\,(MONstep_{a7})}{AX, h'(n) < \alpha, (\forall j(i_1 < n+1 \wedge j < n+1 \wedge i_1 < j \rightarrow h'(i_1) < h'(j))), i_1 < j_1, i_1 < n+1, i_1 < n \vdash h'(i_1) < \alpha}\,\forall : l}{AX, h'(n) < \alpha, (\forall i \forall j(i < n+1 \wedge j < n+1 \wedge i < j \rightarrow h'(i) < h'(j))), i_1 < j_1, i_1 < n+1, i_1 < n \vdash h'(i_1) < \alpha}\,\forall : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1, i_1 < n \vdash h'(i_1) < \alpha}\,def \quad \dfrac{\dfrac{}{i_1 = n \vdash i_1 = n} \quad \dfrac{}{h'(n) < \alpha \vdash h'(n) < \alpha}}{h'(n) < \alpha, i_1 = n \vdash h'(i_1) < \alpha}\,\vee : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1, i_1 < n \vee i_1 = n \vdash h'(i_1) < \alpha}\,=: r \quad \dfrac{}{i_1 < n+1 \vdash i_1 < n+1}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1, i_1 < n \vee i_1 = n \vdash h'(i_1) < \alpha}\,\rightarrow : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1 \rightarrow i_1 < n \vee i_1 = n \vdash h'(i_1) < \alpha}\,AXIOM : A10}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < \alpha}\,=: r \quad \dfrac{}{AX \vdash if\ \ulcorner n+1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha = \alpha}}{AX, h'(n) < \alpha, MON(h', n), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}\,=: r \quad \dfrac{}{j_1 = n+1 \vdash j_1 = n+1}$$

$$\dfrac{}{AX, h'(n) < \alpha, MON(h', n), j_1 = n+1, i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}$$

$(MONstep_{a5})$

$$\dfrac{\dfrac{}{AX, MON(h', n), j_1 < (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}\,(MONstep_{a4}) \quad \dfrac{}{AX, h'(n) < \alpha, MON(h', n), j_1 = (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}\,(MONstep_{a5})}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1) \vee j_1 = (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \ulcorner j_1 < n+1 \urcorner\ then\ h'(j_1)\ else\ \alpha)}\,\vee : l$$

$(MONstep_{a3})$

$$\dfrac{\dfrac{}{AX \vdash i_1 < n+1 \to (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1)}(BIF1(i_1 < n+1, h'(i_1), \alpha)) \quad \dfrac{\dfrac{}{i_1 < n+1 \vdash i_1 < n+1} \quad \dfrac{}{(if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1) \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1)} \to : l}{i_1 < n+1, i_1 < n+1 \to (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1) \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1)} \to : l}{AX, i_1 < n+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1)}\ cut$$

$(MONstep_{a2})$

$$\dfrac{\dfrac{\dfrac{}{j_1 < (n+1)+1 \vdash j_1 < (n+1)+1} \quad \dfrac{}{j_1 < (n+1) \vee j_1 = (n+1) \vdash j_1 < (n+1) \vee j_1 = (n+1)} \to : l}{j_1 < (n+1)+1, j_1 < (n+1)+1 \to j_1 < (n+1) \vee j_1 = (n+1) \vdash j_1 < (n+1) \vee j_1 = (n+1)} w : l}{AX, j_1 < (n+1)+1, j_1 < (n+1)+1 \to j_1 < (n+1) \vee j_1 = (n+1) \vdash j_1 < (n+1) \vee j_1 = (n+1)} AXIOM : A10$$

$$\dfrac{\dfrac{}{AX, j_1 < (n+1)+1 \vdash j_1 < (n+1) \vee j_1 = (n+1)} \quad \dfrac{\dfrac{}{AX, AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} c : l}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} =: r}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} cut$$

$(MONstep_{a3})$

$$\dfrac{}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1) \vee j_1 = (n+1), i_1 < j_1, i_1 < n+1 \vdash h'(i_1) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)}$$

$(MONstep_{a1})$

$$\dfrac{\dfrac{}{AX, i_1 < n+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) = h'(i_1)}(MONstep_{a2}) \quad \dfrac{\dfrac{\dfrac{\dfrac{}{AX, i_1 < j_1, j_1 < (n+1)+1 \vdash i_1 < n+1}(ARITH_5(i_1, j_1, n)) \quad \dfrac{}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)}(MONstep_{a1})}{AX, AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1, j_1 < (n+1)+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} cut}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < j_1, i_1 < n+1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} c : l}{AX, h'(n) < \alpha, MON(h', n), j_1 < (n+1)+1, i_1 < (n+1)+1, i_1 < j_1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < (if\ \lfloor j_1 < n+1 \rfloor\ then\ h'(j_1)\ else\ \alpha)} w : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < (n+1)+1, j_1 < (n+1)+1, i_1 < j_1 \vdash (if\ \lfloor i_1 < n+1 \rfloor\ then\ h'(i_1)\ else\ \alpha) < t(h', n, j_1, \alpha)} def}{AX, h'(n) < \alpha, MON(h', n), i_1 < (n+1)+1, j_1 < (n+1)+1, i_1 < j_1 \vdash t(h', n, i_1, \alpha) < t(h', n, j_1, \alpha)} \wedge : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < (n+1)+1, j_1 < (n+1)+1 \wedge i_1 < j_1 \vdash t(h', n, i_1, \alpha) < t(h', n, j_1, \alpha)} \wedge : l}{AX, h'(n) < \alpha, MON(h', n), i_1 < (n+1)+1 \wedge j_1 < (n+1)+1 \wedge i_1 < j_1 \vdash t(h', n, i_1, \alpha) < t(h', n, j_1, \alpha)} \to : r}{AX, h'(n) < \alpha, MON(h', n) \vdash i_1 < (n+1)+1 \wedge j_1 < (n+1)+1 \wedge i_1 < j_1 \to t(h', n, i_1, \alpha) < t(h', n, j_1, \alpha)}$$

$(MONstep)$

5 NOCCstep

$$\dfrac{\dfrac{}{AX \vdash i_2 < n+1 \to (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2)}(BIF1(i_2 < n+1, h'(i_2), \alpha)) \quad \dfrac{\dfrac{\dfrac{}{i_2 < n+1 \vdash i_2 < n+1} \quad \dfrac{}{NOCC(h', n, \sigma), i_2 < n+1, (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2) \vdash (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2)} \to : l}{NOCC(h', n, \sigma), i_2 < n+1, i_2 < n+1 \to (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2) \vdash (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2)} c : l}{AX, NOCC(h', n, \sigma), i_2 < n+1 \vdash (if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = h'(i_2)} cut$$

$(NOCCstep_a)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{i_2 < n+1 \vdash i_2 < n+1} \quad \dfrac{}{f(h'(i_2)) = \sigma \vdash f(h'(i_2)) = \sigma} \to : l}{i_2 < n+1 \to f(h'(i_2)) = \sigma, i_2 < n+1 \vdash f(h'(i_2)) = \sigma} \forall : l}{(\forall i(i < n+1 \to f(h'(i)) = \sigma)), i_2 < n+1 \vdash f(h'(i_2)) = \sigma} def}{NOCC(h', n, \sigma), i_2 < n+1 \vdash f(h'(i_2)) = \sigma} =: r$$

$$\dfrac{\dfrac{\dfrac{}{n+1 < n+1 \vdash n+1 < n+1} \neg : l}{\neg n+1 < n+1, n+1 < n+1 \vdash} \neg : r \quad \dfrac{}{if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha = \alpha \vdash if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha = \alpha} \quad \dfrac{}{f(\alpha) = \sigma \vdash f(\alpha) = \sigma} =: r}{\neg n+1 < n+1 \vdash \neg n+1 < n+1} \quad \dfrac{}{f(\alpha) = \sigma, if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha = \alpha \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha) = \sigma} \to : l}{f(\alpha) = \sigma, \neg n+1 < n+1, \neg n+1 < n+1 \to if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha = \alpha \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha) = \sigma} cut$$

$$\dfrac{\dfrac{}{AX \vdash \neg n+1 < n+1 \to if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha = \alpha}(BIF0(n+1 < n+1, h'(n+1), \alpha))}{AX, f(\alpha) = \sigma, \neg n+1 < n+1 \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha) = \sigma} \quad \dfrac{}{AX, f(\alpha) = \sigma \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha) = \sigma} AXIOM : A33$$

$(NOCCstep_b)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{i_2 < (n+1)+1 \vdash i_2 < (n+1)+1} \quad \dfrac{\dfrac{}{i_2 = (n+1) \vdash i_2 = (n+1)} \quad \dfrac{\dfrac{}{i_2 = (n+1) \vdash i_2 = (n+1)} \quad \dfrac{}{AX, f(\alpha) = \sigma \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(n+1)\ else\ \alpha) = \sigma} (NOCCstep_b)}{AX, f(\alpha) = \sigma, i_2 = (n+1) \vdash f(if\ \lfloor n+1 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = \sigma} =: r}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma), i_2 < n+1 \vee i_2 = (n+1) \vdash f(if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = \sigma} \vee : l}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma), i_2 < (n+1)+1, i_2 < (n+1)+1 \to (i_2 < n+1 \vee i_2 = (n+1)) \vdash f(if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = \sigma} \to : l}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma), i_2 < (n+1)+1 \vdash f(if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = \sigma} AXIOM : A10$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma), i_2 < (n+1)+1 \vdash f(if\ \lfloor i_2 < n+1 \rfloor\ then\ h'(i_2)\ else\ \alpha) = \sigma} def}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma), i_2 < (n+1)+1 \vdash f(t(h', n, i_2, \alpha)) = \sigma} \to : r}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma) \vdash i_2 < (n+1)+1 \to f(t(h', n, i_2, \alpha)) = \sigma} \forall : r}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma) \vdash (\forall i(i < (n+1)+1 \to f(t(h', n, i, \alpha)) = \sigma))} def}{AX, f(\alpha) = \sigma, NOCC(h', n, \sigma) \vdash NOCC((\lambda x(t(h', n, x, \alpha))), n+1, \sigma)} w : l}{AX, h'(n) < \alpha, f(\alpha) = \sigma, NOCC(h', n, \sigma) \vdash NOCC((\lambda x(t(h', n, x, \alpha))), n+1, \sigma)}$$

$(NOCCstep)$

6 Lemmas for if-then-else

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \gamma = \gamma}}{AX \vdash 0 + \gamma = \gamma}\,EQAX : A31 : 0 + \gamma = \gamma}{AX \vdash (0 * \gamma) + \gamma = \gamma}\,EQAX : A5 : 0 * \gamma = 0}{AX \vdash (0+1) * \gamma = \gamma}\,EQAX : A6 : (0+1) * \gamma = (0 * \gamma) + \gamma}{AX \vdash 1 * \gamma = \gamma}\,EQAX : A31 : 0 + 1 = 1}{AX \vdash (1 \dot{-} 0) * \gamma = \gamma}\,EQAX : A21 : 1 \dot{-} 0 = 1}{AX \vdash 0 + ((1 \dot{-} 0) * \gamma) = \gamma}\,EQAX : A31 : 0 + ((1 \dot{-} 0) * \gamma) = ((1 \dot{-} 0) * \gamma)}{AX \vdash (0 * \beta) + ((1 \dot{-} 0) * \gamma) = \gamma}\,EQAX : A5 : 0 * \beta = 0$$

$(IF0(\beta, \gamma))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \beta = \beta}}{AX \vdash 0 + \beta = \beta}\,EQAX : A31 : 0 + \beta = \beta}{AX \vdash (0 * \beta) + \beta = \beta}\,EQAX : A5 : 0 * \beta = 0}{AX \vdash (0+1) * \beta = \beta}\,EQAX : A6 : (0+1) * \beta = (0 * \beta) + \beta}{AX \vdash 1 * \beta = \beta}\,EQAX : A31 : 0 + 1 = 1}{AX \vdash (1 * \beta) + 0 = \beta}\,EQAX : A30 : (1 * \beta) + 0 = 1 * \beta}{AX \vdash (1 * \beta) + (0 * \gamma) = \beta}\,EQAX : A5 : 0 * \gamma = 0}{AX \vdash (1 * \beta) + ((0 \dot{-} 0) * \gamma) = \beta}\,EQAX : A20 : 0 \dot{-} 0 = 0}{AX \vdash (1 * \beta) + (((0+1) \dot{-} (0+1)) * \gamma) = \beta}\,EQAX : A22 : (0+1) \dot{-} (0+1) = 0 \dot{-} 0}{AX \vdash (1 * \beta) + (((0+1) \dot{-} 1) * \gamma) = \beta}\,EQAX : A30 : 0 + 1 = 1}{AX \vdash (1 * \beta) + ((1 \dot{-} 1) * \gamma) = \beta}\,EQAX : A30 : 0 + 1 = 1$$

$(IF1(\beta, \gamma))$

$$\dfrac{\dfrac{\dfrac{\dfrac{X \vdash X}{\neg X, X \vdash}\,\neg : l}{\neg X \vdash \neg X}\,\neg : r}{\neg X, \neg X \to {}_\iota X_\lrcorner = 0 \vdash {}_\iota X_\lrcorner = 0}\,\to : l \qquad \dfrac{\dfrac{\dfrac{X \vdash X}{\neg X, X \vdash}\,\neg : l}{\neg X \vdash \neg X}\,\neg : r}{\neg X, \neg X \to {}_\iota X_\lrcorner = 0 \vdash {}_\iota X_\lrcorner = 0}\,\to : l \quad AXIOM : A24 \qquad (IF0(\alpha, \beta))}{AX, \neg X \vdash {}_\iota X_\lrcorner = 0} \quad AXIOM : A24 \qquad \dfrac{AX \vdash 0 * \alpha + (1 \dot{-} 0) * \beta = \beta}{AX, \neg X \vdash 0 * \alpha + (1 \dot{-} {}_\iota X_\lrcorner) * \beta = \beta}\,=: r$$

$$\dfrac{\dfrac{AX, \neg X \vdash {}_\iota X_\lrcorner * \alpha + (1 \dot{-} {}_\iota X_\lrcorner) * \beta = \beta}{AX, \neg X \vdash if\ {}_\iota X_\lrcorner\ then\ \alpha\ else\ \beta = \beta}\,def}{AX \vdash \neg X \to if\ {}_\iota X_\lrcorner\ then\ \alpha\ else\ \beta = \beta}\,\to : r$$

$(BIF0(X, \alpha, \beta))$

$$\dfrac{\dfrac{\dfrac{X \vdash X}{X, X \to {}_\iota X_\lrcorner = 1 \vdash {}_\iota X_\lrcorner = 1}\,\to : l}{AX, X \vdash {}_\iota X_\lrcorner = 1}\,AXIOM : A23 \qquad \dfrac{\dfrac{X \vdash X}{X, X \to {}_\iota X_\lrcorner = 1 \vdash {}_\iota X_\lrcorner = 1}\,\to : l \quad AXIOM : A23 \qquad (IF1(\alpha, \beta))}{AX, X \vdash {}_\iota X_\lrcorner = 1} \qquad \dfrac{AX \vdash 1 * \alpha + (1 \dot{-} 1) * \beta = \alpha}{AX, X \vdash {}_\iota X_\lrcorner * \alpha + (1 \dot{-} {}_\iota X_\lrcorner) * \beta = \alpha}\,=: r$$

$$\dfrac{\dfrac{AX, X \vdash {}_\iota X_\lrcorner * \alpha + (1 \dot{-} {}_\iota X_\lrcorner) * \beta = \alpha}{AX, X \vdash if\ {}_\iota X_\lrcorner\ then\ \alpha\ else\ \beta = \alpha}\,def}{AX \vdash X \to if\ {}_\iota X_\lrcorner\ then\ \alpha\ else\ \beta = \alpha}\,\to : r$$

$(BIF1(X, \alpha, \beta))$

$$\dfrac{\dfrac{\dfrac{}{{}^\ulcorner \alpha^\urcorner = 0 \vdash {}^\ulcorner \alpha^\urcorner = 0} \qquad \dfrac{\dfrac{\dfrac{}{{}^\ulcorner \alpha^\urcorner = 0 \vdash {}^\ulcorner \alpha^\urcorner = 0} \qquad AX \vdash 0 * \beta + (1 \dot{-} 0) * \gamma = \gamma}{AX, {}^\ulcorner \alpha^\urcorner = 0 \vdash 0 * \beta + (1 \dot{-} {}^\ulcorner \alpha^\urcorner) * \gamma = \gamma}\,=: r}{AX, {}^\ulcorner \alpha^\urcorner = 0 \vdash {}^\ulcorner \alpha^\urcorner * \beta + (1 \dot{-} {}^\ulcorner \alpha^\urcorner) * \gamma = \gamma}\,=: r}{AX, {}^\ulcorner \alpha^\urcorner = 0 \vdash (if\ {}^\ulcorner \alpha^\urcorner\ then\ \beta\ else\ \gamma) = \gamma}\,def}{AX \vdash {}^\ulcorner \alpha^\urcorner = 0 \to (if\ {}^\ulcorner \alpha^\urcorner\ then\ \beta\ else\ \gamma) = \gamma}\,\to : r$$

$(IIF0(\alpha, \beta, \gamma))$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{{}^\ulcorner \alpha^\urcorner = 1 \vdash {}^\ulcorner \alpha^\urcorner = 1} \qquad \dfrac{\dfrac{\dfrac{}{{}^\ulcorner \alpha^\urcorner = 1 \vdash {}^\ulcorner \alpha^\urcorner = 1} \qquad AX \vdash 1 * \beta + (1 \dot{-} 1) * \gamma = \beta}{AX, {}^\ulcorner \alpha^\urcorner = 1 \vdash 1 * \beta + (1 \dot{-} {}^\ulcorner \alpha^\urcorner) * \gamma = \beta}\,=: r}{AX, {}^\ulcorner \alpha^\urcorner = 1 \vdash {}^\ulcorner \alpha^\urcorner * \beta + (1 \dot{-} {}^\ulcorner \alpha^\urcorner) * \gamma = \beta}\,=: r}{AX, {}^\ulcorner \alpha^\urcorner = 1 \vdash (if\ {}^\ulcorner \alpha^\urcorner\ then\ \beta\ else\ \gamma) = \beta}\,def}{AX \vdash {}^\ulcorner \alpha^\urcorner = 1 \to (if\ {}^\ulcorner \alpha^\urcorner\ then\ \beta\ else\ \gamma) = \beta}\,\to : r$$

$(IIF1(\alpha, \beta, \gamma))$

7 Arithmetic

$$\dfrac{\dfrac{\dfrac{}{0 < 0 + 1 \vdash 0 < 0 + 1}\,AXIOM : A19 : 0 < x + 1}{AX \vdash 0 < 0 + 1} \qquad \dfrac{\dfrac{\alpha < \alpha + 1 \to \alpha + 1 < (\alpha + 1) + 1 \vdash \alpha < \alpha + 1 \to \alpha + 1 < (\alpha + 1) + 1}{AX \vdash \alpha < \alpha + 1 \to \alpha + 1 < (\alpha + 1) + 1}\,AXIOM : A32 : x < y \to x + 1 < y + 1}{AX \vdash (\forall n(n < n + 1 \to n + 1 < (n + 1) + 1))}\,\forall : r}{AX \vdash 0 < 0 + 1 \wedge ((\forall n(n < n + 1 \to n + 1 < (n + 1) + 1)))}\,\wedge : r \qquad (\forall n(n < n + 1)) \vdash (\forall n(n < n + 1))$$

$$\dfrac{AX, 0 < 0 + 1 \wedge ((\forall n(n < n + 1 \to n + 1 < (n + 1) + 1))) \to (\forall n(n < n + 1)) \vdash (\forall n(n < n + 1))}{AX \vdash (\forall n(n < n + 1))}\,AXIOM : IND : sub(Y = (\lambda x(x < x + 1)))\,\to : l$$

$(ARITH_1)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{0 + 1 < 0 \vdash 0 + 1 < 0}}{\neg 0 + 1 < 0, 0 + 1 < 0 \vdash}\,\neg : l}{\neg 0 + 1 < 0 \vdash \neg 0 + 1 < 0}\,\neg : r}{AX \vdash \neg 0 + 1 < 0}\,AXIOM : A1 : \neg(x + 1 < 0)}{AX \vdash \neg 0 + 1 < 0 \wedge (\forall n(\neg(n + 1 < n) \to \neg((n + 1) + 1 < n + 1)))}\,\wedge : r \qquad (\forall n(\neg n + 1 < n)) \vdash (\forall n(\neg n + 1 < n))}{AX, (\neg 0 + 1 < 0 \wedge (\forall n(\neg(n + 1 < n) \to \neg((n + 1) + 1 < n + 1)))) \to (\forall n(\neg(n + 1 < n))) \vdash (\forall n(\neg n + 1 < n))}\,AXIOM : IND : sub(Y = (\lambda x(\neg x + 1 < x)))\,\to : l}{AX \vdash (\forall n(\neg n + 1 < n))}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(n+1) + 1 < n + 1 \vdash (n+1) + 1 < n + 1}}{(n+1) + 1 < n + 1 \to n + 1 < n, (n+1) + 1 < n + 1 \vdash n + 1 < n}\,\to : l}{(n+1) + 1 < n + 1 \to n + 1 < n, \neg n + 1 < n, (n+1) + 1 < n + 1 \vdash}\,\neg : l}{(n+1) + 1 < n + 1 \to n + 1 < n, \neg n + 1 < n \vdash \neg(n+1) + 1 < n + 1}\,\neg : r}{(n+1) + 1 < n + 1 \to n + 1 < n \vdash \neg(n+1 < n) \to \neg((n+1) + 1 < n + 1)}\,\to : r}{AX \vdash \neg(n + 1 < n) \to \neg((n+1) + 1 < n + 1)}\,\forall : r}{AX \vdash (\forall n(\neg(n + 1 < n) \to \neg((n+1) + 1 < n + 1)))}\,\wedge : r \qquad n + 1 < n \vdash n + 1 < n$$

$(ARITH_2)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x_1 < y_1 \wedge y_1 < x_1 \vdash x_1 < y_1 \wedge y_1 < x_1} \qquad \dfrac{\dfrac{}{x_1 < x_1 \vdash x_1 < x_1}}{\neg x_1 < x_1, x_1 < x_1 \vdash}\,\neg : l}{\neg x_1 < x_1, x_1 < y_1 \wedge y_1 < x_1, x_1 < y_1 \wedge y_1 < x_1 \to x_1 < x_1 \vdash}\,\to : l}{AX, x_1 < y_1 \wedge y_1 < x_1, x_1 < y_1 \wedge y_1 < x_1 \to x_1 < x_1 \vdash}\,AXIOM : A33 : IRREF_{LT}}{AX, x_1 < y_1 \wedge y_1 < x_1 \vdash}\,AXIOM : A34 : TRANS_{LT}}{AX \vdash \neg(x_1 < y_1 \wedge y_1 < x_1)}\,\neg : r$$

$(ARITH_3(x_1, y_1))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{1<0\vdash 1<0}\,\neg:l}{1<0,\neg 1<0\vdash}\,AXIOM:A7}{AX,1<0\vdash}}{x_1+1<x_1+0\vdash x_1+1<x_1+0}\,\rightarrow:l}{AX,x_1+1<x_1+0\rightarrow 1<0,x_1+1<x_1+0\vdash}\,EQAX:1+x_1=x_1+1}{AX,1+x_1<x_1+0\rightarrow 1<0,x_1+1<x_1+0\vdash}\,EQAX:0+x_1=x_1+0}{AX,1+x_1<0+x_1\rightarrow 1<0,x_1+1<x_1+0\vdash}\,AXIOM:A17:CANCEL_{PLUS}}{AX,x_1+1<x_1+0\vdash}\,EQAX:A30:x_1+0=x_1}{AX,x_1+1<x_1\vdash}\,\neg:r}{AX\vdash\neg x_1+1<x_1}$$

$(ARITH_4(x_1))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{j<(n+1)+1\vdash j<(n+1)+1}\quad\dfrac{\dfrac{\dfrac{\dfrac{}{i<j\vdash i<j}\quad\dfrac{}{j<n+1\vdash j<n+1}\,\wedge:r}{i<j,j<n+1\vdash i<j\wedge j<n+1}\quad\dfrac{}{i<n+1\vdash i<n+1}\,\rightarrow:l}{i<j,j<n+1,i<j\wedge j<n+1\rightarrow i<n+1\vdash i<n+1}\,AXIOM:A34}{AX,i<j,j<n+1\vdash i<n+1}\quad\dfrac{\dfrac{}{j=n+1\vdash j=n+1}\quad\dfrac{}{i<n+1\vdash i<n+1}\,=:l}{i<j,j=n+1\vdash i<n+1}\,\vee:l}{AX,i<j,j<n+1|j=n+1\vdash i<n+1}\,\rightarrow:l}{AX,i<j,j<(n+1)+1,j<(n+1)+1\rightarrow(j<n+1|j=n+1)\vdash i<n+1}\,AXIOM:A10}{AX,i<j,j<(n+1)+1\vdash i<n+1}$$

$(ARITH_5(i,j,n))$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{0<(n_1+1)\vdash 0<(n_1+1)}\,AXIOM:A19}{AX\vdash 0<(n_1+1)}\quad\dfrac{}{0+n_0<(n_1+1)+n_0\vdash 0+n_0<(n_1+1)+n_0}\,\rightarrow:l}{AX,0<(n_1+1)\rightarrow 0+n_0<(n_1+1)+n_0\vdash 0+n_0<(n_1+1)+n_0}\,AXIOM:A36}{AX\vdash 0+n_0<(n_1+1)+n_0}\,EQAX:A12:(n_1+1)+n_0=n_0+(n_1+1)$$
$$\dfrac{AX\vdash 0+n_0<n_0+(n_1+1)}{AX\vdash n_0<n_0+(n_1+1)}\,EQAX:A31:0+n_0=n_0$$
$$\dfrac{AX\vdash n_0<n_0+(n_1+1)}{AX\vdash n_0<(n_0+n_1)+1}\,EQAX:A13:n_0+(n_1+1)=(n_0+n_1)+1$$

$(LTSUM(n_0,n_1))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{AX\vdash n_0<(n_0+n_1)+1}\quad\dfrac{\dfrac{\dfrac{\dfrac{}{AX\vdash n_1<(n_1+n_0)+1}\,EQAX:A12:n_1+n_0=n_0+n_1}{AX\vdash n_1<(n_0+n_1)+1}\quad(LTSUM(n_1,n_0))}{AX,T\vdash f((n_0+n_1)+1)=1,(n_1<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=0}\,\wedge:r}{AX,T\vdash(n_0<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=1,(n_1<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=0}\,\wedge:r}{AX,T\vdash(n_0<(n_0+n_1)+1)\wedge f((n_0+n_1)+1)=1,(\exists y((n_1<y)\wedge f(y)=0))}\,\exists:r}{AX,T\vdash(\exists y((n_0<y)\wedge f(y)=1)),(\exists y((n_1<y)\wedge f(y)=0))}\,\exists:r}{AX,T\vdash(\exists y((n_0<y)\wedge f(y)=1)),(\forall x\exists y((x<y)\wedge f(y)=0))}\,\forall:r}{AX,T\vdash(\forall x\exists y((x<y)\wedge f(y)=1)),(\forall x\exists y((x<y)\wedge f(y)=0))}\,\forall:r}{AX,T\vdash(\forall x\exists y((x<y)\wedge f(y)=1)),I(0)}\,def}{AX,T\vdash I(1),I(0)}$$

$(INFTAPE)$

B.1 Version 2, Prover 9 refutation

```
set( prolog_style_variables ).

============================== PROOF =================================

% ---------- Comments from original proof ----------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 32.
% Level of proof is 9.
% Maximum clause weight is 21.000.
% Given clauses 61.


1 f(A) != 1 | -(s26(q2) < s26(q2) + 1) | -(s25(q2,s26(q2)) < B) | -(s25(q2,s26(q2)) < B) | f(B) != 1 # label(sequent0) # label(axiom
    ). [assumption].
2 f(A) != 1 | -(s26(q2) < s26(q2) + 1) | -(s25(q2,s26(q2)) < B) | f(B) != 1. [copy(1),merge(d)].
5 f(A) != 0 | -(s10(q1) < s10(q1) + 1) | -(s9(q1,s10(q1)) < B) | -(s9(q1,s10(q1)) < B) | f(B) != 0 # label(sequent2) # label(axiom).
    [assumption].
6 f(A) != 0 | -(s10(q1) < s10(q1) + 1) | -(s9(q1,s10(q1)) < B) | f(B) != 0. [copy(5),merge(d)].
9 f((A + B) + 1) = 0 | f((A + B) + 1) = 1 # label(sequent4) # label(axiom). [assumption].
13 0 + A = A # label(sequent7) # label(axiom). [assumption].
16 A + B = B + A # label(sequent9) # label(axiom). [assumption].
17 0 + A < (B + 1) + A # label(sequent10) # label(axiom). [assumption].
18 A < (B + 1) + A. [copy(17), rewrite([13(2)])].
20 A + (B + 1) = (A + B) + 1 # label(sequent13) # label(axiom). [assumption].
21 1 + (A + B) = A + (B + 1). [copy(20), rewrite([16(6)]), flip(a)].
23 f(A) != 1 | -(s26(q2) < 1 + s26(q2)) | -(s25(q2,s26(q2)) < A). [factor(2,a,d), rewrite([16(9)])].
24 f(A) != 0 | -(s10(q1) < 1 + s10(q1)) | -(s9(q1,s10(q1)) < A). [factor(6,a,d), rewrite([16(9)])].
25 f(1 + (A + B)) = 0 | f(1 + (A + B)) = 1. [back_rewrite(9), rewrite([16(3),16(9)])].
27 A < 1 + A. [para(13(a,1),18(a,2,1))].
30 f(A) != 0 | -(s9(q1,s10(q1)) < A). [back_unit_del(24), unit_del(b,27)].
31 f(A) != 1 | -(s25(q2,s26(q2)) < A). [back_unit_del(23), unit_del(b,27)].
32 (A + 1) + B = 1 + (B + A). [para(21(a,2),16(a,1)), flip(a)].
33 1 + (A + B) = B + (A + 1). [para(16(a,1),21(a,1,2))].
37 1 + (1 + (A + B)) = A + (1 + (B + 1)). [para(21(a,2),21(a,1,2)), rewrite([16(9)])].
39 f(1 + A) = 0 | f(1 + A) = 1. [para(13(a,1),25(a,1,1,2)), rewrite([13(8)])].
46 f((A + 1) + s9(q1,s10(q1))) != 0. [resolve(30,b,18,a)].
48 f((A + 1) + s25(q2,s26(q2))) != 1. [resolve(31,b,18,a)].
58 f(1 + (s9(q1,s10(q1)) + A)) != 0. [para(16(a,1),46(a,1,1)), rewrite([21(7,R)])].
69 (A + 1) + B = 1 + (A + B). [para(16(a,1),32(a,2,2))].
77 f(1 + (A + s25(q2,s26(q2)))) != 1. [back_rewrite(48), rewrite([69(7)])].
82 f(A + (1 + s25(q2,s26(q2)))) != 1. [para(21(a,1),77(a,1,1)), rewrite([16(6)])].
96 f(1 + (1 + (A + s9(q1,s10(q1))))) != 0. [para(33(a,2),58(a,1,1,2))].
101 f((1 + s25(q2,s26(q2))) + A) != 1. [para(16(a,1),82(a,1,1))].
174 f(1 + (1 + ((1 + s25(q2,s26(q2))) + A))) != 1. [para(37(a,2),101(a,1,1))].
210 f(1 + (1 + (A + s9(q1,s10(q1))))) = 1. [resolve(39,a,96,a)].
211 $F. [resolve(210,a,174,a)].

============================== end of proof ==========================
```

B.2 Version 3, Prover 9 refutation

```
set( prolog_style_variables ).

============================== PROOF =================================

% ---------- Comments from original proof ----------
% Proof 1 at 0.01 (+ 0.00) seconds.
% Length of proof is 19.
% Level of proof is 6.
% Maximum clause weight is 25.000.
% Given clauses 19.


1 0 + A < (B + 1) + A # label(sequent0) # label(axiom). [assumption].
2 A + (B + 1) = (A + B) + 1 # label(sequent1) # label(axiom). [assumption].
3 (A + B) + 1 = A + (B + 1). [copy(2), flip(a)].
4 f((A + B) + 1) = 0 | f((A + B) + 1) = 1 # label(sequent2) # label(axiom). [assumption].
5 f(A + (B + 1)) = 0 | f(A + (B + 1)) = 1. [copy(4), rewrite([3(3),3(9)])].
7 f(A) != B | -(s10(q1(B)) < s10(q1(B)) + 1) | -(s9(q1(B),s10(q1(B))) < C) | f(C) != B # label(sequent4) # label(axiom). [
      assumption].
8 A + B = B + A # label(sequent5) # label(axiom). [assumption].
9 0 + A = A # label(sequent6) # label(axiom). [assumption].
11 f(A) != B | -(s10(q1(B)) < 1 + s10(q1(B))) | -(s9(q1(B),s10(q1(B))) < A). [factor(7,a,d), rewrite([8(8)])].
12 1 + (A + B) = A + (B + 1). [back_rewrite(3), rewrite([8(3)])].
13 A < (B + 1) + A. [back_rewrite(1), rewrite([9(2)])].
15 f((A + 1) + B) = 0 | f(B + (A + 1)) = 1. [para(8(a,1),5(a,1,1))].
27 f((A + 1) + s9(q1(B),s10(q1(B)))) != B | -(s10(q1(B)) < 1 + s10(q1(B))). [resolve(13,a,11,c)].
31 A < 1 + A. [para(9(a,1),13(a,2,1))].
33 f((A + 1) + s9(q1(B),s10(q1(B)))) != B. [back_unit_del(27), unit_del(b,31)].
52 f(1 + (s9(q1(A),s10(q1(A))) + B)) != A. [para(8(a,1),33(a,1,1)), rewrite([12(7,R)])].
60 f(1 + (s9(q1(0),s10(q1(0))) + A)) = 1. [resolve(15,a,33,a), rewrite([12(9,R)])].
76 f(1 + (A + s9(q1(B),s10(q1(B))))) != B. [para(8(a,1),52(a,1,1,2))].
77 $F. [resolve(76,a,60,a)].

============================== end of proof ==========================
```

Full resolution simulation example for chapter

$$\dfrac{\dfrac{\dfrac{\forall x P(x,s) \vdash \forall x P(x,s)}{\forall x P(x,s) \vdash \forall x P(x,s), \forall x P(x,t)}\, w:r}{\vdash \forall x P(x,s), \forall x P(x,s) \to \forall x P(x,t)}\, \to\!:r}{\forall X(X(s) \to X(t)) \vdash \forall x P(x,s), \forall x P(x,s) \to \forall x P(x,t)}\, w:l$$

(π_1)

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle P(v,s)\rangle^{\lambda x P(x,s)} \vdash \langle P(v,s)\rangle^{\lambda x P(x,s)}}{\forall x P(x,s) \vdash \langle P(v,s)\rangle^{\lambda x P(x,s)}}\, \forall^{sk}:l}{\forall x P(x,s) \vdash \langle P(v,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,t)}\, w:r}{\vdash \langle P(v,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,s) \to \forall x P(x,t)}\, \to\!:r}{\forall X(X(s) \to X(t)) \vdash \langle P(v,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,s) \to \forall x P(x,t)}\, w:l$$

(π_2)

$$\dfrac{\dfrac{\dfrac{\dfrac{\langle P(d,s)\rangle^{\lambda x P(x,s)} \vdash \langle P(d,s)\rangle^{\lambda x P(x,s)}}{\forall x P(x,s) \vdash \langle P(d,s)\rangle^{\lambda x P(x,s)}}\, \forall^{sk}:l}{\forall x P(x,s) \vdash \langle P(d,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,t)}\, w:r}{\vdash \langle P(d,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,s) \to \forall x P(x,t)}\, \to\!:r}{\forall X(X(s) \to X(t)) \vdash \langle P(d,s)\rangle^{\lambda x P(x,s)}, \forall x P(x,s) \to \forall x P(x,t)}\, w:l$$

(π_3)

$$\dfrac{\dfrac{\dfrac{Y(s) \vdash Y(s) \qquad Y(t) \vdash Y(t)}{Y(s), Y(s) \to Y(t) \vdash Y(t)}\, \to\!:l}{Y(s), \forall X(X(s) \to X(t)) \vdash Y(t)}\, \forall^{sk}:l}{Y(s), \forall X(X(s) \to X(t)) \vdash Y(t), \forall x P(x,s) \to \forall x P(x,t)}\, w:r$$

(π_4)

$$\dfrac{\dfrac{\dfrac{\forall xP(x,s) \vdash \forall xP(x,s) \qquad \forall xP(x,t) \vdash \forall xP(x,t)}{\forall xP(x,s), \forall xP(x,s) \to \forall xP(x,t) \vdash \forall xP(x,t)} \to : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \forall xP(x,t)} \forall^{sk} : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \forall xP(x,t), \forall xP(x,s) \to \forall xP(x,t)} w : r$$

(π_5)

$$\dfrac{\dfrac{\dfrac{\forall xP(x,s) \vdash \forall xP(x,s) \qquad \dfrac{\langle P(u,t)\rangle^{\lambda xP(x,t)} \vdash \langle P(u,t)\rangle^{\lambda xP(x,t)}}{\forall xP(x,t) \vdash \langle P(u,t)\rangle^{\lambda xP(x,t)}} \forall^{sk} : l}{\forall xP(x,s), \forall xP(x,s) \to \forall xP(x,t) \vdash \langle P(u,t)\rangle^{\lambda xP(x,t)}} \to : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \langle P(u,t)\rangle^{\lambda xP(x,t)}} \forall^{sk} : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \langle P(u,t)\rangle^{\lambda xP(x,t)}, \forall xP(x,s) \to \forall xP(x,t)} w : r$$

(π_6)

$$\dfrac{\dfrac{\dfrac{\forall xP(x,s) \vdash \forall xP(x,s) \qquad \dfrac{\langle P(c,t)\rangle^{\lambda xP(x,t)} \vdash \langle P(c,t)\rangle^{\lambda xP(x,t)}}{\forall xP(x,t) \vdash \langle P(c,t)\rangle^{\lambda xP(x,t)}} \forall^{sk} : l}{\forall xP(x,s), \forall xP(x,s) \to \forall xP(x,t) \vdash \langle P(c,t)\rangle^{\lambda xP(x,t)}} \to : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \langle P(c,t)\rangle^{\lambda xP(x,t)}} \forall^{sk} : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \langle P(c,t)\rangle^{\lambda xP(x,t)}, \forall xP(x,s) \to \forall xP(x,t)} w : r$$

(π_7)

$$\dfrac{\dfrac{\dfrac{\forall xP(x,t) \vdash \forall xP(x,t)}{\forall xP(x,t), \forall xP(x,s) \vdash \forall xP(x,t)} w : l}{\forall xP(x,t) \vdash \forall xP(x,s) \to \forall xP(x,t)} \to : r}{\forall xP(x,t), \forall X(X(s) \to X(t)) \vdash \forall xP(x,s) \to \forall xP(x,t)} w : l$$

(π_8)

$$\dfrac{\dfrac{\dfrac{\dfrac{P(c,t) \vdash P(c,t)}{P(c,t) \vdash \forall xP(x,t)} \forall : r}{P(c,t), \forall xP(x,s) \vdash \forall xP(x,t)} w : l}{P(c,t) \vdash \forall xP(x,s) \to \forall xP(x,t)} \to : r}{P(c,t), \forall X(X(s) \to X(t)) \vdash \forall xP(x,s) \to \forall xP(x,t)} w : l$$

(π_9)

$$\dfrac{\dfrac{\dfrac{(\pi_7) \qquad (\pi_9)}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash P(c,t), \forall xP(x,s) \to \forall xP(x,t) \qquad P(c,t), \forall X(X(s) \to X(t)) \vdash \forall xP(x,s) \to \forall xP(x,t)} cut}{\forall xP(x,s), \forall X(X(s) \to X(t)), \forall X(X(s) \to X(t)) \vdash \forall xP(x,s) \to \forall xP(x,t), \forall xP(x,s) \to \forall xP(x,t)} c : l}{\forall xP(x,s), \forall X(X(s) \to X(t)) \vdash \forall xP(x,s) \to \forall xP(x,t), \forall xP(x,s) \to \forall xP(x,t)} c : r$$

(π_{10})

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(d,s)\vdash P(d,s)}{P(d,s)\vdash \forall x P(x,s)}\,\forall^{sk}:r \quad \dfrac{\langle P(c,t)\rangle^{\lambda x P(x,t)}\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}{\forall x P(x,t)\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\,\forall^{sk}:l}{P(d,s),\forall x P(x,s)\to \forall x P(x,t)\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\,\to:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)}}\,\forall^{sk}:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \langle P(c,t)\rangle^{\lambda x P(x,t)},\forall x P(x,s)\to \forall x P(x,t)}\,w:r \quad \begin{array}{c}(\pi_9)\\ P(c,t),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t)\end{array}}{\dfrac{\dfrac{\dfrac{P(d,s),\forall X(X(s)\to X(t)),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t),\forall x P(x,s)\to \forall x P(x,t)}{P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t),\forall x P(x,s)\to \forall x P(x,t)}\,c:l}{P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t)}\,c:r}\,cut$$

(π_{11})

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}(\pi_3)\\ \forall X(X(s)\to X(t))\vdash P(d,s),\forall x P(x,s)\to \forall x P(x,t)\end{array} \quad \begin{array}{c}(\pi_{11})\\ P(d,s),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t)\end{array}}{\forall X(X(s)\to X(t)),\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t),\forall x P(x,s)\to \forall x P(x,t)}\,cut}{\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t),\forall x P(x,s)\to \forall x P(x,t)}\,c:l}{\forall X(X(s)\to X(t))\vdash \forall x P(x,s)\to \forall x P(x,t)}\,c:r$$

(π_{12})

LLK LaTeXStyle

The file is also for download at http://www.logic.at/staff/riener/static/llk.sty.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{ llk }

\RequirePackage{bussproofs}

\newcommand{\lkproves}[0]{\ensuremath{\vdash}}
\renewcommand{\fCenter}{\lkproves}
\newcommand{\apply}[1]{#1}

\newcommand{\AX}[2]{\AxiomC{\ensuremath{#1} \fCenter \ensuremath{#2}}}
\newcommand{\UI}[2]{\UnaryInfC{\ensuremath{#1} \fCenter \ensuremath{#2}}}
\newcommand{\BI}[2]{\BinaryInfC{\ensuremath{#1} \fCenter \ensuremath{#2}}}
\newcommand{\LL}[1]{\LeftLabel{\footnotesize \ensuremath{#1}}}
\newcommand{\RL}[1]{\RightLabel{\footnotesize \ensuremath{#1}}}
\newcommand{\RLN}[1]{\RightLabel{#1}}

% labels
\newcommand{\SALLL}{\RL{\forall:l}}
\newcommand{\SALLR}{\RL{\forall:r}}
\newcommand{\SEXL}{\RL{\exists:l}}
\newcommand{\SEXR}{\RL{\exists:r}}
\newcommand{\SALLSKL}{\RL{\forall^{sk}:l}}
\newcommand{\SALLSKR}{\RL{\forall^{sk}:r}}
\newcommand{\SEXSKL}{\RL{\exists^{sk}:l}}
\newcommand{\SEXSKR}{\RL{\exists^{sk}:r}}
\newcommand{\SANDL}{\RL{\land:l}}
\newcommand{\SANDR}{\RL{\land:r}}
\newcommand{\SORL}{\RL{\lor:l}}
\newcommand{\SORR}{\RL{\lor:r}}
\newcommand{\SIMPL}{\RL{\impl:l}}
\newcommand{\SIMPR}{\RL{\impl:r}}
\newcommand{\SNEGL}{\RL{\neg:l}}
\newcommand{\SNEGR}{\RL{\neg:r}}
\newcommand{\SEQL}{\RL{=:l}}
\newcommand{\SEQSKL}{\RL{=_{sk}:l}}
\newcommand{\SEQLA}{\RL{=:l_1}}
\newcommand{\SEQLB}{\RL{=:l_2}}
\newcommand{\SEQR}{\RL{=:r}}
\newcommand{\SEQSKR}{\RL{=_{sk}:r}}
\newcommand{\SEQRA}{\RL{=:r_1}}
```

```
\newcommand{\SEQRB}{\RL{=:r_2}}
\newcommand{\SWEAKL}{\RL{w:l}}
\newcommand{\SWEAKR}{\RL{w:r}}
\newcommand{\SWEAKRL}{\RL{w:∗}}
\newcommand{\SCONTRL}{\RL{c:l}}
\newcommand{\SCONTRR}{\RL{c:r}}
\newcommand{\SCONTRRL}{\RL{c:∗}}
\newcommand{\SCUT}{\RL{cut}}
\newcommand{\SDEF}{\RL{def}}
\newcommand{\SBETA}{\RL{\beta}}
\newcommand{\SINSTLEMMA}[1]{\RL{LEMMA: #1}}
\newcommand{\SINSTAXIOM}[1]{\RL{AXIOM: #1}}
\newcommand{\SEQAXIOM}[1]{\RL{EQAX: #1}}

%lk rules
\newcommand{\ALLL}   [3]{\SALLL \UI{#2}{#3} }
\newcommand{\ALLR}   [3]{\SALLR \UI{#2}{#3} }
\newcommand{\EXL}    [3]{\SEXL  \UI{#2}{#3} }
\newcommand{\EXR}    [3]{\SEXR  \UI{#2}{#3} }
\newcommand{\ALLSKL} [3]{\SALLSKL \UI{#2}{#3} }
\newcommand{\ALLSKR} [3]{\SALLSKR \UI{#2}{#3} }
\newcommand{\EXSKL}   [3]{\SEXSKL \UI{#2}{#3} }
\newcommand{\EXSKR}   [3]{\SEXSKR \UI{#2}{#3} }
\newcommand{\ANDL}  [2]{\SANDL \UI{#1}{#2} }
\newcommand{\ANDR}  [2]{\SANDR \BI{#1}{#2} }
\newcommand{\ORL}    [2]{\SORL \BI{#1}{#2} }
\newcommand{\ORR}    [2]{\SORR \UI{#1}{#2} }
\newcommand{\IMPL}   [2]{\SIMPL \BI{#1}{#2}}
\newcommand{\IMPR}   [2]{\SIMPR \UI{#1}{#2}}
\newcommand{\NEGL}   [2]{\SNEGL \UI{#1}{#2}}
\newcommand{\NEGR}   [2]{\SNEGR \UI{#1}{#2}}
\newcommand{\EQL}    [2]{\SEQL \BI{#1}{#2}}
\newcommand{\EQSKL} [2]{\SEQSKL \BI{#1}{#2}}
\newcommand{\EQLA}   [2]{\SEQLA \BI{#1}{#2}}
\newcommand{\EQLB}   [2]{\SEQLB \BI{#1}{#2}}
\newcommand{\EQR}    [2]{\SEQR \BI{#1}{#2}}
\newcommand{\EQSKR} [2]{\SEQSKR \BI{#1}{#2}}
\newcommand{\EQRA}   [2]{\SEQRA \BI{#1}{#2}}
\newcommand{\EQRB}   [2]{\SEQRB \BI{#1}{#2}}
\newcommand{\WEAKL} [2]{\SWEAKL \UI{#1}{#2}}
\newcommand{\WEAKR} [2]{\SWEAKR \UI{#1}{#2}}
\newcommand{\CONTRL} [2]{\SCONTRL \UI{#1}{#2}}
\newcommand{\CONTRR} [2]{\SCONTRR \UI{#1}{#2}}
\newcommand{\CUT}    [2]{\SCUT    \BI{#1}{#2}}
\newcommand{\DEF}    [2]{\SDEF    \UI{#1}{#2}}
\newcommand{\BETA}   [2]{\SBETA   \UI{#1}{#2}}
\newcommand{\INSTLEMMA}[3]{\SINSTLEMMA{#1} \UI{#2}{#3}}
\newcommand{\INSTAXIOM}[3]{\SINSTAXIOM{#1} \UI{#2}{#3}}
\newcommand{\EQAXIOM} [3]{\SEQAXIOM{#1} \UI{#2}{#3}}


\newcommand{\CONTINUEWITH}[1]{
 \noLine
```

```
 \UnaryInfC{\ensuremath{(#1)}}
}

\newcommand{\CONTINUEFROM}[3]{
 \AxiomC{\ensuremath{(#1)}}
 \noLine
 \UI{#2}{#3}
}

% unary equational  rules
\newcommand{\SEQUL}{\RL{=:ul}}
\newcommand{\SEQUR}{\RL{=:ur}}
\newcommand{\UEQR}[2]{\SEQUR \UI{#1}{#2}}
\newcommand{\UEQL}[2]{\SEQUL \UI{#1}{#2}}



% environments
\newenvironment{declaration}[0]{
\ section {Type  Declarations }
$
\begin{array}{ll@{ :  }l}
}{
\end{array}
$
}
\newenvironment{theoryaxioms}[0]{
\ section {Theory Axioms}
}{
}

\newenvironment{ definitions }[0]{
\ section { Definitions }
$
\begin{array}{l@{\;\equiv\;  }l}
}{
\end{array}
$
}

\newcommand{\TYPEDEC}[3]{ #1 & #2 & #3 \\}
\newcommand{\CONSTDEC}[2]{\TYPEDEC{const}{#1}{#2}}
\newcommand{\VARDEC}[2]{\TYPEDEC{var}{#1}{#2}}

\newcommand{\AXIOMDEC}[3]{#1 & #2 & #3 \\}
\newcommand{\SEQUENT}[2]{{\ensuremath{#1 \lkproves #2}} }
\newcommand{\SEQUENTLINE}[2]{\ensuremath{#1 \lkproves #2};\\ }

\newcommand{\PREDDEF}[2]{#1 & #2 \\}
\newcommand{\FUNDEF}[2]{#1 & #2 \\}

\newcommand{\ienc}[1]{\ensuremath{\ulcorner{#1}\urcorner}}
\newcommand{\benc}[1]{\ensuremath{\llcorner{#1}\lrcorner}}
```

```
\newcommand{\impl}{\ensuremath{\rightarrow}}
\newcommand{\dimpl}{\ensuremath{\leftrightarrow}}
\newcommand{\bm}{\ensuremath{\dotdiv}}
\newcommand{\spc}[0]{\mbox{ }}
\newcommand{\ite}[3]{\ensuremath{if\spc #1\spc then\spc #2\spc else\spc #3}}

\newcommand{\MON}[2]{(\forall i \forall j (i<#2+1 \land j<#2+1 \land i<j \impl #1(i)<#1(j))) }
\newcommand{\NOCC}[3]{(\forall i (i<#2+1 \impl f(#1(i))=#3)) }



% Ral %

\newcommand{\SSUB}[0]{\RL{Sub}}
\newcommand{\SNEGT}[0]{\RL{\neg^T}}
\newcommand{\SORT}[0]{\RL{\lor^T}}
\newcommand{\SANDLT}[0]{\RL{\land^T_l}}
\newcommand{\SANDRT}[0]{\RL{\land^T_r}}
\newcommand{\SIMPT}[0]{\RL{\impl^T}}
\newcommand{\SALLT}[0]{\RL{\forall^T}}
\newcommand{\SEXISTST}[0]{\RL{\exists^T}}
\newcommand{\SNEGF}[0]{\RL{\neg^F}}
\newcommand{\SORLF}[0]{\RL{\lor^F_l}}
\newcommand{\SORRF}[0]{\RL{\lor^F_r}}
\newcommand{\SANDF}[0]{\RL{\land^F}}
\newcommand{\SIMPLF}[0]{\RL{\impl^F_l}}
\newcommand{\SIMPRF}[0]{\RL{\impl^F_r}}
\newcommand{\SALLF}[0]{\RL{\forall^F}}
\newcommand{\SEXISTSF}[0]{\RL{\exists^F}}
\newcommand{\SEQF}[0]{\RL{=^F}}
\newcommand{\SEQT}[0]{\RL{=^T}}
\newcommand{\SEQFF}[0]{\RL{=^F_{flip}}}
\newcommand{\SEQFT}[0]{\RL{=^T_{flip}}}

\newcommand{\SUB}[2]{\SSUB \UI{#1}{#2}}
\newcommand{\NEGT}[2]{\SNEGT \UI{#1}{#2}}
\newcommand{\ORT}[2]{\SORT \UI{#1}{#2} }
\newcommand{\ANDLT}[2]{\SANDLT \UI{#1}{#2}}
\newcommand{\ANDRT}[2]{\SANDRT \UI{#1}{#2}}
\newcommand{\IMPT}[2]{\SIMPT \UI{#1}{#2}}
\newcommand{\ALLT}[3]{\SALLT \UI{#2}{#3}}
\newcommand{\EXISTST}[3]{\SEXISTST \UI{#2}{#3}}
\newcommand{\NEGF}[2]{\SNEGF \UI{#1}{#2}}
\newcommand{\ORLF}[2]{\SORLF \UI{#1}{#2}}
\newcommand{\ORRF}[2]{\SORRF \UI{#1}{#2}}
\newcommand{\ANDF}[2]{\SANDF \UI{#1}{#2}}
\newcommand{\IMPLF}[2]{\SIMPLF \UI{#1}{#2}}
\newcommand{\IMPRF}[2]{\SIMPRF \UI{#1}{#2}}
\newcommand{\ALLF}[3]{\SALLF \UI{#2}{#3}}
\newcommand{\EXISTSF}[3]{\SEXISTSF \UI{#2}{#3}}
\newcommand{\EQF}[2]{\SEQF \BI{#1}{#2}}
\newcommand{\EQT}[2]{\SEQF \BI{#1}{#2}}
\newcommand{\EQFF}[2]{\SEQFF \BI{#1}{#2}}
```

```
\newcommand{\EQFT}[2]{\SEQFT \BI{#1}{#2}}


\endinput
```

GAPT Proof Analysis Script

The file is also available as `examples/ntape/nTape.scala` within the examples of the GAPT distribution.

```
package at.logic.gapt.proofs.ceres_omega

import at.logic.gapt.expr._
import at.logic.gapt.expr.hol.universalClosure
import at.logic.gapt.formats.tptp.TPTPHOLExporter
import at.logic.gapt.proofs.expansion._
import at.logic.gapt.proofs.lk._
import at.logic.gapt.expr.fol.{ reduceHolToFol, replaceAbstractions, undoHol2Fol }
import at.logic.gapt.formats.llk.ExtendedProofDatabase
import at.logic.gapt.proofs.ceres._
import at.logic.gapt.proofs.{ HOLClause, HOLSequent, Sequent }
import at.logic.gapt.proofs.resolution.{ Resolution2RalWithAbstractions, ResolutionToLKProof }
import at.logic.gapt.provers.eprover.EProver
import at.logic.gapt.provers.prover9.Prover9
import at.logic.gapt.utils.{ TimeOutException, withTimeout }

import scala.concurrent.duration.Duration

/**
 * The generic template for using ceres_omega to analyze a proof. It performs the following steps:
 *
 * 1) eliminate definitions ([[AnalysisWithCeresOmega.input_proof]]
 *
 * 2) eliminate definitions ([[AnalysisWithCeresOmega.preprocessed_input_proof1]]
 *
 * 3) expand non-atomic axioms ([[AnalysisWithCeresOmega.preprocessed_input_proof2]])
 *
 * 4) make the proof regular ([[AnalysisWithCeresOmega.preprocessed_input_proof]])
 *
 * 5) convert it to lk_sk ([[AnalysisWithCeresOmega.lksk_proof]])
 *
 * 6) compute the struct, css and projections ([[AnalysisWithCeresOmega.css]],
 *
 *    [[AnalysisWithCeresOmega.projections]], [[AnalysisWithCeresOmega.struct]])
 *
 * 7) map the css to first-order by lambda lifting and erasure of types
 *    ([[AnalysisWithCeresOmega.fol_css]])
 *
 * 8) try to find a refutation of the css
```

```
 *     ([[AnalysisWithCeresOmega.fol_refutation]])
 *
 * 9) reintroduce types (might fail because type erasure is a heuristic which is unsound in general)
 *     (no method available, included in step 11)
 *
 * 10) reintroduce terms abstracted away by lambda lifting
 *     (no method available, included in step 11)
 *
 * 11) construct an r_al proof from the refutation
 *     ([[AnalysisWithCeresOmega.ral_refutation]])
 *
 * 12) construct the acnf
 *     ([[AnalysisWithCeresOmega.acnf]])
 *
 * 13) construct the expansion proof (with atomic cuts)
 *     ([[AnalysisWithCeresOmega.expansion_proof]])
 *
 * 14) print statistics
 *     ([[AnalysisWithCeresOmega.printStatistics]])
 */
abstract class AnalysisWithCeresOmega {
  /** The proof database to start from. */
  def proofdb(): ExtendedProofDatabase

  /** The name of the root proof to start with */
  def root_proof(): String

  /** Determines if and which cuts should be taken into accoutn for cut-elimination. Default:
      propositional cuts are skipped. */
  def skip_strategy() = CERES.skipPropositional( _ )

  /**
   * Timeout for call to theorem provers.
   *
   * @return the timeout as duration. default: 60 seconds
   */
  def timeout(): Duration = Duration( "10s" )

  /**
   * The input LK proof, extracted by the name [[root_proof]] from the proof database ([[proofdb]])
   */
  lazy val input_proof = proofdb proof root_proof

  /**
   * The input proof (TAPEPROOF) after preprocessing step 1: definition elimination
   */
  lazy val preprocessed_input_proof1 = eliminateDefinitions( proofdb.Definitions )( input_proof )

  /**
   * The input proof after preprocessing step 2: expansion of logical axioms to atomic axioms
   */
  lazy val preprocessed_input_proof2 = AtomicExpansion( preprocessed_input_proof1 )
```

```
/**
 * The input proof preprocessing step 3: regularization
 */
lazy val preprocessed_input_proof3 = regularize ( preprocessed_input_proof2 )

/**
 * The input proof (TAPEPROOF) after definition elimination ([[ preprocessed_input_proof1 ]], expansion
      of logical axioms
 * to atomic axioms ([[ preprocessed_input_proof2 ]]) and regularization ([[ preprocessed_input_proof3
      ]])
 */
lazy val preprocessed_input_proof = preprocessed_input_proof3

/**
 * The processed input proof converted to LKsk.
 */
lazy val lksk_proof = skolemizeInferences ( preprocessed_input_proof )

/**
 * The struct of the proof. It is an intermediate representation of the characteristic sequent set.
 */
lazy val struct = extractStruct ( lksk_proof, skip_strategy () )

/**
 * The set of projections of the [[ preprocessed_input_proof ]].
 */
lazy val projections = Projections ( lksk_proof, skip_strategy () )

/**
 * The characteristic sequent set of the [[ preprocessed_input_proof ]].
 */
lazy val css = StandardClauseSet( struct )

/**
 * The characteristic sequent set ([[ css ]]) after removal of labels and subsumption
 */
lazy val preprocessed_css : List[HOLSequent] = {
  val stripped_css = css
  subsumedClausesRemoval( stripped_css.toList )
}

/**
 * The first order export of the preprocessed characteristic sequent set ([[ preprocessed_css ]]),
      together with the map of
 * replacing constants.
 */
lazy val ( abstracted_constants_map, fol_css ) = {
  val css_nolabels = preprocessed_css // remove labels from css
  val ( abs_consts, abs_css ) = replaceAbstractions ( css_nolabels )
  /* map types to first order*/
  val fol_css = reduceHolToFol( abs_css )
  /* converting to clause form, this is cleaner than casting */
  val fol_ccs = fol_css map {
```

```
      case Sequent( ant, succ ) =>
        HOLClause(
          ant map { case atom @ FOLAtom( _, _ ) => atom },
          succ map { case atom @ FOLAtom( _, _ ) => atom }
        )
    }
    ( abs_consts, fol_ccs )
  }

  /**
   * The first order refutation of the first order characteristic sequent set ([[ fol_css ]])
   */
  lazy val fol_refutation = {
    val some_rp = try {
      val css = fol_css // evaluate lazy val, otherwise the thread stays blocked
      withTimeout( timeout() ) { Prover9.getResolutionProof( css ) }
    } catch {
      case e: TimeOutException =>
        println( s"Could not refute the clause set within ${timeout()}." )
        throw e
    }

    some_rp match {
      case None =>
        throw new Exception( "Could not refute clause set!" )
      case Some( rp ) =>
        rp
    }
  }

  /**
   * The expansion proof of the first-order refutation ([[ fol_refutation ]]).
   */
  lazy val fol_refutation_expansion_proof = {
    val lk_rp = ResolutionToLKProof( fol_refutation )
    LKToExpansionProof( lk_rp )
  }

  /**
   * Exports the preprocessed characteristic sequent ([[ preprocessed_css ]]) set to the TPTP THF
       format
   *
   * @param filename The name of the file to export to
   */
  def export_thf( filename: String ): Unit = {
    TPTPHOLExporter( preprocessed_css, filename )
  }

  /**
   * The ral version of the first-order refutation ([[ fol_refutation ]]), with all necessary
       simplifications undone
   */
  lazy val ral_refutation = {
```

```
    val signature = undoHol2Fol.getSignature( lksk_proof, identity[Formula] )

    val converter = Resolution2RalWithAbstractions( signature, abstracted_constants_map )

    converter( fol_refutation )
  }

  /**
   * The simulation of the [[ ral_refutation ]] on the [[ projections ]] i.e. an LKsk proof where cuts only
       work on atom formulas
   */
  lazy val acnf = CERES( lksk_proof.conclusion, projections, ral_refutation )

  /**
   * The expansion proof of the cut-free proof [[acnf]].
   */
  lazy val expansion_proof = LKToExpansionProof( acnf )

  /**
   * A first-order conversion of the deep formula of the [[expansion_proof]].
   */
  lazy val expansion_proof_fol_deep = reduceHolToFol( replaceAbstractions( expansion_proof.
      expansionSequent.deep.toImplication ) )

  /**
   * The proof of the deep formula of the [[expansion_proof]].
   */
  lazy val reproved_deep = {
    EProver.getResolutionProof( expansion_proof_fol_deep ) match {
      case None => throw new Exception( "Could not reprove deep formula!" )
      case Some( p ) => p
    }
  }

  def thf_reproving_deep( filename: Option[String] ): String = {
    filename match {
      case Some( fn ) =>
        TPTPHOLExporter.apply( expansion_proof.expansionSequent, fn, true, true )
      case None =>
        ()
    }

    TPTPHOLExporter.export( expansion_proof.expansionSequent, true, true )
  }

  def printStatistics() = {
    println( "------------- Proof sizes -------------" )
    println( s"Input proof                : ${input_proof.treeLike.size}" )
    println( s"Preprocessed input         : ${preprocessed_input_proof.treeLike.size}" )
    println( s"LKsk input proof           : ${lksk_proof.treeLike.size}" )
    println( s"ACNF output proof          : ${acnf.treeLike.size}" )
    println( "-------------" )
    println( s"Css size                   : ${css.size}" )
```

```
    println( s"Preprocessed css size   : ${preprocessed_css.size}" )
    println( "-------------" )
    println( s"Refutation size (dag)   : ${fol_refutation.dagLike.size}" )
    println( s"Refutation size (tree)  : ${fol_refutation.treeLike.size}" )
    println( s"Refutation depth        : ${fol_refutation.depth}" )
    println( "-------------" )
    println( s"Reproved deep formula proof size (dag)  : ${reproved_deep.dagLike.size}" )
    println( s"Reproved deep formula proof size (tree) : ${reproved_deep.treeLike.size}" )
  }

  /**
   * Prints the preprocessed characteristic sequent set in TPTP THF format. Use [[export_thf]] to write
       it to a file.
   */
  def print_css_thf(): Unit = {
    println( TPTPHOLExporter.export_negative( preprocessed_css ) )
  }

}
```