FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Integrating Theories into Inference Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Martin Riener
Matrikelnummer 9927068

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuer: Univ.-Prof. Dr. Phil. Alexander Leitsch

Wien, 22.03.2011 ______________________ ______________________
(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschließlich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Martin Riener

Abstract

The axiomatization of arithmetical properties in theorem proving creates many straightforward inference steps. In analyzing mathematical proofs with the CERES (Cut-Elimination by Resolution) system, it is convenient to hide these inferences. The central topic of the thesis is the extension of the CERES method to allow reasoning modulo equational theories. For this, the inference systems of Sequent Calculus modulo and Extended Narrowing and Resolution replace their non-equational counterparts in CERES. The method is illustrated by examples comparing inference modulo the theory of associativity and commutativity with unit element to inference in the empty theory.

Zusammenfassung

Die Axiomatisierung arithmetischer Gesetze im Bereich des automatischen Beweisens erzeugt viele für Menschen intuitive Ableitungsschritte. Bei der Analyse mathematischer Beweise mit der CERES Methode ist es von Vorteil, diese auszublenden. Diese Arbeit erweitert CERES um integrierte Gleichungstheorien. Dazu wird die Methode auf einen Sequenzenkalkül modulo und einen passenden Resolutionskalkül übertragen, welche beide dem Prinzip der Deduction Modulo entstammen. Als laufendes Beispiel wird die Theorie modulo kommutativer Monoide verwendet und mit der ursprünglichen Methode verglichen.

Dedication

First of all I am greatly indebted to Prof. Leitsch for his continuous support and patience during the last year. He has the rare ability to plant ideas in one's mind which grow to many insights over time, such that in the end those things seemed as simple to me as they were to him all the time.

Many thanks also go to Tsvetan Dunchev, Tomer Libal, Mikheil Rukhaia and Daniel Weller, the PhD students who kindly shared their room with me and listened to my thoughts while giving me input on them.

Due to the recommendations from my friends Mani Esmaeili, Stoiko Ivanov and Tamàs Schmidt, my talks and the poster turned out to be drastically less confusing. Their emotional support was invaluable when I was once again frustrated over some details of a proof not working out.

At last, my gratitude goes to my sister Ingrid and to my parents Christine and Alfons Riener, who supported me during the last years – financially as well as mentally. Thanks that you've been there for me.

Contents

CHAPTER 1

Introduction

Logic is like the sword – those who appeal to it shall perish by it.
Samuel Butler

The original application of reductive cut-elimination was in the proof of Gentzen's Hauptsatz where it was used to show that first order predicate logic is complete [Gen35]. It also has close relations to Craig's Interpolation Theorem and Herbrand's Theorem [Bus98, TS00]. In the context of analyzing mathematical proofs, a faster cut-elimination method named CERES (Cut-Elimination by Resolution) [BL00] was developed which in contrast to the local proof rewriting method applied during reductive cut-elimination uses the resolution principle to remove cuts. The characteristic clause set extracted during the proof transformation provides additional insight into the mathematical arguments used in the proof [BHL+08]. Nonetheless parts of the theory like the equational laws of associativity and commutativity needed for a proof are applied in a - for humans - quite intuitive way. In order to hide these steps from both the proof itself and the characteristic clause set, the aim of this thesis is to formulate CERES for a sequent calculus with an integrated equational theory. The framework of deduction modulo [DHK03] provides such a sequent calculus, also has a cut-elimination theorem and a corresponding natural deduction and resolution calculus. Since some of the existing proofs analyzed with CERES use associative-commutative operations which have a unit element, the theory of commutative monoids is used for experiments.

In chapters two and three we will present the notion of equational theories in general and that of associative-commutative ones with and without unit element. Chapter four then gives the notions of deduction modulo, whereas the central part formalizing CERES modulo equational theories is contained in chapter five. The remaining chapters six and seven provide a discussion of the results. In the appendix there is also an overview of notations used during the text.

CHAPTER 2

Equational Unification

2.1 Overview

As regards the proposition that three is equal to two and one, which you adduce, Sir, as an example of intuitive knowledge, my comment is that it is simply the definition of the term three; for the simplest definitions of numbers are formed in this manner- two is one and one, three is two and one, four is three and one, and so on.
Gottfried W Leibniz

Equational unification is a generalization of syntactic unification (first explained for the Resolution method [Rob65], an exhaustive presentation can be found in [Kni89, BS01]), which searches for a substitution to make two terms syntactically equal. Here, terms need to be equal modulo a set of equations on terms. Two equivalent characterizations of equality modulo theories are common, one defining a semantics which requires equal terms to evaluate to the same element in the domain and a proof system which uses instantiation and substitution of equals as rules.

2.2 Definition over Universal Algebras

Syntactically, terms are constructed over a signature and a set of variables. The signature Σ contains function symbols (usually denoted by $f, g, \ldots$) with their respective arity; nullary function symbols are also called constants (often named $c, d, \ldots$). Variables (usually denoted by $u, v, \ldots$) are held in a separate set V. The actual terms $T(\Sigma, V)$ are defined inductively: variables and constants are terms; if $t_1, \ldots, t_n$ are terms and f is a function symbol of arity n, then $f(t_1, \ldots, t_n)$ is a term.

For the replacement of variables by terms, we introduce the notion of substitution.

Definition 2.2.1. A substitution $\sigma : V \mapsto T(\Sigma, V)$ is a mapping from variables to terms, where only finitely many variables are not mapped to themselves. Applying σ to a term t, denoted by $t\sigma$ is done recursively:

$$t\sigma = \begin{cases} \sigma(x) & \text{if } t = x \\ c & \text{if } t = c \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The empty substitution i.e. the identity function is denoted by ε.

Application of substitution is written in postfix notation. We also note that substitutions are not commutative.

2.3 Semantics

The semantics given here are in the spirit of the one defined in [BN98], although the ones given in [Pla93, BS81] are very similar. A Σ-algebra $\mathcal{A}$ is a structure $\langle A, \nu \rangle$ containing the carrier set A and an evaluation function ν mapping each function symbol $f \in \Sigma$ of arity n to a function $f^{\mathcal{A}}$ from $A^n \mapsto A$. This alone is not enough to evaluate a function, but similar to predicate logic, given a variable assignment $\psi : V \mapsto A$, terms can be evaluated recursively. In the following definitions no particular assignment is mentioned, instead only propositions over the equality of functions are made.

There are some simple means of creating new algebras from a given one:

Definition 2.3.1 (Subalgebra)**.** Let $\langle A, \nu \rangle$ be a Σ-algebra. Then the structure $\langle B, \nu \rangle$ with $B \subseteq A$ is a *subalgebra* (over the signature Σ) iff for all $f \in \Sigma$, $f^{\mathcal{A}} = f^{\mathcal{B}}$.

Definition 2.3.2 (Homomorphism)**.** Let $\mathcal{A}$ and $\mathcal{B}$ be two Σ-algebras with carrier sets A and B and let ϕ be function from $A \mapsto B$. Then ϕ is a *homomorphism* iff it is compatible with all functions $f \in \Sigma$, i.e. $\phi(f^{\mathcal{A}}(a_1, \ldots, a_n)) = f^{\mathcal{B}}(\phi(a_1), \ldots, \phi(a_n))$.

Definition 2.3.3 (Direct Product)**.** If $\mathcal{A}_1, \ldots, \mathcal{A}_k$ are Σ-algebras with their respective carrier sets $A_1, \ldots, A_k$, then the algebra $\mathcal{P}$ with $P = A_1 \times \ldots \times A_k$ is their *direct product*, where each function symbol f of arity n is interpreted component wise, that is

$$\pi_i(f^{\mathcal{P}}(p_1, \ldots, p_n)) = f^{\mathcal{A}_i}(\pi_1(p_1), \ldots, \pi_n(p_n))$$

where π_i denotes the projection function returning the i-th element of an n-tuple.

Now we can also describe a model of an equational theory by means of homomorphisms:

Definition 2.3.4 (Models of equations)**.** The equation $s \approx t$ *holds* in the Σ-algebra $\mathcal{A}$ with carrier set A, in symbols $\mathcal{A} \models s \approx t$ iff for all homomorphisms $\phi : \mathcal{T}(\Sigma, V) \mapsto A$ from the terms to $\mathcal{A}$, both terms are mapped to the same function i.e. $\phi(s) = \phi(t)$.

If E is a set of equations, then the Σ-algebra $\mathcal{A}$ is a *model* of E iff for each equation $e \in E$, $\mathcal{A} \models e$. The class of all models of E is called the Σ-*variety* of E, in symbols $\mathcal{V}(E)$.

Definition 2.3.5 (Semantic consequence)**.** The equality $s \approx t$ is called a *semantic consequence* of a set of equations E ($E \models s \approx t$) iff for all models $\mathcal{M} \in \mathcal{V}(E)$ the equality $s \approx t$ holds in $\mathcal{M}$.

The equality relation $\approx_E$ induced by the equational theory E is defined such that for terms $s, t \in \mathcal{T}(\Sigma, V)$, the relation $s \approx_E t$ holds iff the equation is a semantic consequence of the theory i.e. $E \models s \approx t$.

Varieties are closed under the operations described above. They also give rise to the syntactic rules of replacement and substitution.

Theorem 2.3.6 (Birkhoff's Theorem [Bir35])**.** Let E be a set of equations. Then the class of varieties $\mathcal{V}(E)$ is closed under the creation of subalgebras and direct products as well as under homomorphisms.

2.4 Proof Systems

There are also proof systems for equational logic, for instance the ones in [Bir35, Lei94, Pla93, BN98, GS93]. The rules given in figure 2.1 are taken from [Lei94, BN98] but have Leibniz' rule of substitution of equals by equals instead of the replacement rule: We have two introduction rules, the first one realizing the axiom of reflexivity, the other one allowing equalities from a given set E as axioms. Apart from the mentioned Leibniz rule, the instantiation rule applies a substitution to an equality. Since Leibniz' rule is a generalization of replacement, the rules of symmetry and transitivity from [Lei94, BN98] are derivable (see figure 2.2). Now a set of equations E proves the equality of the terms s and t, denoted by $\vdash_E s \approx t$, if there is a sequence of rule applications deriving $s \approx t$ from the axioms.

$$\frac{s = t \in E}{\vdash_E s = t}$$

Equational Axiom

$$\frac{}{\vdash_E t = t}$$

Reflexivity

$$\frac{\vdash_E s = t}{\vdash_E s\theta = t\theta}$$

Instantiation

$$\frac{\vdash_E T[u] \qquad \vdash_E u = v}{\vdash_E T[v]}$$

Leibniz

Figure 2.1: Rules of equational inference

Some commonly used equational axioms are associativity (A), commutativity (C), distributivity (D), idempotence (I) and the existence of a unit (U) or an inverse ($^{-1}$) element. Also a combination of these axioms is often desired, since for instance $A \cup U$ describes a monoid. For easier notation, the set union symbol is dropped, so ACU^{-1} stands for the theory of abelian groups. In case of multiple function symbols, the relevant symbol is added as a subscript.

As an example of a derivation, in the theory AC, the equation $f(a, f(b, c)) \approx_{AC} f(f(b, a), c)$ has the proof given in figure 2.4.

$$\dfrac{\vdash_E s = s \qquad \vdash_E s = t}{\vdash_E t = s}\ \mathit{Leibniz}$$

Reflexivity

$$\dfrac{\vdash_E t = u \qquad \vdash_E u = v}{\vdash_E t = v}\ \mathit{Leibniz}$$

Transitivity

$$\dfrac{\dfrac{\dfrac{\vdash_E f(s_1,\ldots,s_n) = f(s_1,\ldots,s_n) \qquad \vdash_E s_1 = t_1}{\vdash_E f(s_1,\ldots,s_n) = f(t_1,s_2\ldots,s_n)}\ \mathit{Leibniz} \qquad \vdash_E s_i = t_i}{\vdash_E f(s_1,\ldots,s_n) = f(t_1,\ldots,t_i,s_{i+1},s_n)} \qquad \vdash_E s_n = t_n}{\vdash_E f(s_1,\ldots,s_n) = f(t_1,\ldots,t_n)}\ \mathit{Leibniz}$$

Replacement

Figure 2.2: Derivation of Reflexivity and Transitivity in Equational Reasoning

$\emptyset$	=	$\{\}$	I_f	=	$\{f(x,x) \approx x\}$
A_f	=	$\{f(x,(y,z)) \approx f(f(x,y),z)\}$	$D_{f,g}$	=	$\{f(g(x,y),z) \approx g(f(x,z),f(y,z))\}$
C_f	=	$\{f(x,y) \approx f(y,x)\}$	U_f	=	$\{f(x,e) \approx x\}$
${}^{-1}{}_f$	=	$\{f(x,x^{-1}) \approx e\}$			

Figure 2.3: The equational axioms for Associativity, Commutativity, Idempotence, Distributivity, Unit Element and Inverse Element.

$$\dfrac{\dfrac{AC \vdash_E f(a,f(b,c)) = f(a,f(b,c)) \qquad \dfrac{AC \vdash_E f(x,f(y,z)) = f(f(x,y),z))}{AC \vdash_E f(a,f(b,c)) = f(f(a,b),c))}\ \mathit{Inst}}{AC \vdash_E f(a,f(b,c)) = f(f(a,b),c))}\ \mathit{Leibniz} \qquad \dfrac{AC \vdash_E f(x,y) = f(y,x)}{AC \vdash_E f(a,b) = f(b,a)}\ \mathit{Inst}}{AC \vdash_E f(a,f(b,c)) = f(f(b,a),c))}\ \mathit{Leibniz}$$

Figure 2.4: Proof of $f(a,f(b,c)) = f(f(b,a),c)$

In [Bir35], also the equivalence of semantic models of equations and their derivability via reflexivity, symmetry, transitivity, replacement and substitution is shown.

Theorem 2.4.1 (Completeness of equational inference [Bir35])**.** If E is an equational theory and $s,t \in \mathcal{T}(\Sigma,V)$ then $s \approx_E t$ iff it is derivable i.e. $\vdash_E s \approx t$.

Also equalities are easily expressible in predicate logic with equality, where the models of the formula $\forall\overline{x_1}(s_1 = t_1) \wedge \ldots \wedge \forall\overline{x_n}(s_n = t_n)$ for the equations $s_i \approx t_i \in E$ with x_i being the free variables of s_i and t_i correspond to the models of the equation $s \approx_E t$ which only have the additional fixed definition of the equality predicate.

2.5 Unification

There are two decision problems for equations we often study: the word problem and the unification problem. The former is the question if, given an equation $s \overset{?}{=} t$, all substitution instances are equal modulo the theory E i.e. $s\sigma \approx_E t\sigma$ for all substitutions σ. This can be solved by the methods presented above.

The latter asks whether for the set of equations $\{s_i \overset{?}{=} t_i | 1 \leqslant n\}$ there exists a substitution σ such that $s_i\sigma \approx_E t_i\sigma$. The generalization to sets is necessary, since even in the empty theory, every equation in $\{x \overset{?}{=} a, x \overset{?}{=} b\}$ has a unifier, but the whole set does not[1]. For unification problems, we are also interested in the set of all possible substitutions, not only in unifiability. A shorter name for unification modulo the theory E is E-unification.

Now we are ready to define the unifier of two terms. Since some unifiers are contained in others, it is also of advantage to introduce a notion of generality and only look at the most general unifiers.

Definition 2.5.1 (Unifier)**.** If $P = \{s_i \overset{?}{=} t_i | 1 \leqslant i \leqslant n\}$ is a unification problem and σ is a substitution such that $s_i\sigma \approx_E t_i\sigma$ for $1 \leqslant i \leqslant n$, then σ is called a unifier of P.

Definition 2.5.2 (Generality [BN98, p. 255])**.** A substitution σ is more general than the substitution τ modulo the equational theory E on a set X, written $\sigma \lessdot_E^X \tau$, if there exists a substitution λ such that $x\sigma \approx_E x\tau\lambda$ for all variables $x \in X$.

The restriction to a set X of variables is introduced, because usually only the variables in the unification problem at hand matter. If we omit X we thus assume X to contain only those variables.

The generality relation has the properties of reflexivity and transitivity and is therefore a quasi-order. Of special note is that it allows incomparable elements to be present. In the theory of commutativity, for instance, the equation $f(x, y) \overset{?}{=} f(a, b)$ has the two unifiers $\sigma_1 = \{x \hookleftarrow a, y \hookleftarrow b\}$ and $\sigma_2 = \{x \hookleftarrow b, y \hookleftarrow a\}$, but neither $\sigma_1 \lessdot_C \sigma_2$ nor $\sigma_2 \lessdot_C \sigma_1$.

Definition 2.5.3 (minimal complete set of unifiers [BN98, p. 256])**.** Given an E-unification problem P, a set of substitution $\mathcal{C}$ is *complete*, if every $\sigma \in \mathcal{C}$ is an E-unifier of P and for each E-unifier τ of P there exists a unifier $\sigma \in \mathcal{C}$ such that $\sigma \lessdot_E \tau$. A complete set $\mathcal{C}$ of unifiers is *minimal*, if for all $\sigma, \tau \in \mathcal{C}$ where $\sigma \lessdot_E \tau$ holds, also $\sigma = \tau$. If a minimal complete set of unifiers contains only one element σ, then σ is called *most general unifier*[2].

Sometimes we also need a more fine-grained classification of equational theories. For this, we can restrict the signature of the terms to be elementary, allow constants or look at general terms. Also the cardinality of the minimal complete set of most general unifiers is a measure for the complexity of the unification problem.

Definition 2.5.4.

[1]This cannot happen for word problems because they are universally quantified.

[2]Sometimes, the minimal complete set of unifiers is also called the set of most general unifiers.

- An *elementary* E-unification problem is only defined for terms containing function symbols and constants from the signature of E.
- An E-unification with *constants* is only defined for terms over the signature of E and additional constant symbols.
- A *general* E-unification problem does not restrict the signature of the terms.

Definition 2.5.5. An E-unification is of type *zero*, if there is no minimal complete set of unifiers (in general). It is of *unitary* type, if there is at most one most general unifier. The type is *finitary* if the cardinality of the complete minimal set of unifiers is finite and *infinitary* if the cardinality is infinite.

For its use in resolution based theorem proving, unitary unification is the most desirable one because the only choice during the application of the resolution rule is that of the clauses and the complementary literals to resolve over. Luckily syntactic unification ($\emptyset$-unification), is in this category.

The generalization of the resolution method to equational resolution [Plo72] nondeterministically chooses an E-unifier of complementary literals during a resolution step. In an implementation this is often approached by backtracking or lazy unification [God90], but both add complexity to the search for a refutation. Since C, AC and ACU are finitary for the general unification problem, they create only a bounded number of choice points, whereas infinitary theories like A[3] need some finite representation for the set, which further increases complexity.

For theories of type zero the notion of generality is no improvement to looking at all unifiers. This is so because if there is no minimal complete set of unifiers, then for every unifier σ, there is a more general unifier τ such that $\tau <_E \sigma$. An example for this is the equation $f(x, f(y, x)) \approx_{AI} f(x, f(z, x))$ in the theory AI.

A summary of the example theories can be found in figure 2.5, a far more detailed discussion also giving results for the elementary and constant case can be found in [BN98, BS01].

Unification type	Theory
zero	AI
unitary	$\emptyset$
finitary	C, AC, ACU
infinitary	A

Figure 2.5: Examples of E-Unification types for the general unification problem

[3]Even very simple equations have no finite minimal complete set of unifiers in A. In an example from [Plo72] the equation $f(a, x) \approx_A f(x, a)$ has the most general unifiers $\sigma_n = \{x \leftarrow a^n\}$ with $n \geqslant 1$, but since each instance of the equation is ground, no unifier is obsolete by generality.

CHAPTER **3**

AC and ACU Unification

3.1 Overview

We toast the Lisp programmer who pens his thoughts within nests of parentheses.
Alan Perlis

Many equational theories in mathematics are associative and commutative; some examples are addition and multiplication over the natural numbers, set union and intersection, logical disjunction and conjunction or the use as a representation of multisets. Before the name ACU became widely used, it was also described as AC1 unification and unification over commutative monoids. Addition and multiplication over the positive natural numbers $\mathbb{N}^+$ is an example of an AC theory without unit. Interestingly, equalities over ACU are easier to solve than those over AC, but many algorithms are suitable for generating a complete minimal set of unifiers for both theories. For our purpose we want to replace syntactic derivations (of which one of the main approaches is narrowing [BS01, Rub99, Rus10]) by a semantic approach, so we will concentrate on the latter.

In the following chapters, f will denote the associative-commutative function symbol and e the unit element.

3.2 Deciding the word problem for AC and ACU

The intuitive interpretation of associativity is that the nesting of f is not important. Commutativity means that the order of the terms contained in a structure of fs does not matter. So basically, an f-term can be seen as a multiset of its arguments. To compare the words s and t modulo associativity, rewriting modulo the rule

$$f(t_1, \dots, f(s_1, \dots, s_m), \dots, t_n) \longrightarrow f(t_1, \dots, s_1, \dots, s_m, \dots, t_n)$$

allows to flatten s and t, such that their normal forms s' and t' only need to be checked for syntactic equivalence [LC89]. To take commutativity into account, we can choose a total ordering $\lhd$ on terms and sort the arguments of f in s' and t' such that in the resulting terms $f(s_1, \dots, s_n)$ and $f(t_1, \dots, t_m)$, the arguments are ascending with regard to $\lhd$, i.e. $s_i \lhd s_j$ and $t_k \lhd t_l$ for $1 \leqslant i < j \leqslant n$ and $1 \leqslant k < l \leqslant m$. A possible choice for $\lhd$ is a lexicographic path ordering induced by the alphabetic ordering of the logical symbols, which is defined for instance in [DJ90]. The unit element can be handled by removing every occurrence of e from the arguments after sorting.

It is also often useful to identify a (flattened) term by the multiset of its arguments, i.e. $f(t_1, \ldots, t_m)$ will be written as $\{t_1^{c_1}, \ldots, t_n^{c_n}\}$, where the c_i denote the multiplicity if t_i in the set. If it is unambiguous, the terms can be concatenated as is commonly used in formal languages, so the former term will be written as $t_1^{c_1} \ldots t_n^{c_n}$.

3.3 Diophantine Equations

Since semantic AC- and ACU-unification solves linear diophantine equations as a subproblem, we need to make a small intermission. Equations of polynomials over the integers were already studied by Diophantus of Alexandria [Dio52] after whom they are named. A famous (general) diophantine equation is $a^n + b^n = c^n$, also known as Fermat's last theorem, which he conjectured to be unsolvable for different a,b and c when $n > 2$ but has been completely proven in the last twentieth century [Wei]. Arbitrary diophantine equations also found a direct application to computability theory in the form of Hilbert's tenth problem [Hil00], which is the question for a recursive algorithm to decide whether a diophantine equation is solvable. Although the general problem was proven undecidable [Rob52, Dav53, DPR61, Mat70], its restriction to linear equations is decidable. Many notions from linear algebra over the reals carry over to equations over integers, but in contrast to $\langle \mathbb{R}, +, \cdot \rangle$, the algebra $\langle \mathbb{Z}, +, \cdot \rangle$ is only a ring, not a field. Solutions can still be expressed as a linear combination of basis vectors, but the number of base vectors may be of exponential size in the number of variables [Lan89, Dom92]. Since the Gaussian algorithm for the computation of a basis is not applicable, other algorithms had to be developed [Lan89, CF89, For87, Con93, Dom91b].

In the following, we will make the basic definitions more precise keeping the presentation close to [CF89]. Then we give an algorithm computing the basis of a linear diophantine equation and also one for systems of equations.

Definition 3.3.1 (Linear Diophantine Equation)**.** A *linear diophantine equation* is of the form

$$\sum_{i=1}^{n} c_i x_i = h$$

where $c_i \in \mathbb{Z}$ are constants and x_i are variables over $\mathbb{Z}$. An equations with $h = 0$ is called *homogeneous* and *inhomogeneous* otherwise. A *solution* is a variable assignment for all x_i such that the equation holds.

Diophantine equations can be elegantly written using the inner vector product. In our case, the original equation has a left-hand side and a right-hand side with positive coefficients only, so often positive and negative coefficients are written separately. Using the notation of $(\underline{\xi}, \underline{\eta})$ for the concatenation of $\underline{\xi}$ and $\underline{\eta}$, a solution is then a vector $(\underline{\xi}, \underline{\eta})$ such that $\underline{a}\,\underline{\xi} - \underline{b}\,\underline{\eta} = h$. Also a linear combination can be easily written as $\sum_{i=i}^{n} s_i\, \underline{v}_i$.

As we are only interested in studying non-negative solutions, for the equation $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = h$, we denote this set by $S(\underline{a}, \underline{b}, \underline{h})$ and write $S(\underline{a}, \underline{b})$ in the homogeneous case.

Definition 3.3.2 (Solution)**.** Let $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = h$ be a linear diophantine equation with $\underline{a} > 0$ and $\underline{b} > 0$. Then its set of (non-negative) solutions is defined as

$$S(\underline{a}, \underline{b}, h) = \{(\xi, \eta) | \underline{a}\,\underline{\xi} - \underline{b}\,\underline{\eta} = h, (\xi, \eta) > 0\}$$

For homogeneous equations we define

$$S(\underline{a}, \underline{b}) = \{(\xi, \eta) | \underline{a}\,\underline{\xi} - \underline{b}\,\underline{\eta} = 0, (\xi, \eta) > 0\}$$

By the lack of an additive and multiplicative inverse, the solutions only form a monoid which is finitely generated from the minimal solutions with respect to pointwise ordering. [CF89]

Now we will formally define the basis of a diophantine equation and look at the minimal solutions with regard to comparing vectors component-wise to see that they form a basis. Indeed, many algorithms [CF89, Lan89] compute these minimal solutions to find a basis.

Definition 3.3.3 (Basis)**.** A *basis* of a homogeneous diophantine equation is a smallest finite set[1] $\{\beta_1, \ldots, \beta_n\}$ such that for every solution $\underline{s} \in S(\underline{a}, \underline{b})$ it is composable of a linear combination of basis vectors i.e. for each s there exists $\lambda_1, \ldots, \lambda_n$ such that $\underline{s} = \sum_{i=1}^{n} \lambda_i\, \underline{\beta}_i$.

Definition 3.3.4 (Pointwise ordering of vectors [CF89])**.** Let $(\underline{\xi}, \underline{\eta})$ and $(\underline{\xi}', \underline{\eta}')$ be two vectors with $|\underline{\xi}| = |\underline{\xi}'| = n$ and $|\underline{\eta}| = |\underline{\eta}'| = m$ and let $\leqslant$ be the usual ordering on natural numbers. Then $\leqslant$ can be extended to a pointwise partial ordering on vectors:

$$(\underline{\xi}, \underline{\eta}) \leqslant (\underline{\xi}', \underline{\eta}') \text{ iff } \underline{\xi}_i \leqslant \underline{\xi}'_i \text{ and } \underline{\eta}_j \leqslant \underline{\eta}'_j \text{ for all } i \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, m\}$$

this ordering can be made strict by defining:

$$(\underline{\xi}, \underline{\eta}) < (\underline{\xi}', \underline{\eta}') \text{ iff } (\underline{\xi}, \underline{\eta}) \leqslant (\underline{\xi}', \underline{\eta}') \text{ and } \underline{\xi}_i < \underline{\xi}'_i \text{ or } \underline{\eta}_j < \underline{\eta}'_j \text{ for some indices } i \text{ and } j.$$

Definition 3.3.5 (Minimal Solutions [CF89])**.** The set of *minimal solutions* of an equation $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = h$ is defined as the subset of $\leqslant$ minimal elements of $S(\underline{a}, \underline{b}, h) \backslash \{\underline{0}\}$ as

$$M(\underline{a}, \underline{b}, h) := \{\underline{x} \in S(\underline{a}, \underline{b}, h) \backslash \{\underline{0}\} | \text{ there exists no } y \in S(\underline{a}, \underline{b}, h) \backslash \{\underline{0}\} \text{ s.t. } y \leqslant x\}$$

and for a homogeneous equation as

$$M(\underline{a}, \underline{b}) := \{\underline{x} \in S(\underline{a}, \underline{b}) \backslash \{\underline{0}\} | \text{ there exists no } y \in S(\underline{a}, \underline{b}) \backslash \{\underline{0}\} \text{ s.t. } y \leqslant x\}$$

[1] The basis is not necessarily unique.

Lemma 3.3.6.
The set $M(\underline{a}, \underline{b})$ is a basis for the homogeneous linear diophantine equation $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = 0$.
The set of $\{m + s | m \in M(\underline{a}, \underline{b}, h), s \in S(\underline{a}, \underline{b})\}$ is a basis of the inhomogeneous linear diophantine equation $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = h$.

Proof. given in [CF89] by showing that

1. $M(\underline{a}, \underline{b})$ and $M(\underline{a}, \underline{b}, h)$ are finite
2. $S(\underline{a}, \underline{b})$ is the set of linear combinations of $M(\underline{a}, \underline{b})$
3. $S(\underline{a}, \underline{b}, \underline{h}) = \{m + s | m \in M(\underline{a}, \underline{b}, h), s \in S(\underline{a}, \underline{b})\}$

□

One of the optimizations of [CF89] is the observation that variables with the same coefficient can be grouped and instead of the sum of the single variables a fresh one is used.

Definition 3.3.7 (Injective companion)**.** Given an equation

$$\sum_{i \in I} a_i \cdot (\sum_{l=1}^{\alpha(a_i)} x_{i,l}) - \sum_{j \in J} b_j \cdot (\sum_{k=1}^{\beta(b_j)} y_{j,k}) = 0$$

where the functions $\alpha : I \mapsto \mathbb{N}^+$ and $\beta : J \mapsto \mathbb{N}^+$ return the number of variables grouped for each element of an appropriate index set $I = \{1, \ldots, n\}$ and $J = \{1, \ldots, m\}$, define its *injective companion* as

$$\sum_{i \in I} a_i \cdot x_i - \sum_{j \in J} b_j \cdot y_j = c$$

.

The function ν mapping $(x_{i,l}, y_{j,k})$ to $(\sum_{l=1}^{\alpha(a_i)} x_{i,l}, \sum_{k=1}^{\beta(b_j)} y_{j,k})$ for $i \in I, j \in J$ then expresses the relationship from the variables of the equation to that of the injective companion. Given the sum, the values of the original variables can be easily enumerated. Since ν allows a direct translation from the original equation to the injective companion and back, the set of solutions and also the set of minimal solutions of the equation can be calculated from the companion.

Lankford's Algorithm

In [Lan89], an easily implementable algorithm was presented. Given an equation $\underline{a}\,\underline{x} - \underline{b}\,\underline{y} = 0$ with $|\underline{a}| = n$ and $|\underline{b}| = m$, it starts with $(\underline{x}, \underline{y}) = (\underline{0}, \underline{0})$ and systematically increasing each component as long as it decreases the norm of the equation. If there is no possibility to increase a component without decreasing the norm, the algorithm halts.

Definition 3.3.8 (Norm of a vector)**.** Let $\underline{v} = (\underline{\xi}, \underline{\eta})$ be a vector. Then we define its norm as

$$\|\underline{v}\|_{\underline{a}} = \underline{a}\,\underline{\xi} - \underline{b}\,\eta$$

.

Taking the identity matrix I^{n+m}, we assign the first n vectors to the set A and the remaining m vectors to the set B.

We now inductively define the sets X,P,N and Z, where X will hold the newly generated vectors, P and N will hold the vectors with positive and negative norm which are irreducible regarding to the already found basis vectors Z. The initial values are:

$$\begin{aligned} X_1 &= \emptyset & P_1 &= A \\ Z_1 &= \emptyset & N_1 &= B \end{aligned}$$

and given the sets of step k we compute those of step $k+1$ as follows:

$$\begin{aligned} X_{k+1} &= (A + N_k) \cup (B + P_k) \\ P_{k+1} &= \{\underline{s} | \underline{s} \in X_{k+1}, \|\underline{s}\| > 0 \text{ and } \underline{s} \text{ irreducible relative to } Z_k\} \\ N_{k+1} &= \{\underline{s} | \underline{s} \in X_{k+1}, \|\underline{s}\| < 0 \text{ and } \underline{s} \text{ irreducible relative to } Z_k\} \\ Z_{k+1} &= Z_k \cup \{\underline{s} | \underline{s} \in X_{k+1}, \|\underline{s}\| = 0\} \end{aligned}$$

A vector $\underline{s}$ is reducible with regard to Z_k if and only if there exists a vector $\underline{z} \in Z_k$ such that $\underline{z} < \underline{s}$. The addition of sets is defined as $A + B = \{a + b | a \in A, b \in B\}$.

The algorithm always terminates with the halting condition that both P_k and N_k are empty. Then the set of minimal vectors with norm 0 i.e. $Z_k = M(\underline{a}, \underline{b})$ is the basis we were searching for.

As an example, take the equation $2x_1 + x_2 - 2y_1 = 0$. To refer to the intermediate results more easily, we name the vectors $\underline{v_1}$ to $\underline{v_7}$ and give their norm and value in figure 3.1. We start by setting $A = P_1 = \{\underline{v_1}, \underline{v_2}\}$, $B = N_1 = \{\underline{v_3}\}$ and $X_1 = Z_1 = \emptyset$. In step 2, we get $X_2 = \{\underline{v_4}, \underline{v_5}\} \cup \{\underline{v_4}, \underline{v_5}\} = \{\underline{v_4}, \underline{v_5}\}$. Since no vector in X_2 has positive norm, $P_2 = \emptyset$ and because $\underline{v_5}$ is irreducible respective to Z_1, $N_2 = \{\underline{v_5}\}$. The vector $\underline{v_4}$ has norm zero and thus $Z_2 = \{\underline{v_4}\}$. In the final step, $X_3 = \emptyset \cup \{\underline{v_6}, \underline{v_7}\}$. Since $\underline{v_4} \leqslant \underline{v_6}$, $\underline{v_6}$ is reducible reducible to Z_2, which means that $P_3 = N_3 = \emptyset$. We get $Z_3 = Z_2 \cup \{\underline{v_7}\}$ and have now calculated $M(\underline{a}, \underline{b}) = \{(1,0,1), (0,2,1)\}$.

step 1			step 2			step 3		
name	$\| \|$	vector	name	$\| \|$	vector	name	$\| \|$	vector
$\underline{v_1}$	2	$(1,0,0)$	$\underline{v_4}$	0	$(1,0,1)$	$\underline{v_6}$	1	$(1,1,1)$
$\underline{v_2}$	1	$(0,1,0)$	$\underline{v_5}$	-1	$(0,1,1)$	$\underline{v_7}$	0	$(0,2,1)$
$\underline{v_3}$	-2	$(0,0,1)$						

Figure 3.1: Overview of intermediary vectors in the calculation of the basis of $2x_1 + x_2 - 2y_1 = 0$

Systems of linear diophantine equations

In [BCD90] an algorithm for systems is given. Similar to [Lan89] and [CF89], the possible solutions are enumerated by assigning each coefficient a distinct line of the identity matrix and constructing candidates by adding one of the identity vectors to candidates from the previous step. In the case of one equation a newly generated candidate is only accepted if its norm is decreasing. Given k systems, we also have a k-tuple of norms, so it is not directly clear what decreasing means in this context. But when the vector is interpreted geometrically, a natural measure emerges: given the system of equations $\{\underline{a_1} \cdot \underline{x} = 0, \ldots, \underline{a_k} \cdot \underline{x} = 0\}$ then let $\underline{x}$ be the vector to be modified, $\underline{e_j}$ be the line of the identity matrix with $\pi_j(x) = 1$ and let $d(\underline{x})$ be the vector of norms $(\|\underline{x}\|_{\underline{a_1}}, \ldots, \|\underline{x}\|_{\underline{a_k}})$. The vector $\underline{x} + \underline{e_j}$ is only kept, if its vector of norms $d(\underline{x} + \underline{e_j})$ is contained in the halfspace that contains the origin and which is delimited by the affine hyperplane orthogonal to $d(\underline{x})$ and containing the extremity of $d(\underline{x})$. This condition can be captured as $d(\underline{x} \cdot \underline{e_j}) < 0$.

Now we can again inductively compute the sets of candidates:

$$\begin{aligned}
P_1 &= \{e_j | 1 \leqslant j \leqslant k\} \\
P_n + 1 &= \{\underline{v} + \underline{e} | \underline{v} \in Q_n, d(\underline{v}) \cdot d(\underline{e_j}) < 0\} \text{ for } n \geqslant 1 \\
B_n &= \{\underline{v} \in P_n | d(\underline{v}) = 0\} \\
Q_n &= \{\underline{v} \in P_n \backslash B_n | \forall \underline{s} \in \textstyle\bigcup_{l<n} B_l, \underline{v} > s\}
\end{aligned}$$

The algorithm stops when $P_n = \emptyset$ and returns $\bigcup_{l<n} B_l$ as the basis.

The calculation can be seen as a dag, where edges go from each vertex representing $\underline{v}$ to the one representing $\underline{v} + e_j$. If a vector $\underline{v}$ has distance n to an identity vector, it will also be contained in P_n. Then two distinct paths leading to the same element occur only due to the fact that $\underline{v} + e_i + e_j = \underline{v} + e_j + e_i$. We can easily exclude one of these versions to create a forest instead of a dag: let $\underline{v} = \sum_{i \in I} e_i$ be a vector in the graph, then only look at those vectors $\underline{v} + e_j$ where $j \geqslant max(I)$. This means, if $\underline{v}$ is increased at position j, then all positions $i < j$ will be fixed from then on.

As an example we calculate the basis of the equation set

x_{11}	x_{12}	x_{21}	x_{22}	x_{31}	x_{32}	x_4		
1	1	0	0	0	0	-1	=	0
0	0	1	1	0	0	-1	=	0
0	0	0	0	1	1	-1	=	0

since there are many common coefficients, we will solve its injective companion

x_1	x_2	x_3	x_4		
1	0	0	-1	=	0
0	1	0	-1	=	0
0	0	1	-1	=	0

with $x_1 = x_{11} + x_{12}$, $x_2 = x_{21} + x_{22}$ and $x_3 = x_{31} + x_{32}$ as constraints.

Figure 3.2 shows the resulting forest, where each node is labeled by the vector $\underline{v}$ and its vector of norms $d(\underline{v})$ and the base node is marked blue.

The resulting basis consists only of the vector $(1, 1, 1, 1)$, but the basis of our original problem is considerably larger:

x_{11}	x_{12}	x_{21}	x_{22}	x_{31}	x_{32}	x_4	#
0	1	0	1	0	1	1	b_1
0	1	0	1	1	0	1	b_2
0	1	1	0	0	1	1	b_3
0	1	1	0	1	0	1	b_4
1	0	0	1	0	1	1	b_5
1	0	0	1	1	0	1	b_6
1	0	1	0	0	1	1	b_7
1	0	1	0	1	0	1	b_8

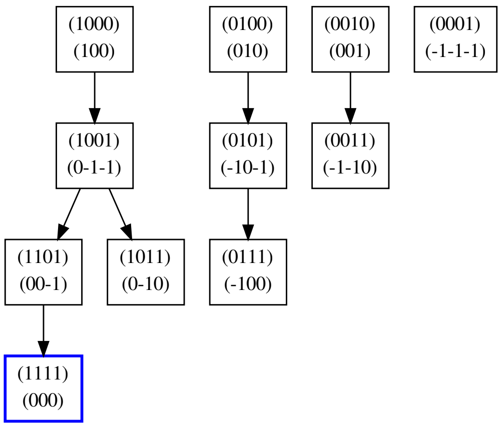


Figure 3.2: Calculation of a basis via the algorithm of Boudet, Contejean and Devie

3.4 Semantic AC and ACU unification

The basic observation leading to semantic unification was that after AC-unification of a single equation, the number of symbols in the terms of its lefthand- and righthand side must be the same, where constant and function terms with a symbol different to f introduce exactly one term (namely themselves) and variables introduce an arbitrary number of terms (in the case of AC at least one). This relationship can be expressed as a linear diophantine equation, where the coefficients represent the number of occurrences of a term and the variable the number of symbols introduced. It is possible to extract a candidate for a unifier from the solution of an equation. Since the number of symbols alone does not sufficiently characterize a unifier, a candidate can only be kept if assignments of multiple terms to the same variable can be unified.

Also an upper bound on the number of introduced symbols can be computed from the equation for a unifier to be part of a complete minimal set of unifiers, which allows to check only finitely many sets.

The method was first explored in [Sti75, Sti81] and in parallel in [LS76], where the main difference is that the first approach uses homogeneous diophantine equations and removes more invalid candidates afterwards, whereas the latter one fixes the count of constants to one and solves an inhomogeneous equation. Termination of the general case was an unsolved problem for some time, until it was proven nine years after the first algorithms were stated [Fag84].

Research afterwards was concerned with making the algorithm more effective, since both the calculation of the basis of a linear diophantine equation and the generation of a unifier from a set of basis vectors is computationally expensive. Although the decision problem is in NP [KN92], calculation of the minimal set of unifiers is at most exponential[2] [Pot91]. A lot of effort went into solving linear diophantine equations more effectively [For87, Lan89]. Some approaches tackled the complexity by incrementally generating unifiers [Con93, LC89] and by reducing some common cases to ones which have a canonical basis, thereby circumventing the need to generate it by an algorithm [LC89].

Much energy was also put into solving systems of equations [AK92, BCD90] which was not only done for performance reasons but also because in contrast to $\emptyset$-unification, a most general unifier of a set of equations can not always be composed of the most general unifiers for each equation in the set. As a simple example take the problem $\{ax \overset{?}{=} by, by \overset{?}{=} cz\}$ which has the most general unifier $\{x \hookleftarrow bc, y \hookleftarrow ac, z \hookleftarrow ab\}$, but the single equation $ax \overset{?}{=} by$ has the most general unifier $\{x \hookleftarrow b, y \hookleftarrow a\}$, in which no variable can be substituted anymore.

Another way to reduce complexity is the restriction to AC-matching [Eke93] i.e. solving only problems, where the left-hand side of each equation is a variable.

Complexity results

A concise summary of the complexity results is given in [BS01]:

Deciding the general unification problem and the one with constants is NP-complete for both AC and ACU [KN92]. The elementary unification problem for ACU has a trivial solution mapping each variable to the unit e and elementary AC unification takes polynomial time [Dom91a].

theory	elementary	constants	general
AC	PTIME	NPTIME	NPTIME
ACU	CONSTANT	NPTIME	NPTIME

Table 3.1: Complexity of the unification problems for AC and ACU

[2]The author could not find a proof that the problem of finding a minimal complete set of unifiers is harder than NP. We know that all algorithms using the basis are exponential since the number of bases may be exponential to the number of coefficients of a problem. Since the unification problem is an existential statement, it is quite possible that finding a minimal solution is in NP but that finding all minimal ones is harder.

theory	elementary	constants	general
AC	ω	ω	ω
ACU	1	ω	ω

Table 3.2: Unification type of AC- and ACU-unification

Stickel's Algorithm

Since many algorithms, including the one implemented for this thesis, extend Stickel's Algorithm, it will be given in more detail. The general unification algorithm uses a specialized variant for the variable-only case, so we define some additional notations and turn our attention the elementary case first:

Definition 3.4.1 (Value of a solution w.r.t. its equation)**.** Let $(\underline{\xi}, \underline{\eta})$ be a solution of the equation $\underline{c}\,\underline{x} - \underline{d}\,\underline{y} = 0$ with $\underline{c} \geqslant \underline{0}$ and $\underline{d} \geqslant \underline{0}$. Then its value is defined as

$$val(\underline{\xi}, \underline{\eta}) = \underline{c}\,\underline{\xi} = \underline{d}\,\underline{\eta}$$

.

Elementary Case

Given a unification problem $s \overset{?}{=} t$ with $s = \{x_1^{c_1}, \ldots, x_n^{c_n}\}$ and $t = \{y_1^{d_1} \ldots y_m^{d_m}\}$, calculate the most general unifiers in the following steps:

1. Translate the terms to a diophantine equation: the multiset notation is already very close to the equation, since we just take each variable times its multiplicity and sum the products for the left-hand and the right-hand side. For each logical variable x we denote the integer variable (counting the symbols introduced by the corresponding logical variable) in the diophantine equation by x', the general form can be given as:

$$x'_1 c_1 + \ldots + x'_k c_k - y'_1 d_1 - \ldots - y'_l d_l = 0$$

   .

2. Remove common occurrences of terms: we can simplify the equation if a variable appears on both sides of the formula by transforming $x'_i c_i - y_j d_j$ to $x_i(c_i - d_j)$ for all indices i, j such that $x_i = y_j$. The paper introduces this as a preprocessing step on the term level because for the general case, different fresh variables are introduced for the same symbol which prevent the simplification of the diophantine equation. It can also happen, that after simplification, there are no more positive or negative coefficients. Then the equation has only the trivial solution assigning zero to all variables, which does not have a corresponding term in AC, but this solution is dropped during the next step.

   The equation after this transformation step has possibly less summands and thus is:

$$x'_1 c_1 + \ldots + x'_n c_n - y'_1 d_1 - \ldots - y'_m d_m = 0$$

.

with $m \leqslant k$ and $n \leqslant l$.

3. Find the basis $\{\underline{\beta_1}, \ldots, \underline{\beta_l}\}$ of the linear diophantine equation and assign each basis vector b_i a fresh logical variable z_i.

4. Calculate the upper bound for the value of a solution which describes a minimal unifier. Let $|\underline{c}| = n$ and $|\underline{d}| = m$, then

$$\mathit{bound}(\underline{c}, \underline{d}) = \mathit{max}(n, m) \cdot \mathit{max}(\mathit{lcm}(c_i, d_j))$$

with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$ is such a bound [Sti81], which is relatively generous and can be refined, but for the purpose of proving the calculation's finiteness, it suffices.

5. Find all non-negative candidate solutions of the diophantine equation by summing all subsets of basis vectors. In the case of AC remove every solution with a component equal to zero since it would require to assign no symbol to a variable; in ACU the unit element will be assigned in step 7.

6. Iterate the list of candidates in ascending value and add only those to the candidates, whose value does not surpass the bound and which are no linear combination of the candidates already selected. To make this more formal, let $\{\underline{v_1}, \ldots, \underline{v_k}\}$ be the sorted set of solutions, then $S_0 = \emptyset$. Given a set S_{i-1} of found solutions, then S_i with $i \leqslant k$ is defined as follows:[3]

$$S_i = \begin{cases} S_{i-1} \cup \{\underline{v_i}\} & \text{if } \mathit{val}(\underline{v_i}) \leqslant \mathit{bound}(\underline{c}, \underline{d}) \text{ and} \\ & \text{there exists no } \lambda_1, \ldots, \lambda_{i-1} \in \mathbb{N} \text{ s.t. } \underline{v_i} = \lambda_1 \underline{v_1} + \ldots + \lambda \underline{v_{i-1}} \\ S_{i-1} & \text{otherwise} \end{cases}$$

Then define the set of checked solutions $S = S_k$.

7. For every checked solution $\underline{s} \in S$ with the linear combination of $\underline{s} = \sum_{i \in I} \underline{\beta_i}$ over the index set $I \subseteq \{1, \ldots, l\}$, generate a new unifier in the following way: substitute x_p by a the term represented by the multiset $\{z_i^{\pi_p(\underline{\beta_i})} | i \in I\}$, with $p \in \{1, \ldots, n\}$ where $\pi_p(\underline{\beta})$ again denotes the projection of a vector to its p^{th} element. In the same way, substitute y_q by a the term represented by $\{z_i^{\pi_q(\underline{\beta_i})} | i \in I\}$, with $p \in \{n+1, \ldots, n+m\}$. In the case of ACU, when the term is represented by the empty set, assign the unit e.

[3]For notational purposes, the left out elements were taken into the linear combination. This makes no difference, since if $\underline{v} = \sum_{i \in I} \lambda_i \underline{u_i}$ for some index set I of non-composable vectors $\underline{u_i}$, and $\underline{w} = \mu \underline{v} + \sum_{j \in J} \lambda_j \underline{u_j}$ for some index set J then it can also be composed without $\underline{v}$ since $\underline{w} = \mu \sum_{i \in I} \lambda_i \underline{u_i} + \sum_{j \in J} \lambda_j \underline{u_j}$. An implementation will certainly only try the non-composable candidates.

8. As a somewhat cosmetical step, we can reuse removed variables: when a unifier σ assigns a variable x of the original term exactly one basis variable z, we then can rename z to x since x is not a variable in the domain of σ. Thus by replacing σ by $\sigma' = \sigma.\{z_i \leftarrow\!\shortmid\, x\}$, the finite representation of the unifier is shorter. As mentioned, the saving is not substantial because for the purpose of resolution, variants of the clauses with fresh variables will be produced before trying to unify them. On the other hand, for a human reader apart from having shorter substitutions, it is much easier to track a variable through a series of resolution steps.

As an example, we solve the unification problem of $f(uvxxxy) \overset{?}{=} f(vvxyy)$. After dropping the common terms, we can solve the simpler problem $f(uxx) \overset{?}{=} f(vy)$. The corresponding diophantine equation is $u' + 2x' - v' - y' = 0$. We calculate the upper bound of values as $max(2,2) \cdot max(1,1,2,2) = 4$. The basis for it is given in table 3.3 and the non-negative solutions in table 3.4.

u'	x'	v'	y'	variable
1	0	1	0	z_1
1	0	0	1	z_2
0	1	1	1	z_3

Table 3.3: Bases of the diophantine equation $u' + 2x' - v' - y' = 0$

u'	x'	v'	y'	$val(\underline{v})$	variables
1	1	2	1	3	$z_1 + z_3$
1	1	1	2	3	$z_2 + z_3$
2	1	2	2	4	$z_1 + z_2 + z_3$

Table 3.4: Positive solutions to the diophantine equation $u' + 2x' - v' - y' = 0$

Since the three solutions are mutually linearly independent, all three of them generate a different mgu:

- $\sigma_1 = \{u \mapsto z_1, x \mapsto z_3, v \mapsto f(z_1 z_3), y \mapsto z_3\}$
  σ_1 can be simplified to $\{v \mapsto f(ux), y \mapsto x\}$

- $\sigma_2 = \{u \mapsto z_2, x \mapsto z_3, v \mapsto z_3, y \mapsto f(z_2 z_3)\}$
  σ_2 can be simplified to $\{v \mapsto x, y \mapsto f(ux)\}$

- $\sigma_3 = \{u \mapsto f(z_1, z_2), x \mapsto z_3, v \mapsto f(z_1 z_3), y \mapsto f(z_2 z_3)\}$
  σ_3 can be simplified to $\{u \mapsto f(z_1, z_2), v \mapsto f(z_1 x), y \mapsto f(z_2 x\}$.

The minimal complete set of unifiers is then $\{\sigma_1, \sigma_2, \sigma_3\}$.

General Case

For the general unification problem, we generalize a given term by replacing every constant and function term starting with a symbol different from f by a fresh variable v_i. Since we later want to specialize our unifiers again, we remember the replacement in the substitution θ, which assigns every variable v_i the term it replaced. In the next step, we then solve the elementary unification problem to get a minimal complete set of unifiers $U_E = \{\sigma_1, \ldots, \sigma_k\}$. Now each substitution $\sigma \in U_E$ may map one of the abstraction variables v_i to a different term than θ does. Only if those terms can be unified, we can generate a most general unifier for the general problem. More formally, from an mgu $\sigma \in U_E$ and the abstraction substitution θ, we merge them to the substitution[4] $\mu_{\sigma,\theta}$:

$$\mu_{\sigma,\theta}(x) = \begin{cases} t & \text{if } \sigma(x) = x, \theta(x) = t \\ t & \text{if } \sigma(x) = t, \theta(x) = x \\ s\tau & \text{if } \sigma(x) = s, \theta(x) = t, s \approx_{AC} t, \exists \tau \text{ s.t. } s\tau = t\tau \\ \text{undefined} & \text{otherwise} \end{cases}$$

Then define the minimal complete set of unifiers $U_G = \{\mu_{\sigma,\theta} | \sigma \in U_E, \mu_{\sigma,\theta} \text{ defined on all variables}\}$. It should be mentioned that $\mu_{\sigma,\tau}$ is not unique because s and t may have multiple unifiers, so it can happen that $|U_G| > |U_E|$. This is the reason why it took some time to prove the termination of the algorithm for the general case, which was accomplished in [Fag84].

3.5 Sets of equations

In [BCD90] there is a rule based approach similar to [MM82] adopted for equational theories. It separates equations over mixed theories into smaller equations over pure ones.

Definition 3.5.1 (Pure and proper equations [BCD90])**.** Let E_i be the equational theories over the terms $T(\Sigma_i, X)$ with indices $i \in I$ and let the combined signature be $\Sigma = \bigcup_{i \in I} \Sigma_i$. Then a term $t \in T(\Sigma, X)$ is *pure* in the theory E_i if it is a term over the signature of E_i i.e. $t \in T(\Sigma_i, X)$. A non-pure term is called *heterogeneous*. An equation $s \overset{?}{=} t$ is *pure* in the theory E_i if s and t are pure in E_i and it is *heterogeneous* if at least one of the terms is heterogeneous. The equation $s \overset{?}{=} t$ is *proper* if either s or t is not a variable.

A problem set of equations

$$P = P_V \cup P_H \cup P_0 \cup \ldots \cup P_n$$

is then divided into subproblems, where
$P_V = \{p | p \in P, p \text{ is not proper}\}$
$P_H = \{p | p \in P, \text{ there is no } E_i \text{ s.t. } p \text{ is homogeneous in } E_i\}$
$P_i \ = \{p | p \in P, p \text{ is homogeneous in } E_i \text{ for some } i \in I\}$

[4]The function μ is only a partial function and thus no real substitution. In the construction of U_G we only keep the total functions, so no harm is done.

The intention is that $\Sigma_0 = \{a, b, c, \ldots, f, g, h, \ldots\}$ contains the theory-free function symbols and that $\Sigma_i = \{+_i\}$ for $i \geqslant 1$ is the signature of AC symbols for different AC theories. Given an algorithm for each theory E_i, this algorithm now combines them to solve problems over heterogeneous equations. For this, the theories are required to be simple:

Definition 3.5.2 (Non-collapsing, regular and simple theory [BCD90])**.**
An equational theory E is *non-collapsing* if for no equation $x \overset{?}{=} t$ with $x \in Var(t)$ and $t \notin X$.

An equational theory E is *regular* if for all equations $s \overset{?}{=} t$ also $Var(s) = Var(t)$ holds.

An equational theory E is *simple* if for all equations $s \overset{?}{=} t$ does not have a solution over E when t is a proper subterm of s.

Definition 3.5.3 (Compound cycle)**.** Let $\{x_1 \overset{?}{=} s_1, \ldots, x_n \overset{?}{=} s_n\}$ be a set of equations with $x_1 \in Var(s_n)$ and $x_i \in Var(s_{i-1})$ for $2 \leqslant i \leqslant n$. It is called a *compound cycle*, if for two indices $k \neq l$ the terms s_k and s_l are pure in different theories.

If a theory is simple, then it is also non-collapsing and regular [BCD90]. If all combined theories are non-collapsing and regular then there are no compound cycles [Yel85, Tid86]. We will later use the fact that there are no compound cycles and that the theory is non-collapsing.

The rules are defined in the form $P \succ Q$ when from the set P we deduce the set Q by one of the rules. If before the set of equations was $S \cup P$ then afterwards it will be $S \cup Q$. Before we present the rules, we need to define the occur-check relation and what a dag-solved form is.

Definition 3.5.4 (Occur-check relation [BCD90])**.** The *occur-check* relation is defined as $x <^{occ} y$ if there exists a chain of equations such that

$$x \overset{?}{=} s_1[x_1], x_1 = s_2[x_2], \ldots, x_n = s_n[y]$$

where at least one of the s_i is not a variable.

Definition 3.5.5 (dag-solved form [BCD90])**.** $Q = \{x_1 \overset{?}{=} t_1, \ldots, x_n \overset{?}{=} t_n\}$ is a *dag-solved form* of a problem set of equations P if P and Q have the same solutions and for all $i, j \in \{1, \ldots, n\}$ the following holds:

1. $i \neq j$ implies $x_i \neq x_j$
2. $x_i <^{occ} x_j$ in P implies $i < j$
3. $t_i \in X$ implies $t_i, x_i \in Var(P)$ and x_i occurs nowhere else in Q
4. $x_i \in Var(P)$ or there exists a $j < i$ s.t. $x_j \in Var(P)$ and $x_j <^{occ} x_i$.

P is *in dag-solved* form, if P is a dag-solved form of P.

Now the rules are the following:

- VA (Variable Abstraction)

$$s \overset{?}{=} t \succ C[x_1, \ldots, x_n] = t, x_1 \overset{?}{=} s_1, \ldots x_n = s_n$$

  if s is heterogeneous and $C[x_1, \ldots, x_n]$ is a maximal pure term such that $s = C[x_1, \ldots, x_n]\{x_1 \hookleftarrow s_1, \ldots x_n \hookleftarrow s_n\}$ and the x_i are fresh variables.

  This is the rule which separates a term into pure theories. The example given is that $f(x, g(y) + g(z+u)) \overset{?}{=} t$ will be transformed to $f(x, v_1) = t, v_i \overset{?}{=} g(y)+g(z+u)$. Then $g(f(g(a, x), x), y) \overset{?}{=} t$ will be decomposed to $g(v_1, y) \overset{?}{=} t, v_1 \overset{?}{=} f(g(a, x), x)$ and subsequently to $g(v_1, y) \overset{?}{=} t, v_1 \overset{?}{=} f(v_2, x), v_2 \overset{?}{=} g(a, x)$.

- E-Res (E-Resolution)

  $P_i \succ Q_i$ if P_i is not in dag-solved form, pure in E_i and $Q = \{x_1 \overset{?}{=} t_1, \ldots, x_n \overset{?}{=} t_n\}$ is a dag solved form of P_i.

  This is the call to the solver in the pure theory. If there is no unifier, the rule fails. If there is more than one unifier we have to look at all possible solutions $Q_i \in unify\_pure(P_i)$ where $unify\_pure$ denotes the function returning the minimal complete set of unifiers for a pure theory. This is a don't know indeterminism which usually makes backtracking necessary.

- Clash

  $s \overset{?}{=} t \succ fail$

  if s is a term with head symbol f and t is a term with head symbol g and $f \neq g$. This is only possible because the theories are collapse-free.

- Merge

  $x \overset{?}{=} s, x \overset{?}{=} t \succ fail$

  if s is a term with head symbol f and t is a term with head symbol g and f and g belong to signatures of different theories. This also is only possible because the theories are collapse-free.

- Combined Occur-Check

  $P \succ fail$

  if P contains a compound cycle.

- Var-Rep (Variable Replacement)

  $\{x \overset{?}{=} y\} \cup P \succ \{x \overset{?}{=} y\} \cup P\{x \hookleftarrow y\}$

  if both x and y occur in P and y occurs in the original problem P^0 or x does not occur in P^0.

- Remove

  $\{x \overset{?}{=} s\} \cup P \succ P$

  if $x \in X \backslash Var(P^0)$ and $x \notin Var(s) \cup Var(P)$

  A variable assignment can be dropped if the variable is neither present in the original problem nor in the rest of the equations looked upon and if the assignment does not violate the occurs check.

There are also two rules which are admissible and tailored to AC-unification:

- E-Rep

  $\{x \overset{?}{=} s\} \cup P \succ \{x \overset{?}{=} s\} \cup P\{x \hookleftarrow s\}$

  if P is a pure subproblem with no cycle in its occur-check graph and $x \in Var(P)$ and $s \notin X$.

- E-Cancel $x + s \overset{?}{=} x + t \succ s \overset{?}{=} t$

  if $+ \in \Sigma_i$ with $i \geqslant 1$ i.e. $+$ is an AC operator. This is similar to Stickel's algorithm which also drops common variables.

Theorem 3.5.6 (Soundness and completeness of the rules [BCD90]). If from our set of equations $P^0 \succ_* S$ we derive an S which is in dag-solved form, it is a most general unifier of P^0. Every most general unifier S is derivable via $\succ_*$ in a finite number of steps.

If we now want to solve the equation set $\{g(u, u, u) = g(f(a, x), f(b, y), f(c, z))\}$, then we can solve it by the derivations given in figure 3.3. The steps done by variable abstraction, replacement and removal are straightforward. The first E-resolution is in the empty theory and just consists of one decomposition step. The second one will be solved by Stickel's elementary algorithm, which solves exactly the system of diophantine equations from chapter 3.3. Assigning the fresh variables $z_1, \ldots, z_8$ to the basis vectors, one solution $\underline{s}$ can be found as $\underline{b_5} + \underline{b_3} + \underline{b_2} = (1, 2, 1, 2, 1, 2, 3)$ which can be found manually by the following assumptions: we know that every component of the solutions must be $\geqslant 1$. Also components $\pi_1(\underline{s}), \pi_3(\underline{s})$ and $\pi_5(\underline{s})$ must be equal to 1, since they will be assigned to constants. Because these constants are different, each component vector $\underline{v}$ with $\pi_1(\underline{v}) = 1$ must have $\pi_3(\underline{v}) = 0$ and $\pi_5(\underline{v}) = 0$ (and vice versa for the positions 3 and 5). Taking this into account, the solution $\underline{b_5} + \underline{b_3} + \underline{b_2}$ is soon found. In the end, we have finally found one of our most general unifiers $\{u \hookleftarrow f(a, b, c), x \hookleftarrow f(b, c), y \hookleftarrow f(a, c), z \hookleftarrow f(a, b)\}$ and leave it as an exercise to show it is the only one.

$$
\begin{array}{ll}
 & \{g(u,u,u) = g(f(a,x), f(b,y), f(c,z))\} \\
\succ_{VA} & \{g(u,u,u) = g(v_1,v_2,v_3), v_1 = f(a,x), v_2 = f(b,y), v_3 = f(c,z)\} \\
\succ_{E-Res} & \{v_1 = u, v_2 = u, v_3 = u, v_1 = f(a,x), v_2 = f(b,y), v_3 = f(c,z)\} \\
\succ_{Var-Rep} & \{v_1 = u, v_2 = u, v_3 = u, u = f(a,x), v_2 = f(b,y), v_3 = f(c,z)\} \\
\succ_{Var-Rep} & \{v_1 = u, v_2 = u, v_3 = u, u = f(a,x), u = f(b,y), v_3 = f(c,z)\} \\
\succ_{Var-Rep} & \{v_1 = u, v_2 = u, v_3 = u, u = f(a,x), u = f(b,y), u = f(c,z)\} \\
\succ_{Remove} & \{v_2 = u, v_3 = u, u = f(a,x), u = f(b,y), u = f(c,z)\} \\
\succ_{Remove} & \{v_3 = u, u = f(a,x), u = f(b,y), u = f(c,z)\} \\
\succ_{Remove} & \{u = f(a,x), u = f(b,y), u = f(c,z)\} \\
\succ_{VA} & \{u = f(v_4,x), u = f(b,y), u = f(c,z), v_4 = a\} \\
\succ_{VA} & \{u = f(v_4,x), u = f(v_5,y), u = f(c,z), v_4 = a, v_5 = b\} \\
\succ_{VA} & \{u = f(v_4,x), u = f(v_5,y), u = f(v_6,z), v_4 = a, v_5 = b, v_6 = c\} \\
\succ_{E-Res} & \{u = f(z_5,z_3,z_2), v_4 = z_5, x = f(z_3,z_2), v_5 = z_3, y = f(z_5,z_2), \\
 & v_6 = z_2, z = f(z_5,z_3), v_4 = a, v_5 = b, v_6 = c\} \\
\succ_{Var-Rep} & \{u = f(z_5,z_3,z_2), v_4 = z_5, x = f(z_3,z_2), v_5 = z_3, y = f(z_5,z_2), \\
 & v_6 = z_2, z = f(z_5,z_3), z_5 = a, v_5 = b, v_6 = c\} \\
\succ_{Remove} & \{u = f(z_5,z_3,z_2), x = f(z_3,z_2), v_5 = z_3, y = f(z_5,z_2), v_6 = z_2, z = f(z_5,z_3), z_5 = a, v_5 = b, v_6 = c\} \\
\succ_{Var-Rep} & \{u = f(z_5,z_3,z_2), x = f(z_3,z_2), v_5 = z_3, y = f(z_5,z_2), v_6 = z_2, z = f(z_5,z_3), z_5 = a, z_3 = b, v_6 = c\} \\
\succ_{Remove} & \{u = f(z_5,z_3,z_2), x = f(z_3,z_2), y = f(z_5,z_2), v_6 = z_2, z = f(z_5,z_3), z_5 = a, z_3 = b, v_6 = c\} \\
\succ_{Var-Rep} & \{u = f(z_5,z_3,z_2), x = f(z_3,z_2), y = f(z_5,z_2), v_6 = z_2, z = f(z_5,z_3), z_5 = a, z_3 = b, z_2 = c\} \\
\succ_{Remove} & \{u = f(z_5,z_3,z_2), x = f(z_3,z_2), y = f(z_5,z_2), z = f(z_5,z_3), z_5 = a, z_3 = b, z_2 = c\} \\
\succ_{Var-Rep} & \{u = f(a,z_3,z_2), x = f(z_3,z_2), y = f(a,z_2), z = f(a,z_3), z_5 = a, z_3 = b, z_2 = c\} \\
\succ_{Remove} & \{u = f(a,z_3,z_2), x = f(z_3,z_2), y = f(a,z_2), z = f(a,z_3), z_3 = b, z_2 = c\} \\
\succ_{Var-Rep} & \{u = f(a,b,z_2), x = f(b,z_2), y = f(a,z_2), z = f(a,b), z_3 = b, z_2 = c\} \\
\succ_{Remove} & \{u = f(a,b,z_2), x = f(b,z_2), y = f(a,z_2), z = f(a,b), z_2 = c\} \\
\succ_{Var-Rep} & \{u = f(a,b,c), x = f(b,c), y = f(a,c), z = f(a,b), z_2 = c\} \\
\succ_{Remove} & \{u = f(a,b,c), x = f(b,c), y = f(a,c), z = f(a,b)\}
\end{array}
$$

Figure 3.3: Rule application to get a solution of $\{g(u,u,u) = g(f(a,x), f(b,y), f(c,z))\}$

CHAPTER 4

Deduction Modulo

Sir Bedevere: "...Exactly. So, logically..."
Peasant: "If she weighed the same as a duck... she's made of wood."
Sir Bedevere: "And therefore..."
Peasant: "...A witch!"
Monty Python - Knights of the Holy Grail

4.1 Deduction Modulo

The original paper [DHK03] on deduction modulo presents both a sequent calculus and a resolution calculus modulo an equational theory and rewrite rules on terms $\mathcal{E}$ and a rewriting system $\mathcal{R}$ on propositional formulas. In Sequent Calculus modulo, the rules of ordinary LK are changed, such that the principal formula in an inference step only needs to be equal modulo the theory. To express resolution modulo, equational resolution is complemented by the Extended Narrowing rule, which handles rewriting of propositional formulas. The name of this calculus is Extended Narrowing and Resolution (ENAR).

To define the notion of equality modulo $\mathcal{RE}$ we stick close to Dowek's [DHK03]:

Definition 4.1.1. A *term rewrite rule* is of the form $l \longrightarrow r$, where l and r are terms and all variables of r are also in l. An *equational axiom* is a pair of terms $l = r$.

A *proposition rewrite rule* is of the form $l \longrightarrow r$ where l and r are propositional formulas and additionally l is atomic.

A *class rewrite system* $\mathcal{RE}$ consists of a set $\mathcal{R}$ of proposition rewrite rules and a set $\mathcal{E}$ of equational axioms and rewrite rules on terms.

Definition 4.1.2 ($\mathcal{R}$-rewriting)**.** Given a rewrite system $\mathcal{R}$, $P \longrightarrow_{\mathcal{R}} P'$ if $P_{|\omega} = \sigma(l)$ and $P' = P[\sigma(r)]_\omega$ for a rule $l \longrightarrow r \in \mathcal{R}$ with a substitution σ at position ω.

Definition 4.1.3 ($\mathcal{RE}$-rewriting)**.** Given a rewrite system $\mathcal{R}$, $P \longrightarrow_{\mathcal{RE}} P'$ if $P \approx_{\mathcal{E}} Q$, $Q_{|\omega} = \sigma(l)$ and $P' \approx_{\mathcal{E}} Q[\sigma(r)]_\omega$ for a rule $l \longrightarrow r \in \mathcal{R}$ with a substitution σ at position ω.

Definition 4.1.4 ($\mathcal{RE}$-equality)**.** If $P \approx_{\mathcal{RE}} Q$, then either $P \equiv_{\mathcal{E}} Q$ or $P \longleftrightarrow^*_{\mathcal{RE}} Q$ (where $\longleftrightarrow^*_{\mathcal{RE}}$ is the reflexive, symmetric and transitive closure of $\longrightarrow_{\mathcal{RE}}$).

A common example given for a class rewrite system is the term equality $\mathcal{E} = ACU$ together with the propositional rewrite system $\mathcal{R} = \{x \times y = 0 \longrightarrow x = 0 \vee y = 0\}$. Then $x + (0 + y) \times (a + x) \approx_{\mathcal{RE}} x + y = 0 \vee x + a = 0$ holds by the following derivation:

$$\begin{array}{lr}
 & (x + (0 + y)) \times (a + x) = 0 \\
\longrightarrow_{\mathcal{E}} & (x + (y + 0)) \times (a + x) = 0 \\
\longrightarrow_{\mathcal{E}} & (x + y) \times (a + x) = 0 \\
\longrightarrow_{\mathcal{R}} & x + y = 0 \vee a + x = 0 \\
\longrightarrow_{\mathcal{E}} & x + y = 0 \vee x + a = 0
\end{array}$$

4.2 Sequent Calculus modulo

$$\frac{}{P \vdash_{\mathcal{RE}} Q} \qquad\qquad \frac{}{R \vdash_{\mathcal{RE}}} \bot, l$$

axiom introduction ($P \approx_{\mathcal{RE}} Q$) bottom introduction ($R \approx_{\mathcal{RE}} \bot$)

$$\frac{\Gamma_1, P \vdash_{\mathcal{RE}} \Delta_1 \qquad \Gamma_2 \vdash_{\mathcal{RE}} Q, \Delta_2}{\Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_1, \Delta_2} \text{cut}$$

cut ($P \approx_{\mathcal{RE}} Q$)

$$c, l \, \frac{\Gamma, Q_1, Q_2 \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta} \qquad\qquad \frac{\Gamma \vdash_{\mathcal{RE}} Q_1, Q_2 \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta} \, c, r$$

contraction ($P \approx_{\mathcal{RE}} Q_1 \approx_{\mathcal{RE}} Q_2$)

$$w, l \, \frac{\Gamma \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta} \qquad\qquad \frac{\Gamma \vdash_{\mathcal{RE}} \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta} \, w, r$$

weakening

Figure 4.1: Introduction- and structural rules of Sequent Calculus modulo, equivalent to [DHK03]

Sequent Calculus modulo is similar to Gentzen's original Sequent Calculus, but active formulas only need to be equal modulo $\approx_{\mathcal{RE}}$. The usual presentation is given in [DHK03], for our purposes, a calculus with multiplicative binary rules and introduction rules without context is easier to handle. Instead of having a permutation rule, sequents are defined as a pair of multisets of formulas, the antecedent and the succedent separated by the derivation symbol $\vdash_{\mathcal{RE}}$. The rules can be found in figure 4.2 and 4.2. We

$$\wedge,l\ \frac{\Gamma, P, Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta} \qquad \frac{\Gamma_1 \vdash_{\mathcal{RE}} P, \Delta_1 \qquad \Gamma_2 \vdash_{\mathcal{RE}} Q, \Delta_2}{\Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} R, \Delta_1, \Delta_2}\ \wedge,r$$

conjunction ($R \approx_{\mathcal{RE}} P \wedge Q$)

$$\vee,l\ \frac{\Gamma_1, P \vdash_{\mathcal{RE}} \Delta_1 \qquad \Gamma_2, Q \vdash_{\mathcal{RE}} \Delta_2}{\Gamma_1, \Gamma_2, R \vdash_{\mathcal{RE}} \Delta_1, \Delta_2} \qquad \frac{\Gamma \vdash_{\mathcal{RE}} P, Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\ \vee,r$$

disjunction ($R \approx_{\mathcal{RE}} P \vee Q$)

$$\Rightarrow,l\ \frac{\Gamma_1 \vdash_{\mathcal{RE}} P, \Delta_1 \qquad \Gamma_2, Q \vdash_{\mathcal{RE}} \Delta_2}{\Gamma_1, \Gamma_2, R \vdash_{\mathcal{RE}} \Delta_1, \Delta_2} \qquad \frac{\Gamma, P \vdash_{\mathcal{RE}} Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\ \Rightarrow,r$$

implication ($R \approx_{\mathcal{RE}} P \Rightarrow Q$)

$$\frac{\Gamma \vdash_{\mathcal{RE}} P, \Delta}{\Gamma, R \vdash_{\mathcal{RE}} \Delta}\ \neg,r \qquad \frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta}{\Gamma \vdash_{\mathcal{RE}} R, \Delta}\ \neg,r$$

negation ($R \approx_{\mathcal{RE}} \neg P$)

$$\forall,l\ \frac{\Gamma, \{t/x\}Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta} \qquad \forall,r\ \frac{\Gamma \vdash_{\mathcal{RE}} \{v/x\}Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}$$

universal quantification ($P \approx_{\mathcal{RE}} \forall x\ Q$ and v is a fresh free variable)

$$\exists,l\ \frac{\Gamma, \{v/x\}Q \vdash_{\mathcal{RE}} \Delta}{\Gamma, P \vdash_{\mathcal{RE}} \Delta} \qquad \exists,r\ \frac{\Gamma \vdash_{\mathcal{RE}} \{t/x\}Q, \Delta}{\Gamma \vdash_{\mathcal{RE}} P, \Delta}$$

existential quantification ($P \approx_{\mathcal{RE}} \exists x\ Q$ and v is a fresh free variable)

Figure 4.2: Logical rules of Sequent Calculus modulo, equivalent to [DHK03]

write $\vdash_{\mathbf{RE}}\ P \vdash_{\mathcal{RE}} Q$ if the sequent $P \vdash_{\mathcal{RE}} Q$ is provable via the rules. Similar to theory-free sequent calculus, every rule has some context formulas which do not change and one or two auxiliary formulas in the upper sequent(s) which are transformed to the principal formula in the lower sequent.

The translation to the Dowek et al.'s calculus can be done via a series of weakenings for every multiplicative and introduction rule. We also choose to introduce Eigenvariables instead of fresh constants in the $\forall, r$ and $\exists, l$ rule.

Sometimes we will need to refer to a specific formula in a sequent or to a specific sequent in a proof. These are called formula-occurrence and sequent-occurrence respectively. For the thesis, the occurrence is sufficient, but if we wanted to change occurrences, we could define formula-occurrences as indices in the antecedent or succedent set and define sequent-occurrences as a list of arcs to follow from the root of the derivation tree.

For rewrite systems in which cut-elimination is possible, Dowek et. al. proved the equivalence to a compatible theory in plain first order logic:

Definition 4.2.1. A theory $\mathcal{T}$ and a class rewrite system $\mathcal{RE}$ are *compatible* if $P \approx_{\mathcal{RE}} Q$ implies $\mathcal{T} \vdash P \Leftrightarrow Q$ and for every proposition P in $\mathcal{T}$, we have $\vdash_{\mathbf{RE}} P$.

Theorem 4.2.2. For every class rewrite system $\mathcal{RE}$ there exists a compatible theory $\mathcal{T}$.

Proof. given in [DHK03]:
For each term rewrite rule $l \longrightarrow r$ and term equality $l = r$ add the axiom

$$\forall \overline{x}\, l = r$$

where $\overline{x}$ are the free variables in l and r. For each proposition rewrite rule $l \longrightarrow r$ add the axiom

$$\forall \overline{x}\, l \Leftrightarrow r$$

where $\overline{x}$ are the free variables in l and r. □

The above formulation is also possible in a sequent calculus without an equality predicate, but this one is shorter. Please notice that our formulation of sequent calculus modulo does not automatically have an equality predicate, since a paramodulation would need to be added to resolution. Modeling all equalities in the underlying class rewrite system is not feasible, because adding rules changes some basic properties of the system. For instance confluence and termination need to be proven anew, if they still hold.

Theorem 4.2.3 (equivalence of $\vdash$ and $\vdash_{\mathbf{RE}}$)**.** For a given class rewrite system $\mathcal{RE}$ and a compatible theory $\mathcal{T}$,

$$\mathcal{T}, \Gamma \vdash \Delta \text{ if and only if } \Gamma \vdash_{\mathbf{RE}} \Delta$$

Proof. Given in [DHK03]. □

The proof uses the fact that every rule in sequent calculus modulo can be described as an application of the theory-free rule of sequent calculus with an added step which proves the theory equivalence of the principal formula from the axioms in $\mathcal{T}$. This leads also to an alternative formulation of sequent calculus modulo, in which sequent calculus is only extended by the conversion rules given in figure 4.3. Amongst others, this version is used in [Her10].

$$\frac{\Gamma, P \vdash_{\mathcal{RE}} \Delta}{\Gamma, Q \vdash_{\mathcal{RE}} \Delta}\, conv, l \qquad\qquad \frac{\Gamma \vdash_{\mathcal{RE}} P, \Delta}{\Gamma \vdash_{\mathcal{RE}} Q, \Delta}\, conv, r$$

$$P \approx_{\mathcal{RE}} Q \qquad\qquad\qquad P \approx_{\mathcal{RE}} Q$$

Figure 4.3: Conversion rules to introduce deduction modulo into theory-free sequent calculus

4.3 Extended Narrowing and Resolution

ENAR is a constrained resolution calculus, which means that in contrast to eagerly applying the most general unifier like in equational resolution [Plo72], a list of $\mathcal{E}$-equalities, under which a clause set is valid is kept. If there is a solution, solving the equalities yields at least one mgu which needs to be applied to the resolution proof.

The usual clause set transformation rules are used to bring a formula into clause form; in the same step skolemization is achieved. For this reason, the strongly quantified variables have to be added as a label to a formula during the application of the rewrite rules given in figure 4.4. Because skolemization removes weakly quantified variables, the label contains exactly the free variables of a formula. Notationally, the label is added as superscript to the formula. Labels need to be taken taken into account during substitution and for $\mathcal{RE}$-equality; since a substitution θ can change the free variables in a formula P^l, the label l' of $\theta P^{l'}$ must be updated to the free variables of θP. Similarly, for the labeled propositions P^l to be $\mathcal{E}$-equivalent or to $\mathcal{R}$-rewrite one the labeled proposition and $Q^{l'}$ requires that the free variables do not change, i.e. $P \approx_{\mathcal{E}} Q$ or $P \longrightarrow_{\mathcal{R}} Q$ must be fulfilled and $l = l'$.

Like in theory-free resolution, a literal is either an atomic proposition or a negated one and the symbol $\Box$ denotes the empty clauseset. We also introduce some shortcut notations: for a proposition P^l, a set of propositions ψ and a set of sets of propositions Φ, the set unions $\psi \cup \{P^l\}$ and $\Phi \cup \{\psi\}$ will be written as ψ, P^l and Φ, ψ respectively.

$$\begin{array}{ll}
\Phi, (\psi,(P \wedge Q)^l\ ) & \longrightarrow \Phi,(\psi, P^l), (\psi, Q^l) \\
\Phi, (\psi,(P \vee Q)^l\ ) & \longrightarrow \Phi,(\psi, P^l, Q^l) \\
\Phi, (\psi,(P \Rightarrow Q)^l\ ) & \longrightarrow \Phi,(\psi, (\neg P)^l, Q^l) \\
\Phi, (\psi,\bot^l\ ) & \longrightarrow \Phi,\psi \\
\Phi, (\psi,(\forall x P)^l\ ) & \longrightarrow \Phi,(\psi, P^{l,x}), \text{ where } x \text{ is a fresh variable} \\
\Phi, (\psi,(\exists x P)^{y_1,\dots,y_n}\ ) & \longrightarrow \Phi,\ (\psi, P\{x \mapsfrom f(y_1,\dots,y_n)\}^{y_1,\dots,y_n}), \text{ where } f \text{ is a fresh function symbol} \\
\Phi, (\psi,(\neg(P \wedge Q))^l\ ) & \longrightarrow \Phi,(\psi, (\neg P)^l, (\neg Q)^l) \\
\Phi, (\psi,(\neg(P \vee Q))^l\ ) & \longrightarrow \Phi,(\psi, \neg P)^l, (\psi, \neg Q)^l \\
\Phi, (\psi,(\neg(P \Rightarrow Q))^l\ ) & \longrightarrow \Phi,(\psi, P^l, (\neg Q)^l) \\
\Phi, (\psi,(\neg\bot)^l\ ) & \longrightarrow \Phi \\
\Phi, (\psi,\neg\neg P^l\ ) & \longrightarrow \Phi,P^l \\
\Phi, (\psi,\neg(\forall x P)^{y_1,\dots,y_n}\ ) & \longrightarrow \Phi,(\psi, (\neg P\{x \mapsfrom f(y_1,\dots,y_n)\})^{y_1,\dots,y_n}), \text{ where } f \text{ is a fresh function symbol} \\
\Phi, (\psi,(\exists x P)^l\ ) & \longrightarrow \Phi(\psi, (\neg P)^{l,x}), \text{ where } x \text{ is a fresh variable}
\end{array}$$

Figure 4.4: Clause set transformation rules for ENAR

Since deduction modulo allows to rewrite (propositional) formulas, there is no direct semantics of ENAR like for equational resolution[1]. Instead equivalence to sequent calculus modulo is shown. The confluence of the rewrite relation $\longrightarrow_{\mathcal{RE}}$ is a necessary ingredient in the soundness and completeness proofs of ENAR, which can be seen as the condition that rewriting is in some sense deterministic. In-

[1]There it is shown that it suffices to use the factor algebra on terms induced by the equality relation instead of the original term algebra.

Resolution

$$\frac{\{P_1,\ldots,P_n,Q_1,\ldots,Q_m\}[\mathscr{C}_1] \qquad \{\neg R_1,\ldots,\neg R_p,S_1,\ldots,S_q\}[\mathscr{C}_2]}{\{Q_1,\ldots,Q_m,S_1,\ldots,S_q\}[\mathscr{C}_1 \cup \mathscr{C}_2 \cup \{P_1 \approx_{\mathcal{E}} \ldots \approx_{\mathcal{E}} P_n \approx_{\mathcal{E}} S_1 \approx_{\mathcal{E}} \ldots \approx_{\mathcal{E}} S_q\}]}$$

Extended Narrowing

$$\frac{U[\mathscr{C}]}{U'[\mathscr{C} \cup \{U|_\omega \approx_{\mathcal{E}} l\}]} \text{ if } l \longrightarrow r \in \mathcal{R}, U|_\omega \text{ atomic proposition and } U' \in cl(\{U[r]_\omega\})$$

Figure 4.5: Rules of ENAR

terestingly, termination is not necessary for the proofs, although non-terminating rewrite systems are not well suited for automation.

Theorem 4.3.1 (ENAR Soundness)**.** Let $\mathcal{RE}$ be a class rewrite system with a confluent $\mathcal{RE}$-rewrite relation $\longrightarrow_{\mathcal{RE}}$. If there exists an ENAR refutation $cl(\phi, \neg\psi)[\emptyset] \underset{\mathcal{RE}}{\Vdash\!\!\longrightarrow} \Box[\mathscr{C}]$ with $\mathcal{E}$-unifiable constraints $\mathscr{C}$, then there exists a LK modulo proof of $\phi \vdash_{\mathcal{RE}} \psi$.

Proof. Given in [DHK03]. □

Theorem 4.3.2 (ENAR Completeness)**.** Let $\mathcal{RE}$ be a class rewrite system with a confluent $\mathcal{RE}$-rewrite relation $\longrightarrow_{\mathcal{RE}}$. If there exists a cut-free proof of $\phi \vdash_{\mathcal{RE}} \psi$, then there is an ENAR refutation $cl(\phi, \neg\psi)[\emptyset] \underset{\mathcal{RE}}{\Vdash\!\!\longrightarrow} \Box[\mathscr{C}]$ with $\mathcal{E}$-unifiable constraints $\mathscr{C}$.

Proof. Given in [DHK03]. □

There is also a stronger completeness result by Hermant:

Theorem 4.3.3 (ENAR soundness w.r.t cut-free sequent calculus modulo)**.** Let $\mathcal{RE}$ be a class rewrite system with a confluent $\mathcal{RE}$-rewrite relation $\longrightarrow_{\mathcal{RE}}$. If there exists an ENAR refutation $cl(\phi, \neg\psi)[\emptyset] \underset{\mathcal{RE}}{\Vdash\!\!\longrightarrow} \Box[\mathscr{C}]$ with $\mathcal{E}$-unifiable constraints $\mathscr{C}$, then there exists a cut-free LK modulo proof of $\phi \vdash_{\mathcal{RE}} \psi$.

Proof. Given in [Her10]. □

4.4 Cut-Elimination

The elimination of the cut rule in sequent calculus modulo cannot be shown in general, for instance one could try to formalize naive set theory by rewrite rules and yield a proof where cut is not admissible: [Dow01]

Naive set theory[2] allows the description of sets via the comprehensions scheme

$$\forall \overline{x} \exists y \forall z ((z \in y) \Leftrightarrow P)$$

for arbitrary propositions P with the variables $\overline{x}$ and z being free in P and y not occurring in P. Its skolemization is called the conversion scheme:

$$\forall \overline{x} \forall z ((z \in f_{\overline{x},z,P}(\overline{x})) \Leftrightarrow P)$$

If we abstract f over the variables $\overline{x}$, we can write $f_{\overline{x},z,P}(\overline{x})$ prettier as $\{z|P\}$. The conversion scheme now has the form

$$\forall \overline{x} \forall z (z \in \{z|P\} \Leftrightarrow P)$$

It is also possible to restrict z to be an element of some set A in the conversion scheme:

$$\forall \overline{x} \forall z (z \in \{z \in A|P\} \Leftrightarrow P)$$

A possible rewrite rule for restricted conversion would look like the following:

$$t \in \{z \in A|P\} \longrightarrow A \wedge P[t/z]$$

Now, Crabbé's proposition, a variant of Russel's paradox [Rus96, §100]:

$$\{x \in A|x \notin x\} \in \{x \in A|x \notin x\}$$

rewrites to

$$\{x \in A|x \notin x\} \in A \wedge \neg\{x \in A|x \notin x\} \in \{x \in A|x \notin x\}$$

which on one hand shows non-termination of the conversion rewrite rule and on the other hand gives rise to a propositional sequent calculus modulo proof containing a cut which can not be eliminated. Let C denote Crabbé's proposition and B stand for $\{x \in A|x \notin x\} \in A$, then our concrete instance of conversion can be formulated as the rule $C \longrightarrow B \wedge \neg C$. Using this rule the following proof of $\vdash_{\mathcal{RE}} \neg B$ has no cut-free proof:

$$\dfrac{\dfrac{\dfrac{B \vdash_{\mathcal{RE}} B \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \vdash_{\mathcal{RE}} A}{A, B \vdash_{\mathcal{RE}} A}\,w,l}{A, B, \neg A \vdash_{\mathcal{RE}}}\,\neg,l}{A, A \vdash_{\mathcal{RE}}}\,\wedge,l}{A \vdash_{\mathcal{RE}}}\,c,l}{\vdash_{\mathcal{RE}} \neg A}\,\neg,r}{B \vdash_{\mathcal{RE}} A}\,\wedge,l \qquad \dfrac{\dfrac{\dfrac{\dfrac{A \vdash_{\mathcal{RE}} A}{A, B \vdash_{\mathcal{RE}} A}\,w,l}{A, B, \neg A \vdash_{\mathcal{RE}}}\,\neg,l}{A, A \vdash_{\mathcal{RE}}}\,\wedge,l}{A \vdash_{\mathcal{RE}}}\,c,l}{B \vdash_{\mathcal{RE}}}\,cut}{\vdash_{\mathcal{RE}} \neg B}\,\neg,r$$

[2]Naive set theory is contradictory – we will use a variant of one of the best known counterexamples, Russel's paradox, to show the non-termination of the rewrite system. In [DW03] there is also a counterexample to cut-elimination for a confluent, terminating rewrite system, but the proof uses the encoding of proofs in typed lambda calculus introduced in the paper, so we will stick to this one.

For a similar proof in natural deduction, there are direct proofs [Ekm94, Hal83] of non-admissibility of cut-elimination. In our case, we can easily check that there is no ENAR proof of $\neg B$, since neither the extended resolution rule nor the extended narrowing rule is applicable. Since ENAR corresponds to the cut-free segment of sequent calculus modulo, there also exists no cut-free proof in sequent calculus modulo.

For the rules of simple type theory, confluent and terminating quantifier free rewrite systems and what is most important for this work, also for congruences on terms, G. Dowek and B. Werner proved that cut-elimination holds. [DW03]

Resolution Proofs in Sequent Calculus modulo

A sequent containing only atomic formulas can be interpreted as a clause with the formulas in the antecedent corresponding to the negative literals and those in the succedent corresponding to the positive literals. The extended resolution rule can be simulated in sequent calculus modulo by contracting all the positive and negative clauses and then introducing a cut over the remaining positive and negative literal. Since the extended narrowing rule can rewrite to a non-atomic formula, the sequent needs to be brought into clause form again. This is one of the causes why we restrict ourselves to equalities on terms only. Another important detail is that we have to solve the final set of term constraints to yield a (most general) unifier σ which needs to be applied to the proof to gain an instantiation of the terms resolved over. This is not yet a ground proof, since unification may leave free variables in the terms. We can solve this by introducing a substitution τ which maps each variable to a fresh constant. Let $\theta = \sigma\tau$ the ground substitution, then we can translate each resolution step as can be seen in figure 4.6. Since θ is a mgu for the formulas resolved over, the contractions are sound. Subsequently, θ is a mgu for both positive and negative literals, so the condition $P_1 \approx_{\mathcal{RE}} R_1$ holds and the cut rule is applicable.

$$\frac{\{P_1, \dots, P_n, Q_1, \dots, Q_m\}[\mathscr{C}_1] \qquad \{\neg R_1, \dots, \neg R_p, S_1, \dots, S_q\}[\mathscr{C}_2]}{\{Q_1, \dots, Q_m, S_1, \dots, S_q\}[\mathscr{C}_1 \cup \mathscr{C}_2 \cup \{P_1 \approx_{\mathcal{E}} \dots P_n \approx_{\mathcal{E}} R_1 \approx_{\mathcal{E}} \dots \approx_{\mathcal{E}} R_p\}]}$$

$$\Downarrow$$

$$\cfrac{\cfrac{\cfrac{\overline{Q}_-\theta \vdash_{\mathcal{RE}} P_1\theta, \dots, P_n\theta, \overline{Q}_+\theta}{\overline{Q}_-\theta \vdash_{\mathcal{RE}} P_1\theta, \overline{Q}_+\theta}\,c, lr \qquad \cfrac{R_1\theta, \dots, R_n\theta, \overline{S}_-\theta \vdash_{\mathcal{RE}} \overline{S}_+\theta}{R_1\theta, \overline{S}_-\theta \vdash_{\mathcal{RE}} \overline{S}_+\theta}\,c, lr}{\overline{Q}_-\theta, \overline{S}_-\theta \vdash_{\mathcal{RE}} \overline{Q}_+\theta, \overline{S}_+\theta}\,cut}{\overline{Q}_-\theta, \overline{S}_-\theta \vdash_{\mathcal{RE}} \overline{Q}_+\theta, \overline{S}_+\theta}\,c, lr$$

Figure 4.6: Translation of a resolution step to sequent calculus modulo

In the end, deriving the empty clause in ENAR corresponds to deriving the the empty sequent i.e. the most simple contradiction in sequent calculus modulo.

CHAPTER 5

Cut-elimination Modulo

Besides it is an error to believe that rigor in the proof is the enemy of simplicity. On the contrary we find it confirmed by numerous examples that the rigorous method is at the same time the simpler and the more easily comprehended.

David Hilbert

5.1 CERES

Since the actual algorithm of CERES modulo is very close to the original CERES method, most of the definitions would be a repetition with small changes. Instead, we here give an overview of how the method works. It consists of four steps:

1. Skolemization of the proof
   During the construction of the proof projections in step 4, additional variables may turn up in a sequent. To prevent capture of eigenvariables, the proof will be skolemized. In some cases, the skolem terms can be also given a mathematical interpretation [BHL$^+$08].

2. Cut-transformation to tautologies via T_{cut}
   This transformation replaces cut rules by implication rules on the left, which add tautologies to the antecedent of the conclusion. We now have a cut-free proof with additional formulas in the end-sequent, which will be removed in step 4. Newer formulations of CERES do not use this (somehow artificial) transformation anymore, because it is only present to ease the reasoning about the method but the same characteristic clause set can be constructed with only small modifications to the definitions in step 3.

3. Construction of the characteristic clause set and refutation of it
   Since our formulation of sequent calculus does not allow a context in the axiom rules and we can restrict ourselves to the introduction of atom formulas, the sequent of an axiom rule directly corresponds to a clause set. We now follow all formulas going into a tautology in the end-sequent up

to the axiom rule in which they were introduced and construct the so called characteristic clause set from them which still proves the tautology in the end-sequent. Since a tautology in the antecedent of a sequent means that this formula is unsatisfiable, we construct a resolution refutation instead of a sequent calculus proof.

4. Constructing the projections and completing the resolution proof with it
In the last step, we construct a proof of the end-sequent with one of the clauses from the characteristic clause set as additional formula, the so called projection of the proof to a clause. Then we repeat the resolution refutation from step 3 with the projections. The simulation of a resolution rule in sequent calculus again introduces atomic cuts, but these are much easier to eliminate, so constructing a proof in this so called atomic cut normal form (ACNF) is sufficient for our cause. Even with the elimination of the atomic cuts, the CERES method has a lower computational complexity than reductive cut-elimination [BL00].

5.2 CERES modulo

Definitions

For the formulation of CERES modulo, we need some additional definitions.

Definition 5.2.1 (Merging of Sequents)**.** For two sequents $S_1 : \overline{P} \vdash_{\mathcal{RE}} \overline{Q}$ and $S_2 : \overline{R} \vdash_{\mathcal{RE}} \overline{T}$, $S_1 \circ S_2$ is defined as $\overline{P}, \overline{R} \vdash_{\mathcal{RE}} \overline{Q}, \overline{T}$.

Definition 5.2.2 (Merging of Sets of Sequents)**.** For two sets of sequents $L_1 : \{S_1, \dots S_n\}$ and $S_2 : \{T_1, \dots, T_m\}$, $S_1 \otimes S_2$ is defined as $\{S_i \circ T_j | 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$.

Definition 5.2.3 (Ancestor Relation)**.**
An axiom has no *immediate ancestor*. For unary rules, let λ denote the sequent occurrence of a consequent of the sequent occurrence μ. If α is the occurrence of the principal formula in λ and β is the occurrence of the auxiliary formula in μ, then β is an *immediate ancestor* of α. If α is a formula occurrence of the context of λ, then its occurrence β in the context of μ is also an ancestor of α.

For binary rules, let λ denote the sequent occurrence of a consequent of the sequent occurrences μ_1 and μ_2. If α is the occurrence of the principal formula in λ and β_1 and β_2 are the occurrences of the auxiliary formulas in μ_1 and μ_2, then β_1 and β_2 are *immediate ancestors* of α. If α is a formula occurrence of the context of λ, then its occurrences β_1 and β_2 in the context of μ are also ancestors of α.

The *ancestor* relation is the reflexive, transitive closure over the immediate ancestor relation.

Since we always define all the auxiliary formulas of a rule as immediate ancestors, we notice that a rule can either only work on ancestors or only work on non-ancestor of a formula. This will simplify the case distinctions on proofs concerning the ancestor relation in sequent calculus proofs.

Definition 5.2.4 (ancestor and non-ancestor function)**.** The *ancestor* function $anc(\lambda, \alpha)$ returns the subsequent containing exactly those formulas of the sequent occurrence λ, which are in ancestor relation to the formula occurrence α.
The *non-ancestor* function $\overline{anc}(\lambda, \alpha)$ returns the subsequent containing exactly those formulas of the sequent occurrence λ, which are not in ancestor relation to the formula occurrence α.

From the definition it is easy to see that if S is the sequent denoted by the sequent occurrence λ, $anc(\alpha, \lambda) \circ \overline{anc}(\alpha, \lambda) = S$ for any formula occurrence α.

Example

To give an example how ancestors are denoted, we formulate a simple proof that for all x, $x + 4$ is an even number(figure 5.1). To denote an even number, the predicate E is used, the function symbol $+$ is considered to be associative, commutative and having a the unit element 0, so our congruence $\mathcal{E} = ACU$ and $\mathcal{R} = \emptyset$. The ancestors of the formula CF are colored blue.

shorthand notation: $CF \equiv \forall x \forall y (E(x + y) \Rightarrow E((x + 1) + (y + 1)))$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{E(x+y) \vdash_{\mathcal{R}\mathcal{E}} E(x+y) \qquad E((x+1)+(y+1)) \vdash_{\mathcal{R}\mathcal{E}} E((x+1)+(y+1))}{E(x+y) \Rightarrow E((x+1)+(y+1)), E(x+y) \vdash_{\mathcal{R}\mathcal{E}} E((x+1)+(y+1))}\,{\Rightarrow,l}}{\forall x(E(x) \Rightarrow E(x+(1+1))), E(x+y) \vdash_{\mathcal{R}\mathcal{E}} E((x+1)+(y+1))}\,{\forall,l}}{\forall x(E(x) \Rightarrow E(x+(1+1)) \vdash_{\mathcal{R}\mathcal{E}} E(x+y) \Rightarrow E((x+1)+(y+1))}\,{\Rightarrow,r}}{\forall x(E(x) \Rightarrow E(x+(1+1))) \vdash_{\mathcal{R}\mathcal{E}} \forall y(E(x+y) \Rightarrow E((x+1)+(y+1)))}\,{\forall,r}}{\forall x(E(x) \Rightarrow E(x+(1+1))) \vdash_{\mathcal{R}\mathcal{E}} \forall x\forall y(E(x+y) \Rightarrow E((x+1)+(y+1)))}\,{\forall,r}$$

$$(\rho_1)$$

$$\dfrac{\dfrac{\dfrac{E(c+c) \vdash_{\mathcal{R}\mathcal{E}} E(c+c) \qquad E((c+c)+(1+1)) \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))}{E(c+c), E(c+c) \Rightarrow E((c+c)+(1+1)) \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))}\,{\Rightarrow,l}}{E(c+c), \forall y(E(c+y) \Rightarrow E((c+y)+(1+1))) \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))}\,{\forall,l}}{E(c+c), \forall x\forall y(E(x+y) \Rightarrow E((x+1)+(y+1))) \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))}\,{\forall,l}$$

$$(\rho_2)$$

$$\dfrac{\begin{matrix}(\rho_1) \\ \forall x(E(x) \Rightarrow E(x+(1+1))) \vdash_{\mathcal{R}\mathcal{E}} CF\end{matrix} \qquad \begin{matrix}(\rho_2) \\ E(c+c), CF \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))\end{matrix}}{E(c+c), \forall x(E(x) \Rightarrow E(x+(1+1))) \vdash_{\mathcal{R}\mathcal{E}} E(c+(1+(1+c)))}\,{cut}$$

Figure 5.1: Example for the ancestor relation

Since we will make proofs by induction, we will need a measure for the complexity of a formula. Its analogon for sequent calculus proofs is called the depth of a proof.

Definition 5.2.5 (Complexity of a formula)**.** If A is an atom formula, its complexity is 1. Suppose A and B are formulas of complexity n and m respectively. Then the complexity of $\neg A$ is $n + 1$ and the complexity of $A \circ B$ for $\circ \in \{\wedge, \vee, \Rightarrow\}$ is $max(n, m) + 1$.

Definition 5.2.6 (Depth of a proof)**.** An axiom introduction occurs at depth 1. Let ρ_1 and ρ_2 be proofs of depth n and m respectively. Then the complexity of the proof π applying a unary rule to ρ_1 is $n + 1$ and the complexity of the proof π' applying a binary rule to ρ_1 and ρ_2 is $max(n, m) + 1$.

Also the usual way of bringing a formula into prenex normal form before bringing it into skolem normal form makes the recognition of the mathematical arguments used harder. So we use structural skolemization instead, which replaces quantifiers at their original position. The notions of polarity and weak/strong quantifiers is then used to refer to quantifiers which would become existential or universal during prenex normal form transformation.

Definition 5.2.7 (Polarity [BL00])**.** The polarity of the occurrence of a formula A in a formula B is either positive or negative. If $A \equiv B$ it is positive. If B has the shape $F \vee G$, $F \wedge G$, $\forall x F$ or $\exists F$ and the polarity of λ in F or G is positive (negative), then the polarity of λ in B is still positive (negative). For $B \equiv F \Rightarrow G$, if λ occurs positively (negatively) in G, the polarity stays the same in B, but if λ occurs positively (negatively) in F, polarities switch i.e. λ occurs negatively (positively) in B. If $B \equiv \neg F$ and λ occurs positively (negatively) in F, it occurs negatively (positively) in B.

Definition 5.2.8 (Weak and strong quantifiers, [BL00])**.**
If $(\forall x)$ occurs positively (negatively) in a formula F, then $(\forall x)$ is a strong (weak) quantifier.
If $(\exists x)$ occurs positively (negatively) in a formula F, then $(\exists x)$ is a weak (strong) quantifier.

From now on we assume we do not have $\Leftrightarrow$ as a logical symbol(which is not present in our formulation of sequent calculus modulo anyway), since then polarity can become ambiguous for some formula occurrence. As an example take the formula

$$F \equiv (\forall x P(x) \Rightarrow \exists y Q(y)) \wedge (\exists y Q(y) \Rightarrow \forall x P(x))$$

where the first occurrence of $(\forall x)$ is weak and the second one is strong. But also $F \equiv \forall x P(x) \Leftrightarrow \exists y Q(y)$ where for consistency reasons, $(\forall x)$ would need to be both weak and strong. A simple solution is to replace every formula $A \Leftrightarrow B$ by $(A \Rightarrow B) \wedge (B \Rightarrow A)$, since they are logically equivalent. This leaves each occurrence of a quantifier to be either weak or strong in a formula.

Another source of problems is overbinding of variables which may destroy eigenvariable conditions if not handled properly. The easiest way is to give variables of the same name in different biding contexts a unique one. A formula where all quantified variables are unique is called rectified.

Definition 5.2.9 (rectified formula, [LF05])**.** A formula A is *rectified* if the following property is fulfilled: if $B_1 \odot B_2$ (for $\odot \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$) is a subformula of A and a variable x occurs bound in B_1 (B_2), then x does not occur in B_2 (B_1).

Additionally, the eigenvariables eliminated in the strong quantifier introduction rules in sequent calculus may have the same name on different branches of the proof tree. A regular proof then has unique eigenvariables in each branch.

Definition 5.2.10 (Regular proof [BL00])**.** A sequent calculus (modulo) proof is called regular if eigenvariables eliminated on different branches of the proof tree are different. More formally: Let ϕ be a subproof of an LK-proof of the form

$$\dfrac{\begin{matrix}(\phi_1) \\ \Gamma_1 \vdash_{\mathcal{RE}} \Delta_1\end{matrix} \qquad \begin{matrix}(\phi_1) \\ \Gamma_2 \vdash_{\mathcal{RE}} \Delta_2\end{matrix}}{S : \Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_1, \Delta_2}\, X$$

where X is a binary rule. Let $V1$ ($V2$) be the set of eigenvariables occurring in ϕ_1 (ϕ_2) but not in S. Then ϕ is called regular if $V1 \cap V2 = \emptyset$. A proof is called regular if all its subproofs are regular.

In the following, we will restrict ourselves to proofs of an end-sequent with rectified formulas and to regular proofs, which makes handling of overbinding considerably easier in our considerations.

Definition 5.2.11 (scope of quantifiers [BEL01])**.** Let A be a rectified formula without $\Leftrightarrow$, and $F : (Qx)G$ be a subformula of A. Then for every subformula $(Q'y)H$ of G, we define $(Qx) <_A (Q'y)$; in this case we say $(Q'y)$ is in the scope of (Qx).

The order $<_A$ is not total since for instance in the formula $(\forall x A(x)) \wedge (\forall y B(y))$, x and y are incomparable. The order $<_A$ can be completed to a total order $\prec_A$ by additionally defining $(Qx) \prec_A (Qy)$ for all subformulas $G : B_1 \odot B_2$ of a formula F for any variable x bound in B_1 and any variable y bound in B_2. If $(Qx) \prec (Qy)$, we can say that (Qx) is *left of* (Qy). Since a formula contains only a finite number of quantifiers, there also exists a least element regarding scope, which is called the *leftmost* quantifier.

Definition 5.2.12 (Dominated quantifier)**.** Let α be an occurrence of the formula F containing the quantifier (Qx) which is an ancestor of an occurrence of the formula G and let (Qy) be a quantifier in G which is not in F. Then (Qy) is said to *dominates* (Qx).

This definition corresponds to definition 5.2.11 of the scope relation on quantifiers: the quantifier (Qx) is in the scope of (Qy) if and only if the introduction rule for (Qx) is further below in the proof than the introduction of (Qy).

Skolemization of Proofs

To skolemize a formula F, we use the same transformation as in theory-free predicate logic. Since we want to preserve the validity of a formula, we need to eliminate all occurrences of a strong quantifier in F.[1]

[1]Refutation al calculi like resolution need to preserve satisfiability instead; in this case we need to eliminate the weak quantifiers in F.

Definition 5.2.13 (Structural skolemization [BL99])**.** The function sk maps closed formulas into closed formulas. It is defined by:

$$sk(F) = \begin{cases} F & \text{if F does not contain strong quantifiers} \\ sk(F_{(Qy)}\{y \leftarrow f()\} & \text{if } (Qy) \text{ is not in the scope of a weak quantifier} \\ sk(F_{(Qy)}\{y \leftarrow f(x_1, \ldots, x_n)\}) & \text{if } (Qy) \text{ is in the scope of the weak} \\ & \text{quantifiers } (Q_1 x_1) <_F \ldots <_F (Q_n x_n) \end{cases}$$

where f is a fresh function symbol and $f()$ is a treated as a constant.
$F_{(Qy)}$ is identical to F with the quantor (Qy) dropped.

This definition is only sensible for rectified formulas since otherwise a variable may be captured by a surrounding quantifier for the same variable name and also applying the substitution of the skolem term to the whole formula only works if there is no variable of the same name in a different binding context.

Even though the definition eliminates quantifiers individually, the steps are independent of each other and we may thus eliminate them in parallel in practice, although for argumentational purposes it is easier to assume that the quantifiers are eliminated from left to right. Since we work in the presence of equational theories we should note that a fresh constant or function symbol may also not be introduced by the theory. Since the name of the skolem symbols is non-deterministic, $sk(F)$ is no function. We can remedy this by introducing the same skolem symbol for the same variable. From now we will assume that $sk(F)$ does this consistent renaming.

In classical logic, skolemization is validity preserving i.e. $\vdash F$ if and only if $\vdash sk(F)$ for all formulas F, although we are mainly interested in the only-if part of the equality. Also in [BL99] a method to transform a sequent calculus proof of F into a proof of $sk(F)$ was given. It works by recursively eliminating only those strong quantifiers which appear in the end-sequent.

First we define, how a skolemized (end-)sequent should look like:

Definition 5.2.14 (Skolemization of a sequent)**.** Let $T \equiv (F_1 \wedge \ldots \wedge F_n) \Rightarrow (G_1 \vee \ldots \vee G_m)$ be the formula corresponding to the sequent $S : F_1, \ldots, F_n \vdash_{\mathcal{RE}} G_1, \ldots, G_m$ and $(F'_1 \wedge \ldots \wedge F'_n) \Rightarrow (G'_1 \vee \ldots \vee G'_m)$ be the skolemization $sk(F)$. Then $F'_1, \ldots, F'_n \vdash_{\mathcal{RE}} G_1, \ldots, G'_m$ is the skolemization of S.

Then an occurrence of a strong quantifier is eliminated in the following way: let ρ be the subproof in figure 5.2 (the argumentation for $\exists, l$ is the same) of a larger proof π where $(Qy)P$ is an ancestor of a formula F in the end-sequent in which (Qy) occurs strongly and where $(Qy)P$ is dominated by the weak quantifiers $(Q_1 x_1), \ldots, (Q_n x_n)$. Also let the variables $x_1, \ldots, x_n$ replace the terms $t_1, \ldots, t_n$ in their respective quantifier introduction rules in π. Then the introduction rule of (Qy) which replaces the eigenvariable v can be left out by substituting $f(t_1, \ldots, t_n)$ for both y and v. Since our proof is regular and the end-sequent is rectified, we can apply the substitution on the whole proof ρ. The inferences below the skipped introduction rule can still be repeated like they appear in ρ. During this process the introduction rules for the weak quantifiers will successively replace the terms $t_1, \ldots, t_n$ by variables until after the last one eliminated, the term $f(x_1, \ldots, x_n)$ has taken the role of y in the proof. This is exactly the substitution that would be applied to the end-sequent during its skolemization.

$$\dfrac{\begin{array}{c}(\rho)\\ \Gamma \vdash F, \Delta\end{array}}{\Gamma \vdash \forall y F, \Delta}\ \forall, r \qquad\qquad \begin{array}{c}(\rho\{v \leftarrow f(t_1, \ldots, t_n)\})\\ \Gamma \vdash F\{v \leftarrow f(t_1, \ldots, t_n)\}, \Delta\end{array}$$

where v is the eigenvariable replaced by the introduction of y

Figure 5.2: Elimination of a strong quantifier during skolemization

To allow equational inferences, we add the conversion rules (figure 4.3) to theory-free sequent calculus and show that it may be still applied if the active formula is skolemized.

$$\dfrac{\Gamma, P \vdash \Delta}{\Gamma, Q \vdash \Delta}\ conv, l \qquad \Rightarrow \qquad \dfrac{\Gamma, sk(P) \vdash \Delta}{\Gamma, sk(Q) \vdash \Delta}\ conv, l$$

where $P \approx_{\mathcal{RE}} Q$ where $sk(P) \approx_{\mathcal{RE}} sk(Q)$

Figure 5.3: Substitutivity of the equational conversion rules

Since by definition 5.2.13 the quantifiers are recursively eliminated one by one and we assume that $sk(F)$ is a function, we will prove $P \approx_{\mathcal{E}} Q$ implies $sk(P) \approx_{\mathcal{E}} sk(Q)$ by induction on the number of strong quantifiers and assume the quantifiers are eliminated from left to right. The task becomes also easier since only equational inferences on terms are allowed and thus P and Q have the same logical structure and only differ on the term level.

We also remember that we assume the proof to be regular and that the end-sequent is rectified. Then a strongly quantified variable only occurs in the subtree above to the introduction allowing us to apply substitutions for eigenvariables to the whole proof.

- There is no strong quantifier in F. Then $sk(F) = F$ and the proof π is also its skolemization.
- The strong quantifier (Qy) which is in the scope of the weak quantifiers $(Q_1 x_1), \ldots (Q_n x_n)$ is eliminated. The argumentation for the usual rules of sequent calculus are given in [BL99], so we only need to argument, why the skolemization does destroy neither soundness nor completeness of the conversion rules.

  For this we need to prove that if $P \approx_{\mathcal{RE}} Q$, then also $P_{(Qy)}\{y \leftarrow f(t_1, \ldots, t_n)\} \approx_{\mathcal{RE}} Q_{(Qy)}\{y \leftarrow f(t_1, \ldots, t_n)\}$ where $t_1, \ldots, t_n$ are the terms eliminated down in the proof by the quantifier introduction rules for $Q_1 x_1, \ldots, Q_n x_n$.[2] A propositional rewrite rule could introduce a strong quanti-

[2]This is not entirely exact, since we will substitute partially instantiated skolem terms which will only become the final skolem term $f(x_1, \ldots, x_n)$ in the end. Luckily, the argumentation does not depend on the terms substituted.

fier, so it is important here that we assume $\mathcal{R}$ to be empty. We then only need that to show that $P \approx_{\mathcal{E}} Q$ implies $P_{(Qy)}\{y \leftarrow\!\!\dashv f(t_1, \ldots, t_n)\} \approx_{\mathcal{E}} Q_{(Qy)}\{y \leftarrow\!\!\dashv f(t_1, \ldots, t_n)\}$.

Now we prove $P \approx_{\mathcal{E}} Q$ implies $P_{(Qy)}\sigma \approx_{\mathcal{E}} Q_{(Qy)}\sigma$ for $\sigma = \{y \leftarrow\!\!\dashv f(t_1, \ldots, t_n)\}$ by induction on the complexity of P (or Q, since they have the same structure).

IB:

- Equational theories are are closed under substitution so $s \approx_{\mathcal{E}} t$ implies $s\sigma \approx_{\mathcal{E}} t\sigma$ for any substitution σ.
- Atom formula:
  Suppose for all $i \in \{1, \ldots, n\}$ $s_i \approx_{\mathcal{E}} t_i$ and $A(s_1, \ldots, s_n) \approx_{\mathcal{E}} A(t_1, \ldots, t_n)$. Then the equational theory may not change the predicate symbol, so by the definition of substitution also $A(s_1\sigma, \ldots, s_n\sigma) \approx_{\mathcal{E}} A(t_1\sigma, \ldots, t_n\sigma)$ holds.

IH: $A \approx_{\mathcal{E}} B$ implies $A\sigma \approx_{\mathcal{E}} B\sigma$ for A and B of complexity $\leqslant n$.

IS:

- Negation: By IH we assume $A\sigma \approx_{\mathcal{E}} B\sigma$ for complexity $\leqslant n$. Since the equality $\approx_{\mathcal{E}}$ is on terms only, also $(\neg A)\sigma \approx_{\mathcal{E}} (\neg B)\sigma$.
- Conjunction, disjunction, implication:
  By IH we again assume $A_1\sigma \approx_{\mathcal{E}} B_1\sigma$ and $A_2\sigma \approx_{\mathcal{E}} B_2\sigma$ for complexity $\leqslant n$. For the same reason as above, also $(A_1 \circ A_2)\sigma \approx_{\mathcal{E}} (B_1 \circ B_2)\sigma$ for $\circ \in \{\wedge, \vee, \Rightarrow\}$.
- Weak quantifier:
  By IH we know $A\sigma \approx_{\mathcal{E}} B\sigma$ for complexity $\leqslant n$. Now also $(QxA\sigma) \approx_{\mathcal{E}} (QxB\sigma)$ since we only replace strongly quantified variables in σ.
- Strong quantifier:
  By IH we know $A\sigma \approx_{\mathcal{E}} B\sigma$ for complexity $\leqslant n$.Let (Qz) be the quantifier applied to A (and B). Then we have two cases:
  * (Qz) is not the quantifier (Qy) eliminated in this step: Since we eliminate quantifiers from left to right, (Qy) is left of (Qx) and thus not occurring in A. Then $A_{(Qy)} = A$ and also $x\sigma = x$ since σ only substitutes y to a term different from itself, so from $A \approx_{\mathcal{E}} B$ we conclude $A_{(Qy)}\sigma \approx_{\mathcal{E}} B_{(Qy)}\sigma$.
  * (Qz) is the quantifier (Qy) we want to eliminate: then $A\sigma \approx_{\mathcal{E}} B\sigma$ is already the formula we seek, since Qz is not applied and by having a regular proof with a rectified end-sequent, we know that (Qy) does not occur twice in the formula, so $A_{(Qy)} = A$ and $B_{(Qy)} = B$.

Eigenvariables are not violated in the original process, because elimination rule for the left-most quantifier is the closest one to the end-sequent. So even though a the term t_i ($a \leqslant i \leqslant n$) may now appear sooner in the proof, there are no strong quantifier rules in between whose eigenvariable condition could be destroyed.

With regard to the conversion rules we notice that they only contain a subset of the variables of the original formulas so eigenvariable conditions can not be violated by them.

Cut Extension

The cut-extension T_{cut} is a proof transformation which replaces the rule for cut over the formula A by an application of the implication left rule for the formula $A \Rightarrow A$. This reduces the task of cut-elimination to one of tautology elimination in the antecedent of the end-sequent of a proof.[3] More formally, the proof of the end-sequent $\Gamma \vdash_{\mathcal{RE}} \Delta$ with n cuts is recursively rewritten, starting from the leaves. Every rule is kept as it is, only the cut-rule is replaced by the implication on the left (see figure 5.4). To avoid variable captures, we need to universally close $A \Rightarrow B$ over the free variables $\overline{x}$ in the formula.

$$\dfrac{\Gamma_1 \vdash_{\mathcal{RE}} \Delta_1, P \qquad \Gamma_2, Q \vdash_{\mathcal{RE}} \Delta_2}{\Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_1, \Delta_2}\ cut \qquad \Rightarrow \qquad \dfrac{\dfrac{\Gamma_1 \vdash_{\mathcal{RE}} \Delta_1, P \qquad \Gamma_2, Q \vdash_{\mathcal{RE}} \Delta_2}{\Gamma_1, \Gamma_2, P \Rightarrow Q \vdash_{\mathcal{RE}} \Delta_1, \Delta_2}\ \Rightarrow, l}{\Gamma_1, \Gamma_2, \forall \overline{x}(P \Rightarrow Q) \vdash_{\mathcal{RE}} \Delta_1, \Delta_2}\ \forall, l$$

Figure 5.4: Replacement of the Cut Rule by Implication

Since for the cut-rule to apply, $P \approx_{\mathcal{RE}} Q$ must hold, it is easy to see that $\forall \overline{x}(P \Rightarrow Q)$ is valid and the rewritten proof new has the end-sequent $\Gamma, \forall \overline{x_1}(A_1 \Rightarrow B_1), \ldots, \forall \overline{x_n}(A_n \Rightarrow B_n) \vdash_{\mathcal{RE}} \Delta$. To handle the tautologies more easily, we can build one valid formula by repeated application of the conjunction rule on the left and get a proof of $\Gamma, \forall \overline{x_1}(A_1 \Rightarrow B_1) \wedge \ldots \wedge \forall \overline{x_n}(A_n \Rightarrow B_n) \vdash_{\mathcal{RE}} \Delta$.

The Characteristic Clause Set

Definition 5.2.15 ($CL(\psi, \alpha)$, analog to [BL00]). Let ψ be a cut-free proof of the sequent S (in sequent calculus modulo) and α be an occurrence of a formula in S. Then the set of *characteristic clauses* $CL(\psi, \alpha)$ is inductively defined:

Let η be the occurrence of an initial sequent $P \vdash_{\mathcal{RE}} Q$ in ϕ, where $P \approx_{\mathcal{RE}} Q$ and let η_1 (η_2) be the occurrence of P (Q). If neither η_1 nor η_2 are ancestors of α, then $\mathcal{C}_\eta = \{\vdash_{\mathcal{RE}}\}$. If both, η_1 and η_2 are ancestors of α, then $\mathcal{C}_\eta = \emptyset$. If only η_1 is ancestor of α, then $\mathcal{C}_\eta = \{P \vdash_{\mathcal{RE}}\}$, if only η_2 is ancestor of α, then $\mathcal{C}_\eta = \{\vdash_{\mathcal{RE}} Q\}$.

[3] The introduction of the cut-extension is somewhat artificial in the sense that we introduce a formula which will later be eliminated again. Newer formulations of CERES like [BL06, HLWP08, LB11] drop this construction, but for easier understanding, the construction was kept.

For an occurrence η of an initial sequent $R \vdash_{\mathcal{RE}}$ with $R \approx_{\mathcal{RE}} \bot$, there are only two possibilities: if R is ancestor of α define $\mathcal{C}_\eta = \emptyset$, if R is not ancestor of α define $\mathcal{C}_\eta = \{\vdash_{\mathcal{RE}}\}$.

Assuming the clause sets $\mathcal{C}_\kappa$ with depth$(\kappa) \leqslant k$ have been constructed for all sequent-occurrences κ in ψ. Then the sequent-occurrence λ with depth$(\lambda) = k + 1$ is defined as follows:

1. Unary Rule: Let λ be the consequent of the occurrence μ, then $\mathcal{C}_\lambda = \mathcal{C}_\mu$.
2. Binary Rule: Let λ be the consequent of the occurrences μ_1 and μ_2 of the binary rule X.
   a) Auxiliary formulas of X are ancestors of α: $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \cup \mathcal{C}_{\mu_2}$.
   b) Auxiliary formulas of X are not ancestors α: $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$ where $\{\overline{P_1} \vdash_{\mathcal{RE}} \overline{Q_1}, \ldots, \overline{P_n} \vdash_{\mathcal{RE}} \overline{Q_n}\} \otimes \{\overline{R_1} \vdash_{\mathcal{RE}} \overline{T_1}, \ldots, \overline{R_m} \vdash_{\mathcal{RE}} \overline{T_m}\} =$ $\{\overline{P_i}, \overline{R_j} \vdash_{\mathcal{RE}} \overline{Q_i}, \overline{T_j} | 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant m\}$.

Now $CL(\psi, \alpha) = \mathcal{C}_\nu$, where ν is the occurrence of the end-sequent.

Lemma 5.2.16 (refutability of $CL(\psi, \alpha)$)**.** In a cut-free proof ψ, the characteristic clause set $CL(\psi, \alpha)$ is refutable if α is a left occurrence of a tautology.

Proof. This proof mirrors the one in [BL00], since it only uses information about the structure of LK proofs, which is the same for sequent calculus modulo. It works by showing that from the characteristic clause set $CL(\psi, \alpha)$ the sequent $B \vdash_{\mathcal{RE}}$ is derivable in sequent calculus modulo, where B is the formula at position α. Since sequent calculus is sound with respect to LK, $\mathcal{T}, B \vdash$ must be provable in LK and by Theorem 4.3.2, there also exists an ENAR refutation of $CL(\psi, \alpha)$.

We prove for all sequent occurrences λ, $\mathcal{C}_\lambda \vdash_{\mathbf{RE}} anc(\lambda, \alpha)$ by structural induction on the depth of the proof.

IB:
The only possible rules of sequent calculus modulo at depth 0 are axiom and bottom introduction[4].

We first look at axiom introduction of $P \approx_{\mathcal{RE}} Q$. Only if both P and Q are ancestors of α ($anc(\lambda, \alpha) = P \vdash_{\mathcal{RE}} Q$), the corresponding clause set is $\mathcal{C}_\lambda = \emptyset$, $P \vdash_{\mathcal{RE}} Q$ is a tautology provable from the empty set of clauses. In the other cases $\mathcal{C}_\lambda$ is defined as $anc(\lambda, \alpha)$, so the clause set trivially proves the ancestor.

Now we look at an occurrence λ of a bottom introduction for a sequent $R \vdash_{\mathcal{RE}}$ with $R \approx_{\mathcal{RE}} \bot$. If R is ancestor of α, we need to prove $R \vdash_{\mathcal{RE}}$ from the empty set, which works because the formula is valid. If R is not ancestor of α, $\vdash_{\mathcal{RE}}$ is provable from $\{\vdash_{\mathcal{RE}}\}$.
IH: suppose $\mathcal{C}_\lambda \vdash_{\mathbf{RE}} anc(\lambda, \alpha)$ for all sequent occurrences λ with $depth(\lambda) \leqslant k$.

[4]see figure 4.1

IS: Let λ be a sequent occurrence in ψ at $depth(\lambda) = k + 1$ and let ψ_λ be the proof of $anc(\lambda, \alpha)$. We now have to make a case distinction on the arity of the inference step and whether it is done on ancestors of α or not:

1. unary inference: here the general structure of ψ_λ with the last inference rule X is:

$$\frac{\begin{matrix}(\chi)\\ \Lambda, \Gamma \vdash_{\mathcal{RE}} \Delta, \Pi\end{matrix}}{\Lambda', \Gamma' \vdash_{\mathcal{RE}} \Delta', \Pi'}\ X$$

   with $anc(\lambda, \alpha) = \Lambda' \vdash_{\mathcal{RE}} \Pi'$ and $anc(\mu, \alpha) = \Lambda \vdash_{\mathcal{RE}} \Pi$ for the predecessor occurrence μ of λ. Since $depth(\mu) = k$, we can apply the induction hypothesis $\mathcal{C}_\mu \vdash_{\mathbf{RE}} \Gamma \vdash_{\mathcal{RE}} \Pi$ and obtain a proof ρ for the ancestors of μ from its corresponding clause set $\mathcal{C}_\mu$. If the inference took place on an non-ancestor of a cut formula, then $\Lambda = \Lambda'$ and $\Pi = \Pi'$ so no rule application is required at all. If the rule X operates on an ancestor, then X now allows to infer $\Lambda' \vdash_{\mathcal{RE}} \Pi'$ from $\mathcal{C}_\mu$, which is equal to $\mathcal{C}_\lambda$ by definition 5.2.15:

$$\frac{\begin{matrix}(\rho)\\ \Lambda \vdash_{\mathcal{RE}} \Pi\end{matrix}}{\Lambda' \vdash_{\mathcal{RE}} \Pi'}\ X$$

   For the rules $\forall, r$ and $\exists, l$ the eigenvariable condition must hold. This is the case, because in χ the eigenvariable condition is not violated, which means that the fresh variable v introduced does not occur in the formulas $(\Lambda \cup \Gamma \cup \Delta \cup \Pi) \backslash \{F\}$ where F is the principal formula of X. But then v does also not occur in $(\Lambda \cup \Pi) \backslash \{F\}$.

   So $\mathcal{C}_\lambda \vdash_{\mathbf{RE}} \Lambda' \vdash_{\mathcal{RE}} \Pi'$ which is $anc(\lambda, \alpha)$.

2. binary inference:

   a) binary inference on ancestors: The situation here is similar to the one before, but with two sources. Let the general structure of the inference of ψ_λ be:

$$\frac{\begin{matrix}(\chi_1)\\ \Lambda_1, \Gamma_1 \vdash_{\mathcal{RE}} \Delta_1, \Pi_1\end{matrix} \qquad \begin{matrix}(\chi_2)\\ \Lambda_2, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_2, \Pi_2\end{matrix}}{\Lambda_1', \Lambda_2', \Gamma_1', \Gamma_2' \vdash_{\mathcal{RE}} \Delta_1', \Delta_2', \Pi_1', \Pi_2'}\ X$$

   with μ_1 and μ_2 denoting the predecessor occurrences of λ and $anc(\lambda, \alpha) = \Gamma_1', \Gamma_2' \vdash_{\mathcal{RE}} \Pi_1', \Pi_2'$, $anc(\mu_1, \alpha) = \Gamma_1' \vdash_{\mathcal{RE}} \Pi_1'$, $anc(\mu_2, \alpha) = \Gamma_2' \vdash_{\mathcal{RE}} \Pi_2'$ being the ancestors of λ, μ_1 and μ_2. Again by induction hypothesis, there are proofs ρ_1 and ρ_2 from $\mathcal{C}_{\mu_1}$ and $\mathcal{C}_{\mu_2}$ of the ancestors of μ_1 and μ_2 which can be joined together via X:

$$\frac{\begin{matrix}(\rho_1)\\ \Gamma_1 \vdash_{\mathcal{RE}} \Pi_1\end{matrix} \qquad \begin{matrix}(\rho_2)\\ \Gamma_2 \vdash_{\mathcal{RE}} \Pi_2\end{matrix}}{\Gamma_1', \Gamma_2' \vdash_{\mathcal{RE}} \Pi_1', \Pi_2'}\ X$$

   so indeed $\mathcal{C}_{\mu_1} \cup \mathcal{C}_{\mu_2} = \mathcal{C}_\lambda \vdash_{\mathbf{RE}} anc(\lambda, \alpha)$.

b) binary inference on non-ancestors:

Since the inference takes place on non-ancestors, the formulas to be proven are the same. Then the general structure of the inference of ψ_λ is:

$$\dfrac{\begin{matrix}(\chi_1) \\ \Lambda_1, \Gamma_1 \vdash_{\mathcal{RE}} \Delta_1, \Pi_1\end{matrix} \qquad \begin{matrix}(\chi_2) \\ \Lambda_2, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_2, \Pi_2\end{matrix}}{\Lambda_1', \Lambda_2', \Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Delta_1', \Delta_2', \Pi_1, \Pi_2}\, X$$

with μ_1 and μ_2 denoting the predecessor occurrences of λ and $anc(\lambda, \alpha) = \Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Pi_1, \Pi_2$, $anc(\mu_1, \alpha) = \Gamma_1 \vdash_{\mathcal{RE}} \Pi_1$, $anc(\mu_2, \alpha) = \Gamma_2 \vdash_{\mathcal{RE}} \Pi_2$ being the ancestors of λ, μ_1 and μ_2. Also by induction hypothesis, we again get the proofs ρ_1 and ρ_2 of $\Gamma_1 \vdash_{\mathcal{RE}} \Pi_1$ and $\Gamma_2 \vdash_{\mathcal{RE}} \Pi_2$.

By looking at Definition 5.2.15, we need to prove $\mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2} \vdash_{\mathbf{RE}} \Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Pi_1, \Pi_2$. For this, for each clause in $CL(\mu_2, \alpha)$, we add this clause to ρ_1 by weakening and are then able to use the enriched proofs instead of the introduction of the clause to form a combined proof.

So let $D_i = \{\overline{P_1}, \overline{R_i} \vdash_{\mathcal{RE}} \overline{Q_1}, \overline{T_i}; \ldots; \overline{P_n}, \overline{R_i} \vdash_{\mathcal{RE}} \overline{Q_n}, \overline{T_i}\}$ with $i \in \{1, \ldots, m\}$ be those parts of $\mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$ that are combined with clause i from $\mathcal{C}_{\mu_2}$. We also notice that $\bigcup_{i \in \{1,\ldots,m\}} D_i = \mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$.

For every i each axiom introduction $\overline{P_j} \vdash_{\mathcal{RE}} \overline{Q_j}$ with $1 \leqslant j \leqslant n$ of ρ_1 can be extended to an introduction of $\overline{P_j}, \overline{R_i} \vdash_{\mathcal{RE}} \overline{Q_j}, \overline{T_i}$. By repeating the rule applications of ρ_1 we can prove the extended end sequents S_i of $\rho_1[\overline{R_i} \vdash_{\mathcal{RE}} \overline{T_i}]$ which have the structure $\overline{R_i}, \ldots, \overline{R_i}, \Gamma_1 \vdash_{\mathcal{RE}} \Pi_1, \overline{T_i}, \ldots, \overline{T_i}$ and can be reduced to $\overline{R_i}, \Gamma_1 \vdash_{\mathcal{RE}} \Pi_1, \overline{T_i}$ via some contractions depending on the number of binary rules in ρ_1. Since we assume ρ_1 and ρ_2 to be regular, no eigenvariable condition is violated by this procedure and the resulting proofs are correct. In the following, we denote the proofs by σ_i.

Now we can use the proofs σ_i as initial sequents instead of the original $R_i \vdash_{\mathcal{RE}} T_i$. Again by repeating the rule applications of ρ_2 we get an end sequent $\Gamma_1, \ldots, \Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Pi_1, \ldots, \Pi_1, \Pi_2$ which can be reduced to $\Gamma_1, \Gamma_2 \vdash_{\mathcal{RE}} \Pi_1, \Pi_2$ via contraction.

So indeed $\mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2} = \mathcal{C}_\lambda \vdash_{\mathbf{RE}} anc(\lambda, \alpha)$.

□

Example

In the example in figure 5.1 (which is is already a cut-extension), with α denoting the occurrence of the formula $\forall x(E(x) \Rightarrow E(x + (1 + 1)))$ collecting the formulas going into cuts, the characteristic clause set is $\{E(x + y) \vdash_{\mathcal{RE}} E((x + 1) + (y + 1)); \vdash_{\mathcal{RE}} E(c + c); E((c + c) + (1 + 1)) \vdash_{\mathcal{RE}}\}$: Since the top left inference is on non-ancestors of α, the sequents $E(x+y) \vdash_{\mathcal{RE}}$ and $\vdash_{\mathcal{RE}} E((x+1)+(y+1))$

are merged. All other binary inferences take place on ancestors of α and are thus combined via set-union. The tree in figure 5.5 shows the derivation.[5]

$$\dfrac{\dfrac{\mathrm{E(x+y)} \vdash_{\mathcal{RE}} \qquad \vdash_{\mathcal{RE}} \mathrm{E((x+1)+(y+1))}}{\mathrm{E(x+y)} \vdash_{\mathcal{RE}} \mathrm{E((x+1)+(y+1))}} \otimes \qquad \dfrac{\vdash_{\mathcal{RE}} \mathrm{E(c+c)} \qquad \mathrm{E((c+c)+(1+1))} \vdash_{\mathcal{RE}}}{\vdash_{\mathcal{RE}} \mathrm{E(c+c)};\ \mathrm{E((c+c)+(1+1))} \vdash_{\mathcal{RE}}} \cup}{\mathrm{E(x+y)} \vdash_{\mathcal{RE}} \mathrm{E((x+1)+(y+1))};\ \vdash_{\mathcal{RE}} \mathrm{E(c+c)};\ \mathrm{E((c+c)+(1+1))} \vdash_{\mathcal{RE}}} \cup$$

Figure 5.5: Example of a characteristic clause set calculation

Figure 5.6 now shows how to construct a resolution refutation from the characteristic clause set.

$$\dfrac{\vdash_{\mathcal{RE}} E(c+c) \qquad \dfrac{E(x+y) \vdash_{\mathcal{RE}} E(x+1) + (y+1) \qquad E((c+c) + (1+1) \vdash_{\mathcal{RE}}}{E(c+c) \vdash_{\mathcal{RE}}} \ \sigma = \{x \mapsfrom 0; y \mapsfrom c+c\}}{\vdash_{\mathcal{RE}}}$$

Figure 5.6: Resolution refutation of the example characteristic clause set

Projections

One of the central ideas of CERES is to mirror the resolution refutation of the characteristic clause set with proofs of the end-sequent without the cut-extension but additionally of one clause from the characteristic clause set. These enriched proofs are called projections and are constructed by extracting that part of the proof which contributes to the additional clause while replacing the remaining branches of the proof by appropriate weakenings.

Lemma 5.2.17. Let ψ be a skolemized, cut-free proof of a sequent $S : A, \Gamma \vdash_{\mathcal{RE}} \Delta$, where A is valid and α is the occurrence of A in S. Let $C : \overline{P} \vdash_{\mathcal{RE}} \overline{Q}$ be a clause in $CL(\psi, \alpha)$. Then there exists a cut-free proof $\psi[C]$ of $\overline{P}, \Gamma \vdash_{\mathcal{RE}} \Delta, \overline{Q}$ with $l(\psi[C]) \leqslant l(\psi)$.

We inductively construct a proof that proves $\overline{anc}(\lambda, \alpha) \circ C$ for each $C \in \mathcal{C}_\lambda$. For the end-sequent, this is exactly what we want to show: $\overline{P}, \Gamma \vdash_{\mathcal{RE}} \Delta, \overline{Q}$.

IB:

For an axiom introduction $S : P \vdash_{\mathcal{RE}} Q$ with $P \approx_{\mathcal{RE}} Q$ at position λ, if both formulas are ancestors of α, then it will not turn up in any projection. In the other three cases, $\mathcal{C}_\lambda$ is defined as $anc(\lambda, \alpha)$, so $\overline{anc}(\lambda, \alpha) \circ \mathcal{C}_(\lambda) = \overline{anc}(\lambda, \alpha) \circ anc(\lambda, \alpha) = S$ is the whole sequent, which is valid.

[5]Newer formulations of CERES like [BHL+08, Wel10, Pal09, LB11] drop the notion of characteristic clause set and replace it by clause terms, which are trees with sequents as leafs and the merge and union operation as inner nodes. If the inner nodes are labeled accordingly, figure 5.5 also represents the clause term.

For the bottom introduction $S : R \vdash_{\mathcal{RE}}$ with $R \approx_{\mathcal{RE}} \bot$, there are two cases, one of them does not prove any non-ancestor, so only the case where there are no ancestors remains. But then again, $R \vdash_{\mathcal{RE}} \circ \vdash_{\mathcal{RE}} = R \vdash_{\mathcal{RE}}$, which is valid.
IH: suppose $\psi[\mathcal{C}_\mu]$ is constructed for all proofs ψ of length $\leqslant k$ with the end-sequent occurrence μ.

Now we have to make a case distinction on the construction of $\mathcal{C}$:

1. Unary rule:
   Let the inference have the following shape, with $length(\mu) = k$, the formulas $\overline{R}$, $\overline{R}'$,$\overline{T}$ and $\overline{T}'$ denoting ancestors of α and the formulas Π, Π', Λ and Λ' denoting non-ancestors of α.

$$\frac{\begin{array}{c}(\rho)\\ (\mu)\overline{R}', \Pi' \vdash_{\mathcal{RE}} \overline{T}', \Lambda'\end{array}}{(\lambda)\overline{R}, \Pi \vdash_{\mathcal{RE}} \overline{T}, \Lambda}$$

   By IH, we can assume $\rho[C]$ is already constructed for all $C \in \mathcal{C}_\mu$. From the definition of $\mathcal{C}_,$ we also know that the clause set does not change in unary inferences, so $\mathcal{C}_\lambda = \mathcal{C}_\mu$. Nonetheless the construction of the projection differs whether the rule is applied to ancestors of α or not:

   a) Inference on ancestors:
   In this case, the non-ancestors don't change, so $\rho[C]$ is already the proof we need:

$$\frac{\begin{array}{c}(\rho)\\ \overline{R}', \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{T}'\end{array}}{\overline{R}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{T}} X \quad \Rightarrow \quad \begin{array}{c}(\rho[\overline{P} \vdash_{\mathcal{RE}} \overline{Q}])\\ \overline{P}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{Q}\end{array}$$

   Also by IH we we know that $length(\rho[C]) \leqslant length(\rho)$, but since we did not change the proof, trivially also $length(\rho[C]) \leqslant length(\rho) + 1$.

   b) Inference on non-ancestors:
   Here only the non-ancestors change, so by just repeating the inference, we get a proof of $\overline{P}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{T}$:

$$\frac{\begin{array}{c}(\rho)\\ \overline{R}, \Pi' \vdash_{\mathcal{RE}} \Lambda', \overline{T}\end{array}}{\overline{R}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{T}} X \quad \Rightarrow \quad \frac{\begin{array}{c}(\rho[\overline{P} \vdash_{\mathcal{RE}} \overline{Q}])\\ \overline{P}, \Pi' \vdash_{\mathcal{RE}} \Lambda', \overline{Q}\end{array}}{\overline{P}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{Q}} X$$

   Since C can contain variables, an application of the $\forall, r$ and $\exists, l$ rule could now violate their eigenvariable conditions. This is the reason why we required ψ to be skolemized, where those rules do not occur anymore.

   By IH we we know that $length(\rho[C]) \leqslant length(\rho)$. We apply exactly one rule on both ρ and $\rho[C]$, so the inequality still holds.

2. Binary rule:
   Let the binary rule have the following shape, with $length(\rho_i) \leqslant k$, $\mathcal{C}_\lambda = \overline{P} \vdash_{\mathcal{RE}} \overline{Q}$, $\overline{anc}(\lambda, \alpha) = \Pi \vdash_{\mathcal{RE}} \Lambda$ and $anc(\lambda, \alpha) = \overline{R} \vdash_{\mathcal{RE}} \overline{T}$:

$$\dfrac{\begin{array}{c}(\rho_1)\\ (\mu_1)\overline{R_1}, \Pi_1 \vdash_{\mathcal{RE}} \overline{T_1}, \Lambda_1\end{array} \qquad \begin{array}{c}(\rho_2)\\ (\mu_2)\overline{R_2}, \Pi_2 \vdash_{\mathcal{RE}} \overline{T_2}, \Lambda_2\end{array}}{(\lambda)\Gamma \vdash_{\mathcal{RE}} \Delta}\, X$$

   a) On ancestors:
      From the definition, we know that $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \cup \mathcal{C}_{\mu_2}$. Then either C is element of $\mathcal{C}_{\mu_1}$ or $\mathcal{C}_{\mu_2}$. We can w.l.o.g. assume $C \in \mu_1$. By IH we can construct $\rho_1[C]$. Since we are not interested in proving clauses in C_2, we do not need ρ_2 but can add the formulas Π_2 and Λ_2 by weakening instead.

$$\dfrac{\begin{array}{c}(\rho_1)\\ \overline{R_1}, \Pi_1 \vdash_{\mathcal{RE}} \Lambda_1, \overline{T_1}\end{array} \qquad \begin{array}{c}(\rho_2)\\ \overline{R_2}, \Pi_2 \vdash_{\mathcal{RE}} \Lambda_2, \overline{T_2}\end{array}}{\overline{R}, \Pi_1, \Pi_2 \vdash_{\mathcal{RE}} \Lambda_1, \Lambda_2, \overline{T}}\, X \quad \Rightarrow \quad \dfrac{\begin{array}{c}(\rho_i[\overline{P_i} \vdash_{\mathcal{RE}} \overline{Q_i}])\\ \overline{P_i}, \Pi_i \vdash_{\mathcal{RE}} \Lambda_i, \overline{Q_i}\end{array}}{\overline{P_i}, \Pi_1, \Pi_2 \vdash_{\mathcal{RE}} \Lambda_1, \Lambda_2, \overline{Q_i}}\, w, l+r$$

      We know from the IH that $length(\rho_i[C_i]) \leqslant length(\rho_i)$, so the proof of the sequent-occurrence λ has then $length(\rho_1) + length(\rho_2) + 1$, whereas the projection has a length of $length(\rho_1) + n$, where n is the number of formulas in $\Pi_2 \vdash_{\mathcal{RE}} Q_2$. Since the introduction rules do not have a context, there is no way to introduce n formulas in a proof of length $\leqslant n$. So the projection has a lesser length than the proof of λ.

   b) On non-ancestors:
      In this case, the parts of C do come from both ρ_1 and ρ_2: by definition of $\mathcal{C}$, $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$. Since $C \in \mathcal{C}_\Lambda$, there are clauses $C_1 \in \mathcal{C}_{\mu_1}$ and $C_2 \in \mathcal{C}_{\mu_2}$ such that $C_1 \circ C_2 = \overline{P_1} \vdash_{\mathcal{RE}} \overline{Q_1} \circ \overline{P_2} \vdash_{\mathcal{RE}} \overline{Q_2}$. Applying the IH, we can construct the proofs $\rho_1[C_1]$ and $\rho_2[C_2]$. Since the rule is applied on non-ancestors, we can repeat it. Because we have a multiplicative sequent calculus, the ancestor formulas are merged, which is exactly, what we want to achieve to prove $\Pi \vdash_{\mathcal{RE}} \Lambda[C]$:

$$\dfrac{\begin{array}{c}(\rho_1)\\ \overline{R}, \Pi_1 \vdash_{\mathcal{RE}} \Lambda_1, \overline{T}\end{array} \qquad \begin{array}{c}(\rho_2)\\ \overline{R}, \Pi_2 \vdash_{\mathcal{RE}} \Lambda_2, \overline{T}\end{array}}{\overline{R}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{T}}\, X \quad \Rightarrow \quad \dfrac{\begin{array}{c}(\rho_1[\overline{P_1} \vdash_{\mathcal{RE}} \overline{Q_1}])\\ \overline{P_1}, \Pi_1 \vdash_{\mathcal{RE}} \Lambda_1, \overline{Q_1}\end{array} \qquad \begin{array}{c}(\rho_2[\overline{P_2} \vdash_{\mathcal{RE}} \overline{Q_2}])\\ \overline{P_2}, \Pi_2 \vdash_{\mathcal{RE}} \Lambda_2, \overline{Q_2}\end{array}}{\overline{P_1}, \overline{P_2}, \Pi \vdash_{\mathcal{RE}} \Lambda, \overline{Q_1}, \overline{Q_2}}\, X$$

      Again, the length of of the proof of the sequent-occurrence λ is $length(\rho_1) + length(\rho_2) + 1$ and by IH we know that $length(\rho_i[C_i]) \leqslant length(\rho_i)$. Now the length of our constructed projection is $length(\rho_1[C_1]) + length(\rho_2[C_2]) + 1 \leqslant length(\rho_1) + length(\rho_2) + 1$.

Example

Since the characteristic clause set from figure 5.5 contains three clauses, there are also three projections which are given in figure 5.7.

Projection to $E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1))$:

$$\dfrac{\dfrac{E(x+y) \vdash_{\mathcal{RE}} E(x+y) \qquad E((x+1)+(y+1)) \vdash_{\mathcal{RE}} E((x+1)+(y+1))}{E(x+y) \Rightarrow E((x+1)+(y+1)), E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1))}\Rightarrow,l}{\forall x(E(x) \Rightarrow E(x+(1+1))), E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1))}\forall,l$$

Projection to $\vdash_{\mathcal{RE}} E(c+c)$:

$$\dfrac{\dfrac{E(c+c) \vdash_{\mathcal{RE}} E(c+c)}{E(c+c) \vdash_{\mathcal{RE}} E(c+c), E(c+(1+(1+c)))}w,r}{E(c+c), \forall x(E(x) \Rightarrow E(x+(1+1))) \vdash_{\mathcal{RE}} E(c+c), E(c+(1+(1+c)))}w,l$$

Projection to $E((c+1)+(c+1)) \vdash_{\mathcal{RE}}$:

$$\dfrac{\dfrac{E((c+c)+(1+1)) \vdash_{\mathcal{RE}} E(c+(1+(1+c)))}{E(c+c), E((c+c)+(1+1)) \vdash_{\mathcal{RE}} E(c+(1+(1+c)))}w,l}{E(c+c), \forall x(E(x) \Rightarrow E(x+(1+1))), E((c+c)+(1+1)) \vdash_{\mathcal{RE}} E(c+(1+(1+c)))}w,l$$

Figure 5.7: Example for projections

Inserting the Projections into the Resolution Proof

We have already seen how to translate a resolution proof into sequent calculus modulo, but in the case of projections, our clause sequents have an additional context $\Gamma \vdash_{\mathcal{RE}} \Delta$ and instead of the empty sequent we want to derive $\Gamma \vdash_{\mathcal{RE}} \Delta$. Now changing the resolution translation from figure 4.6 to one with context (figure 5.8) leaves us to check whether this could introduce any problems. Since our clauses are ground, they cannot interfere with the free variables in $\Gamma \vdash_{\mathcal{RE}} \Delta$, but the multiplicative nature of the rules of sequent calculus modulo doubles the context after each resolution step. This can be remedied with a series of contractions after each resolution translation. By the simulation of the resolution steps, we reach a proof with only atomic cuts of $\Gamma \vdash_{\mathcal{RE}} \Delta[\vdash_{\mathcal{RE}}] \equiv \Gamma \vdash_{\mathcal{RE}} \Delta$ in the end, which is what we wanted to achieve.

$$\dfrac{\dfrac{\dfrac{\Gamma, \overline{Q}\_\theta \vdash_{\mathcal{RE}} P_1\theta, \ldots, P_n\theta, \overline{Q}_+\theta, \Delta}{\Gamma, \overline{Q}\_\theta \vdash_{\mathcal{RE}} P_1\theta, \overline{Q}_+\theta, \Delta}c,lr \qquad \dfrac{\Gamma, R_1\theta, \ldots, R_n\theta, \overline{S}\_\theta \vdash_{\mathcal{RE}} \overline{S}_+\theta, \Delta}{\Gamma, R_1\theta, \overline{S}\_\theta \vdash_{\mathcal{RE}} \overline{S}_+\theta, \Delta}c,lr}{\Gamma, \Gamma, \overline{Q}\_\theta, \overline{S}\_\theta \vdash_{\mathcal{RE}} \overline{Q}_+\theta, \overline{S}_+\theta, \Delta, \Delta}cut}{\Gamma, \overline{Q}\_\theta, \overline{S}\_\theta \vdash_{\mathcal{RE}} \overline{Q}_+\theta, \overline{S}_+\theta, \Delta}c,lr$$

Figure 5.8: Translation of resolution steps with projections into sequent calculus modulo

Example

Since the endsequent repeats itself often in the proof, we denote its antecedent by $\Gamma \equiv E(c+c), \forall x(E(x) \Rightarrow E(x+(1+1)))$ and its successfully by $\Delta \equiv E(c+(1+(1+c)))$. From our resolution proof we get the substitution $\sigma = \{x \leftarrow\!\!\dashv 0, y \leftarrow\!\!\dashv c + c\}$, which already creates ground instances of the clauses in the characteristic clause set. The combined proof in ACNF can be seen in figure 5.9; ρ denotes the original proof (figure 5.1).

$$\dfrac{\begin{array}{c}\rho[\vdash_{\mathcal{RE}} E(c+c)]\sigma\\ \Gamma \vdash_{\mathcal{RE}} E(c+c), \Delta\end{array} \qquad \dfrac{\begin{array}{c}\rho[E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1))]\sigma\\ \Gamma, E(0+(c+c)) \vdash_{\mathcal{RE}} E((c+1)+(c+1)), \Delta\end{array} \qquad \begin{array}{c}\rho[E((c+c)+(1+1) \vdash_{\mathcal{RE}}]\sigma\\ \Gamma, E((c+c)+(1+1)) \vdash_{\mathcal{RE}} \Delta\end{array}}{\Gamma, E(0+(c+c)) \vdash_{\mathcal{RE}} \Delta}\,cut}{\Gamma \vdash_{\mathcal{RE}} \Delta}\,cut$$

Figure 5.9: Example in ACNF after combination of the projections

CHAPTER 6

Implementation and Experiments

6.1 Overview

To experiment with CERES modulo, a specific equational theory needed to be chosen. AC and ACU were selected because they are among the best studied equational theories. Also the long range goal is to analyze Fürstenberg's proof of the infinity of primes in CERES modulo as was already done with CERES [BHL$^+$08] since it contains many arithmetical inferences which could profit from the integration of ACU. Also an implementation was started as part of the GAP project [LWWP$^+$] which contains the new implementation of the CERES method based on typed lambda calculus.

6.2 Examples and Comparison to CERES

The proof of the infinity of primes is huge - even the first instance of the schema is a proof tree of roughly 500 nodes - so extracting the characteristic clause set by hand is not an effective option. A proof over Fibonacci numbers showing that $\forall x fib(x) + 1 < fib(x + 3)$ should have been a simpler example. It sounded reasonably small, but some rules intuitively used by humans had to be proved as lemmas and made the whole proof longer than expected. The principle $\forall x \forall y \forall n(x < y \Rightarrow x + n < y + n)$ i.e. the monotonicity of the lesser-than relation on natural numbers is an example for this. Although the intuitive lemmas provide many non-trivial cuts, the increase of the size of the whole proof makes it too large for a manual extraction of the characteristic clause set.

The third example tried was the one from section 5.2 proving that four is an even number containing the cut lemma that for every even number x, the number x+4 is also an even. For a comparison with CERES for LKDe [BHL$^+$06], the deduction modulo proof can be transformed to a sequent calculus proof with the theory AC by theorem 4.2.3. Its characteristic clause set then additionally contains the paramodulants proving the equalities. The characteristic clause set is given in figure 6.1 and the resolution refutation is shown in figure 6.2. The resulting cut-free proof in ACNF (figure 6.3) is larger than the corresponding proof obtained by CERES modulo because it also needs the projections to the equational axioms. For the refutation no equational inferences were needed in sequent calculus modulo. On the other hand equational resolution generates more intermediary clauses since the minimal complete set of unifiers is usually considerably larger than the single unifier theory-free unification generates.

$CL(\phi, \alpha) \;=\; \{E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1)); \vdash_{\mathcal{RE}} E(c+c); E(c+(1+(1+c))) \vdash_{\mathcal{RE}}\} \;\cup$
$\{\vdash_{\mathcal{RE}} C_4 = C_3; \vdash_{\mathcal{RE}} C_3 = C_2; \vdash_{\mathcal{RE}} C_2 = C_1\} \cup$
$\{\vdash_{\mathcal{RE}} C_5 = C_6; \vdash_{\mathcal{RE}} C_6 = C_7; \vdash_{\mathcal{RE}} C_7 = C_8, \vdash_{\mathcal{RE}} C_8 = C_9\}$

shorthand notation:
$C_1 \equiv c+(1+(1+c)), C_2 \equiv c+((1+1)+c), C_3 \equiv c+(c+(1+1)), C_4 \equiv (c+c)+(1+1)$
$C_5 \equiv (c+1)+(y+1), C_6 \equiv c+(1+(y+1)), C_7 \equiv c+((y+1)+1), C_8 \equiv c+(y+(1+1)),$
$C_9 \equiv (c+y)+(1+1).$

Figure 6.1: Characteristic clause set for the example in CERES with LKDe

$$\dfrac{\dfrac{\vdash_{\mathcal{RE}} C_1 = C_2 \qquad \vdash_{\mathcal{RE}} C_2 = C_3}{\vdash_{\mathcal{RE}} C_1 = C_3}\,= \qquad \vdash_{\mathcal{RE}} C_3 = C_4}{\vdash_{\mathcal{RE}} C_1 = C_4}\,=$$
$$\rho_1$$

$$\dfrac{\dfrac{\vdash_{\mathcal{RE}} C_5 = C_6 \qquad \vdash_{\mathcal{RE}} C_6 = C_7}{\vdash_{\mathcal{RE}} C_5 = C_7}\,= \qquad \dfrac{\vdash_{\mathcal{RE}} C_7 = C_8 \qquad \vdash_{\mathcal{RE}} C_8 = C_9}{\vdash_{\mathcal{RE}} C_7 = C_9}\,=}{\vdash_{\mathcal{RE}} C_5 = C_9}\,=$$
$$\rho_2$$

$$\dfrac{\dfrac{\overset{\rho_2}{\vdash_{\mathcal{RE}} C_5 = C_9} \qquad \dfrac{\vdash_{\mathcal{RE}} E(c+c) \qquad E(x+y) \vdash_{\mathcal{RE}} E((x+1)+(y+1))}{\vdash_{\mathcal{RE}} E((c+1)+(c+1))}\,R}{\vdash_{\mathcal{RE}} E((c+y)+(1+1))}\,= \qquad \dfrac{E(c+(1+(1+c))) \vdash_{\mathcal{RE}} \qquad \overset{\rho_1}{\vdash_{\mathcal{RE}} C_1 = C_4}}{E((c+c)+(1+1)) \vdash_{\mathcal{RE}}}\,=}{\vdash_{\mathcal{RE}}}\,R, \sigma = \{y \leftarrow c\}$$

Figure 6.2: Resolution refutation for the example in CERES with LKDe

6.3 Implementation Details

Since GAP is written in the functional Scala programming language, using Scala was an obvious choice. Stickel's unification algorithm (see section 3.4) was chosen because most improvements are based on it. This choice proved to be very hindering, since the expected ease of adaptability is not provided by the algorithm.

Diophantine Solver

To solve the diophantine equations, Lankford's algorithm (see section 3.3) was used. Its mathematical specification is very close to an implementation by iterating over the inductive steps of the algorithm. In absence of a suited vector library, a vector implementation is also included. Since most operations on vectors are iterations, vectors are represented as LISP-style lists of integers and also matrices are lists of vectors. For the sets, the HashSet datatype wes deemed fitting.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash C_5 = C_6, \Delta \quad \Gamma \vdash C_6 = C_7, \Delta}{\Gamma \vdash C_5 = C_7, \Delta}\,= \quad \dfrac{\Gamma \vdash C_7 = C_8, \Delta \quad \Gamma \vdash C_8 = C_9, \Delta}{\Gamma \vdash C_7 = C_9, \Delta}\,=}{\Gamma \vdash C_5 = C_9, \Delta}\,= \quad \dfrac{\Gamma \vdash E(c+c), \Delta \quad \Gamma, E(c+c) \vdash E((c+1)+(c+1)), \Delta}{\Gamma \vdash E((c+1)+(c+1)), \Delta}\,cut}{\Gamma \vdash E((c+c)+(1+1)), \Delta}\,=$$

$$\pi_1$$

$$\dfrac{\Gamma, E(c+(1+(1+c))) \vdash \Delta \quad \dfrac{\dfrac{\Gamma \vdash C_1 = C_2, \Delta \quad \Gamma \vdash C_2 = C_3, \Delta}{\Gamma \vdash C_1 = C_3, \Delta}\,= \quad \Gamma \vdash C_3 = C_4, \Delta}{\Gamma \vdash C_1 = C_4, \Delta}\,=}{\Gamma, E((c+c)+(1+1)) \vdash \Delta}\,=$$

$$\pi_2$$

$$\dfrac{\overset{\pi_1}{\Gamma \vdash E((c+c)+(1+1)), \Delta} \quad \overset{\pi_1}{\Gamma, E((c+c)+(1+1)) \vdash \Delta}}{\Gamma \vdash \Delta}\,cut$$

Figure 6.3: ACNF of the example in CERES with LKDe

Unification

The separation of the elementary case from the general case in the mathematical description of Stickel's algorithm leaves some room for improvements in the implementation. In the abstraction step it is already known which variables will be unified with a constant or a term not starting with the symbol f i.e. which coefficients of a solution have to be equal to one. For this reason two functions $unifiable\_invariant : \mathbb{N}^{n+m} \mapsto \mathbb{B}$ and $unifiable\_condition : \mathbb{N}^{m+n} \mapsto \mathbb{B}$ are created on the fly. Let I be the set of indices of coefficients which will need to be equal to one because they will be unified with a constant later. Then $unifiable\_invariant(\underline{v})$ is true if and only if $\pi_i(\underline{v}) \leqslant 1$ for $i \in I$. For ACU $unifiable\_condition(\underline{v})$ is true if and only if $\pi_i(\underline{v}) = 1$ for $i \in I$ and for AC $unifiable\_condition(\underline{v})$ is true if and only if $\pi_i(\underline{v}) = 1$ for $i \in I$ and $\pi_j(\underline{v}) > 0$ with $j \in \{1, \ldots, n+m\}$. These two functions are used to remove solutions as early as possible during the generation of solutions from the basis vectors.

The unification algorithm itself is a modification of the rule based approach [MM82], its code is given in listing 1 in appendix 7. The main idea is to generalize the function unify to return a list of substitutions and to define composition of lists of substitutions as $S_1 \circ S_2 = \{\sigma_1\sigma_2 | \sigma_1 \in S_1, \sigma_2 \in S_2\}$. As it turned out, for equational theories solutions can not be composed that way. In the beginning, the missing ability to solve systems of solutions was underestimated, since we expected the composability in the following way: a set of equations $P = \{t_1 \overset{?}{=} t_2 \overset{?}{=} t_3\}$ is decomposable such that if there exists a most general unifier θ such that $s_1\theta = s_2\theta$ and there exists another most general unifier σ such that $s_2\theta\sigma = s_3\theta\sigma$, then $\theta\sigma$ is a most general unifier of P.

The example of $\{f(x,a) \overset{?}{=}_{AC} f(y,b) \overset{?}{=}_{AC} f(z,c)\}$ from section 3.4 is a counterexample to this assumption. At first sight, this might not seem too restricting since in immediate consequence, the extended resolution step in figure 4.5 would be forced to resolve only over one positive and one negative clause. This is insofar wrong since also the resolution of $\{P(u,u,u), \neg P(f(a,x), f(b,y), f(c,z))\}$ leads to a system of equations. Even the restriction to monadic predicates is not enough, since functions can play the same role as polyadic predicates. The corresponding counterexample is then $\{P(g(u,u,u)),$ $\neg P(g(f(a,x), f(b,y), f(c,z)))\}$. For this reason the implementation can only handle monadic predicates and function symbols, which is a too small subset to be used in a reasonable prover.

CHAPTER 7

Summary and Future Work

The CERES method has been successfully extended to equational theories and tried out on small examples in the theory ACU. One observation is that for CERES modulo to really make a difference, a proof has to make actual use of theory inferences. For this reason, existing proofs need to be modified which requires substantial additional programming. In the case of Fürstenberg's proof of the infinity of primes, the proof is represented in the language HLK (shorthand for Handy LK) [SHW] which for instance has automatic inference of propositional parts of a proof amongst its features. To fit the proof into the framework of deduction modulo, either the C++ implementation of HLK needs to be adapted to equational inferences or a replacement needs to be found[1]. Also the current implementation still concentrates on the refutation of the characteristic clause set and needs to be extended by the representation of sequent calculus modulo proofs for the extraction of the characteristic clause term and the projections to work.

On the theoretical level, an obvious extension is to allow propositional rewrite rules. Since a propositional rewrite rule can rewrite a formula occurrence to one containing strong quantifiers, some eigenvariable conditions may no longer hold during application of the CERES modulo method. A possible approach to this is to label formulas with the context of the strongly quantified variables in which they were before skolemization. The consequence for the resolution part is not that severe; the ENAR rule already captures these concerns because it has skolemization integrated into its clause set transformation (see figure 4.4). Since deduction modulo simple type theory also has the cut-elimination property, it could be interesting to study its relation to higher order sequent calculi, especially since the calculus $LK_{\mathcal{SK}}$ used for CERES$^\omega$ [Wel10] already has labels for formulas and also the $(\forall, r)$ and $(\exists, l)$ rules are restricted to the generalization of skolem terms only.

[1]The new implementation of CERES in Scala has similar needs. A possible replacement could be the use of a higher order proof assistant like Isabelle [Isa] or Coq [Coq].

Listings

Listing 1: Unification Algorithm

```
def unify(function: ConstantSymbolA,
            terms: List[(FOLTerm, FOLTerm)],
            substs: List[Substitution[FOLTerm]]) :
List[Substitution[FOLTerm]] = {
  terms match {
    case (term1, term2) :: rest =>
      term1 match {
      // comparing constant to sthg else
        case FOLConst(c1) =>
          term2 match {
            // if the two constants are equal, the substitution is not
            // changed, else not unifiable
            case FOLConst(c2) =>
              if (c1 == c2)
                collect(substs, ((s: Substitution[FOLTerm]) =>
                    unify(function, rest, List(s))))
              else Nil
            // second one is a var => flip & variable elimination
            case FOLVar(v) =>
              val ve = Substitution[FOLTerm](term2.asInstanceOf[FOLVar],
                  term1)
              val newterms = rest map makesubstitute_pair(ve)
              collect(substs, (s: Substitution[FOLTerm]) =>
                                unify(function, newterms, List(ve compose s)))
            // anything else is not unifiable
            case _ =>
              Nil
          }
        // comparing function symbol to sthg else
        case Function(f1, args1) =>
          term2 match {
            // decomposition or ac unification, if the function symbols are
            // the same, else not unifiable
            case Function(f2, args2) =>
              if (f1 != f2) {
                Nil
              } else if (args1.length != args2.length) {
                throw new Exception("function symbol occured with different
                    arity!")
              } else if (f1 == function) {
                //ac unification
                val acunivs = ac_unify(function, term1, term2)
                collect(acunivs, ((acu: Substitution[FOLTerm]) =>
                  collect(substs, ((subst: Substitution[FOLTerm]) =>
                    unify(function, rest map makesubstitute_pair(subst), List
                        ((acu compose subst))))
                    )))
              } else {
                //non ac unification => decomposition
                collect(substs, (s: Substitution[FOLTerm]) => unify(function,
                    (args1 zip args2) ::: rest, List(s)))
              }
            // variable as second term: flip & variable elimination
            case FOLVar(v) =>
              //occurs check
              if (occurs(term2.asInstanceOf[FOLVar], term1)) {
                Nil
              } else {
                val ve = Substitution[FOLTerm](term2.asInstanceOf[FOLVar],
                    term1)
                val newterms = rest map makesubstitute_pair(ve)
                collect(substs, (s: Substitution[FOLTerm]) => unify(function,
                    newterms, List((ve compose s))))
              }
            // anything else is not unifiable
            case _ =>
              Nil
          }
        //variable elimination
        case FOLVar(v) =>
          term2 match {
            case FOLVar(w) =>
              if (v == w) {
                collect(substs, (s: Substitution[FOLTerm]) => unify(function,
                     rest, List(s)))
              } else {
                val ve = Substitution[FOLTerm](term1.asInstanceOf[FOLVar],
                    term2)
                val newterms = rest map makesubstitute_pair(ve)
                collect(substs, (s: Substitution[FOLTerm]) => unify(function,
                     newterms, List((ve compose s).asInstanceOf[Substitution[
                    FOLTerm]])))
              }

            case _ =>
              //occurs check
              if (occurs(term1.asInstanceOf[FOLVar], term2)) {
                Nil
              } else {
                val ve = Substitution[FOLTerm](term1.asInstanceOf[FOLVar],
                    term2)
                val newterms = rest map makesubstitute_pair(ve)
                collect(substs, (s: Substitution[FOLTerm]) => unify(function,
                     newterms, List[Substitution[FOLTerm]]((ve compose s))))
              }
          }

        //this should be empty in the end
        case _ =>
          throw new Exception("there should be only variables, constants and
              functions in first order terms!")
      }

    case Nil =>
      substs
  }
}
```

Fibonacci Example

Axioms

A_1	$f(0) = 1$	A_6	$\forall x \forall y (x < x + (y + 1))$
A_2	$f(1) = 1$	A_7	$\forall x \neg x < x$
A_3	$\forall x (f((x + 1) + 1)) = f(x + 1) + f(x)$	A_8	$\forall x \forall y \forall z (x < y) \wedge (y < z) \Rightarrow (x < z)$
A_4	$\forall x \neg x < 0$	A_9	$\forall x \forall y \forall n (x + n = y + n \Rightarrow x = y)$
A_5	$\forall x \forall y (x < y \Leftrightarrow \exists z (x + z + 1 = y))$		

$IND[A] \quad A(0) \wedge (\forall A(x) \Rightarrow A(x + 1)) \Rightarrow \forall x A(x)$

Definitions

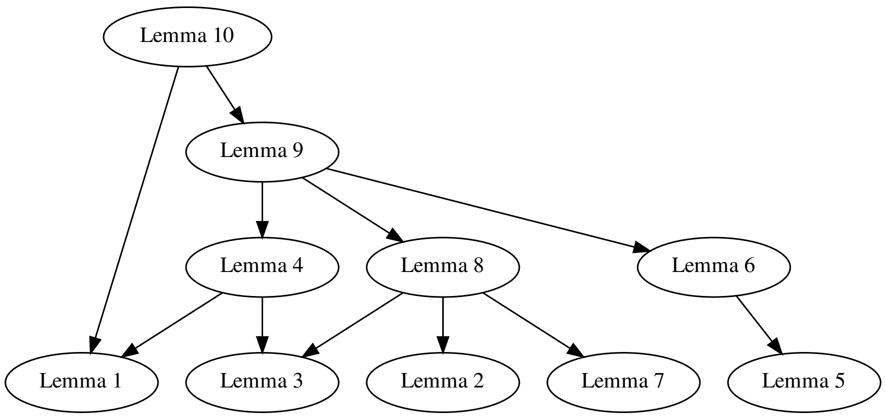
$2 \equiv 1 + 1$

$3 \equiv (1 + 1) + 1$

$4 \equiv ((1 + 1) + 1) + 1$

$AX \equiv A_1, \ldots, A_9$

Dependencies



Lemma 1: $\vdash 2 < f(3)$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{AX \vdash_{\mathcal{RE}} 2+0+1=2+1}{AX \vdash_{\mathcal{RE}} \exists z 2+z+1=2+1}\,\exists,r \qquad AX, 2<2+1 \vdash_{\mathcal{RE}} 2<2+1}{AX, \exists z 2+z+1=2+1 \Rightarrow 2<2+1 \vdash_{\mathcal{RE}} 2<2+1}}{AX, 2<2+1 \Rightarrow \exists z 2+z+1=2+1, \exists z 2+z+1=2+1 \Rightarrow 2<2+1 \vdash_{\mathcal{RE}} 2<2+1}\,w,l}{AX, 2<2+1 \Leftrightarrow \exists z 2+z+1=2+1 \vdash_{\mathcal{RE}} 2<2+1}\,\wedge,l}{AX, \forall y(2<y \Leftrightarrow \exists z 2+z+1=y) \vdash_{\mathcal{RE}} 2<2+1}\,\forall,l}{AX, A_5 \vdash_{\mathcal{RE}} 2<2+1}\,\forall,l}{AX \vdash_{\mathcal{RE}} 2<2+1}\,c,l}{(**)}$$

$$\dfrac{\dfrac{AX \vdash_{\mathcal{RE}} f(1)=1 \qquad \dfrac{AX \vdash_{\mathcal{RE}} f(1)=1 \qquad \dfrac{AX \vdash_{\mathcal{RE}} f(0)=1 \qquad \dfrac{(**)}{AX \vdash_{\mathcal{RE}} 2<2+1}}{AX \vdash_{\mathcal{RE}} 2<1+f(0)+1}\,=,r}{AX \vdash_{\mathcal{RE}} 2<1+f(0)+f(1)}}{AX \vdash_{\mathcal{RE}} 2<f(1)+f(0)+f(1)}\,=,r}{(*)}$$

$$\dfrac{\dfrac{\dfrac{AX, f(3)=f(2)+f(1) \vdash_{\mathcal{RE}} f(3)=f(2)+f(1)}{AX, AX_3 \vdash_{\mathcal{RE}} f(3)=f(2)+f(1)}\,\forall,l}{AX \vdash_{\mathcal{RE}} f(3)=f(2)+f(1)}\,c,l \qquad \dfrac{\dfrac{\dfrac{AX, f(2)=f(1)+f(0) \vdash_{\mathcal{RE}} f(3)=f(2)+f(1)}{AX, AX_3 \vdash_{\mathcal{RE}} f(2)=f(1)+f(0)}\,\forall,l}{AX \vdash_{\mathcal{RE}} f(2)=f(1)+f(0)}\,c,l \qquad (*)}{AX \vdash_{\mathcal{RE}} 2<f(2)+f(1)}\,=,r}{AX \vdash_{\mathcal{RE}} 2<f(3)}\,=,r$$

Lemma 2: $0 < 1$

$$
\begin{prooftree}
\AxiomC{$AX \vdash_{\mathcal{RE}} 0 + 1 + 0 = 1$}
\RightLabel{$\exists, r$}
\UnaryInfC{$AX \vdash_{\mathcal{RE}} \exists z 0 + 1 + z = 1$}
\AxiomC{$AX, 0 < 1 \vdash_{\mathcal{RE}} 0 < 1$}
\BinaryInfC{$AX, \exists z 0 + 1 + z = 1 \Rightarrow 0 < 1 \vdash_{\mathcal{RE}} 0 < 1$}
\RightLabel{w, l}
\UnaryInfC{$AX, 0 < 1 \Rightarrow \exists z 0 + 1 + z = 1, \exists z 0 + 1 + z = 1 \Rightarrow 0 < 1 \vdash_{\mathcal{RE}} 0 < 1$}
\RightLabel{$\wedge, l$}
\UnaryInfC{$AX, 0 < 1 \Leftrightarrow \exists z 0 + 1 + z = 1 \vdash_{\mathcal{RE}} 0 < 1$}
\RightLabel{$\forall, l$}
\UnaryInfC{$AX, \forall y (0 < y \Leftrightarrow \exists z 0 + 1 + z = y) \vdash_{\mathcal{RE}} 0 < 1$}
\RightLabel{$\forall, l$}
\UnaryInfC{$AX \vdash_{\mathcal{RE}} 0 < 1$}
\end{prooftree}
$$

Lemma 3: $\forall xyz (x < y \Rightarrow x < y + z)$

$$
\begin{prooftree}
\AxiomC{$c_1 < c_2 + c_3, c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3$}
\AxiomC{$\vdash_{\mathcal{RE}} A_6$}
\AxiomC{$c_1 < c_2 + c_3, c_1 < c_2, c_2 + c_3 < c_2 + c_3 + 0 + 1 \vdash_{\mathcal{RE}} c_2 + c_3 < c_2 + c_3 + 1$}
\RightLabel{$\forall, l$}
\UnaryInfC{$c_1 < c_2 + c_3, c_1 < c_2, \forall y (c_2 + c_3 < c_2 + c_3 + y + 1) \vdash_{\mathcal{RE}} c_2 + c_3 < c_2 + c_3 + 1$}
\RightLabel{$\forall, l$}
\UnaryInfC{$c_1 < c_2 + c_3, c_1 < c_2, A_6 \vdash_{\mathcal{RE}} c_2 + c_3 < c_2 + c_3 + 1$}
\RightLabel{cut}
\BinaryInfC{$c_1 < c_2 + c_3, c_1 < c_2 \vdash_{\mathcal{RE}} c_2 + c_3 < c_2 + c_3 + 1$}
\BinaryInfC{$c_1 < c_2 + c_3, c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3 \wedge c_2 + c_3 < c_2 + c_3 + 1$}
\AxiomC{$c_1 < c_2 + c_3 + 1 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3 + 1$}
\RightLabel{$\Rightarrow, l$}
\BinaryInfC{$(c_1 < c_2 + c_3, c_1 < c_2, c_1 < c_2 + c_3 \wedge c_2 + c_3 < c_2 + c_3 + 1) \Rightarrow c_1 < c_2 + c_3 + 1 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3 + 1$}
\UnaryInfC{(*)}
\end{prooftree}
$$

$$
\begin{prooftree}
\AxiomC{$c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2 + 0$}
\RightLabel{$\Rightarrow, r$}
\UnaryInfC{$\vdash_{\mathcal{RE}} c_1 < c_2 \Rightarrow c_1 < c_2 + 0$}
\AxiomC{$c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2$}
\AxiomC{$A_8 \vdash_{\mathcal{RE}} (c_1 < c_2 + c_3 \wedge c_2 + c_3 < c_2 + c_3) \Rightarrow c_1 < c_2 + c_3 + 1$}
\AxiomC{(*)}
\RightLabel{cut}
\BinaryInfC{$c_1 < c_2 + c_3, c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3 + 1$}
\RightLabel{$\Rightarrow, l$}
\BinaryInfC{$c_1 < c_2 \Rightarrow c_1 < c_2 + c_3, c_1 < c_2 \vdash_{\mathcal{RE}} c_1 < c_2 + c_3 + 1$}
\RightLabel{$\Rightarrow, r$}
\UnaryInfC{$c_1 < c_2 \Rightarrow c_1 < c_2 + c_3 \vdash_{\mathcal{RE}} c_1 < c_2 \Rightarrow c_1 < c_2 + c_3 + 1$}
\RightLabel{$\Rightarrow, r$}
\UnaryInfC{$\vdash_{\mathcal{RE}} (c_1 < c_2 \Rightarrow c_1 < c_2 + c_3) \Rightarrow (c_1 < c_2 \Rightarrow c_1 < c_2 + c_3 + 1)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$\vdash_{\mathcal{RE}} \forall z ((c_1 < c_2 \Rightarrow c_1 < c_2 + z) \Rightarrow (c_1 < c_2 \Rightarrow c_1 < c_2 + z + 1))$}
\RightLabel{$\vee, r$}
\BinaryInfC{$\vdash_{\mathcal{RE}} c_1 < c_2 \Rightarrow c_1 < c_2 + 0 \wedge \forall z ((c_1 < c_2 \Rightarrow c_1 < c_2 + z) \Rightarrow (c_1 < c_2 \Rightarrow c_1 < c_2 + z + 1))$}
\AxiomC{$\forall z (c_1 < c_2 \Rightarrow c_1 < c_2 + z) \vdash_{\mathcal{RE}} \forall z (c_1 < c_2 \Rightarrow c_1 < c_2 + z)$}
\BinaryInfC{$(c_1 < c_2 \Rightarrow c_1 < c_2 + 0 \wedge \forall z ((c_1 < c_2 \Rightarrow c_1 < c_2 + z) \Rightarrow (c_1 < c_2 \Rightarrow c_1 < c_2 + z + 1))) \Rightarrow \forall z (c_1 < c_2 \Rightarrow c_1 < c_2 + z) \vdash_{\mathcal{RE}} \forall z (c_1 < c_2 \Rightarrow c_1 < c_2 + z)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$IND \vdash_{\mathcal{RE}} \forall yz (c_1 < y \Rightarrow c_1 < y + z)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$IND \vdash_{\mathcal{RE}} \forall xyz (x < y \Rightarrow x < y + z)$}
\end{prooftree}
$$

Lemma 4: $\forall x 1 < f(x+3)$

$$\dfrac{\dfrac{\vdash_{\mathcal{RE}} A_8 \qquad \dfrac{\dfrac{\dfrac{\vdash_{\mathcal{RE}} 1 < f(0)+1 \qquad \dfrac{\vdash_{\mathcal{RE}} Lemma1 \qquad Lemma1 \vdash_{\mathcal{RE}} f(0)+1 < f(3)}{\vdash_{\mathcal{RE}} f(0)+1 < f(3)}}{\vdash_{\mathcal{RE}} 1 < f(0)+1 \wedge f(0)+1 < f(3)}\,\wedge, r \qquad 1 < f(3) \vdash_{\mathcal{RE}} 1 < f(3)}{1 < f(0)+1 \wedge f(0)+1 < f(3) \Rightarrow 1 < f(3) \vdash_{\mathcal{RE}} 1 < f(3)}}{A_8 \vdash_{\mathcal{RE}} 1 < f(3)}\,\forall l, 2x}{\vdash_{\mathcal{RE}} 1 < f(3)}\,cut}{(*)}$$

$$\dfrac{\dfrac{\dfrac{(*)}{\vdash_{\mathcal{RE}} 1 < f(0+3)} \qquad \dfrac{\dfrac{\dfrac{\vdash_{\mathcal{RE}} Lemma3 \qquad \dfrac{\dfrac{1 < f(x_0+3) \vdash_{\mathcal{RE}} 1 < f(x_0+3) \qquad 1 < f(x_0+3+1) \vdash_{\mathcal{RE}} 1 < f(x_0+3+1)}{1 < f(x_0+3), 1 < f(x_0+3) \Rightarrow 1 < f(x_0+3+1) \vdash_{\mathcal{RE}} 1 < f(x_0+3+1)}}{1 < f(x_0+3), Lemma3 \vdash_{\mathcal{RE}} 1 < f(x_0+3+1)}\,\forall, l \times 3}{1 < f(x_0+3) \vdash_{\mathcal{RE}} 1 < f(x_0+3+1)}\,cut}{\vdash_{\mathcal{RE}} 1 < f(x_0+3) \Rightarrow 1 < f(x_0+3+1)}\,\Rightarrow, r}{\vdash_{\mathcal{RE}} \forall x(1 < f(x+3) \Rightarrow 1 < f(x+3+1))}\,\forall, r}{\vdash_{\mathcal{RE}} 1 < f(0+3) \wedge \forall x(1 < f(x+3) \Rightarrow 1 < f(x+3+1))}\,\Rightarrow, r \qquad \forall x(1 < f(x+3)) \vdash_{\mathcal{RE}} \forall x(1 < f(x+3))}{IND[1 < f(x+3)] \vdash_{\mathcal{RE}} \forall x(1 < f(x+3))}$$

Lemma 5: $\forall x \forall y \forall n (x = y \Rightarrow x + n = y + n)$

$$\dfrac{\dfrac{\dfrac{x_0 = y_0 \vdash_{\mathcal{RE}} x_0 = y_0 \qquad \vdash_{\mathcal{RE}} x_0 + 1 = x_0 + 1}{x_0 = y_0 \vdash_{\mathcal{RE}} x_0 + n_0 = y_0 + n_0}}{\vdash_{\mathcal{RE}} x_0 = y_0 \Rightarrow x_0 + n_0 = y_0 + n_0}\,{\Rightarrow, r}}{\vdash_{\mathcal{RE}} \forall x \forall y \forall n (x = y \Rightarrow x + n = y + n)}\,{\forall, r \times 3}$$

Lemma 6: $\forall x \forall y \forall n (x < y \Rightarrow x + n < y + n)$

$$\dfrac{\dfrac{\vdash_{\mathcal{RE}} Lemma5 \qquad \dfrac{\dfrac{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} x_0 + z_0 + 1 = y_0 \qquad A_5, x_0 + z_0 + 1 + n_0 = y_0 + n_0 \vdash_{\mathcal{RE}} x_0 + n_0 + z_0 + 1 = y_0 + n_0}{A_5, x_0 + z_0 + 1 = y_0, x_0 + z_0 + 1 = y_0 \Rightarrow x_0 + z_0 + 1 + n_0 = y_0 + n_0 \vdash_{\mathcal{RE}} x_0 + n_0 + z_0 + 1 = y_0 + n_0}}{A_5, x_0 + z_0 + 1 = y_0, Lemma5 \vdash_{\mathcal{RE}} x_0 + n_0 + z_0 + 1 = y_0 + n_0}\,{\forall, l \times 3}}{\dfrac{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} x_0 + n_0 + z_0 + 1 = y_0 + n_0}{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} \exists z x_0 + n_0 + z + 1 = y_0 + n_0}\,{\exists, r}}\,{cut}}{\text{(**)}}$$

$$\dfrac{\dfrac{\text{(**)}}{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} \exists z x_0 + n_0 + z + 1 = y_0 + n_0} \qquad \dfrac{\dfrac{x_0 + n_0 < y_0 + n_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0 \qquad \exists z x_0 + n_+ z + 1 = y_0 + n_0 \vdash_{\mathcal{RE}} \exists z x_0 + n_+ z + 1 = y_0 + n_0}{\exists z x_0 + n_+ z + 1 = y_0 + n_0 \Rightarrow x_0 + n_0 < y_0 + n_0, \exists z x_0 + n_0 + z + 1 = y_0 + n_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}\,{\Rightarrow, l}}{A_5, \exists z x_0 + n_0 + z + 1 = y_0 + n_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}\,{\forall, l \times 3 / \wedge, l / w, l}}{\dfrac{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}{\text{(*)}}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\exists z (x_0 + z + 1 = y_0) \vdash_{\mathcal{RE}} \exists z (x_0 + z + 1 = y_0) \qquad x_0 < y_0 \vdash_{\mathcal{RE}} x_0 < y_0}{x_0 < y_0, x_0 < y_0 \Rightarrow \exists z (x_0 + z + 1 = y_0) \Rightarrow \vdash_{\mathcal{RE}} \exists z (x_0 + z + 1 = y_0)}}{A_5, x_0 < y_0 \vdash_{\mathcal{RE}} \exists z (x_0 + z + 1 = y_0)}\,{\forall, l \times 2 + \wedge, l + w} \qquad \dfrac{\dfrac{\text{(*)}}{A_5, x_0 + z_0 + 1 = y_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}}{A_5, \exists z (x_0 + z + 1 = y_0) \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}\,{\exists, l}}{\dfrac{A_5, x_0 < y_0 \vdash_{\mathcal{RE}} x_0 + n_0 < y_0 + n_0}{A_5 \vdash_{\mathcal{RE}} x_0 < y_0 \Rightarrow x_0 + n_0 < y_0 + n_0}\,{\Rightarrow, r}}}{A_5 \vdash_{\mathcal{RE}} \forall x \forall y \forall n (x = y \Rightarrow x + n = y + n)}\,{\forall, r \times 3}$$

Lemma 7: $(A(0) \wedge A(1)) \wedge \forall x A(x+2) \Rightarrow \forall x A(x)$ **(Schema)**

$$
\begin{prooftree}
\AxiomC{$A(0) \vdash_{\mathcal{RE}} A(0)$}
\AxiomC{$A(0), A(x_0+1), A(x_0) \vdash_{\mathcal{RE}} A(x_0+1)$}
\RightLabel{$\forall, l$}
\UnaryInfC{$A(0), \forall x A(x+1), A(x_0) \vdash_{\mathcal{RE}} A(x_0+1)$}
\RightLabel{$\Rightarrow, r$}
\BinaryInfC{$A(0), \forall x A(x+1) \vdash_{\mathcal{RE}} A(x_0) \Rightarrow A(x_0+1)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$A(0), \forall x A(x+1) \vdash_{\mathcal{RE}} \forall x (A(x) \Rightarrow A(x+1))$}
\RightLabel{$\wedge, r$}
\UnaryInfC{$A(0), \forall x A(x+1) \vdash_{\mathcal{RE}} A(0) \wedge \forall x (A(x) \Rightarrow A(x+1))$}
\AxiomC{$A(x_0) \vdash_{\mathcal{RE}} A(x_0)$}
\RightLabel{$\forall, l$}
\UnaryInfC{$\forall x A(x) \vdash_{\mathcal{RE}} A(x_0)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$\forall x A(x) \vdash_{\mathcal{RE}} \forall x A(x)$}
\RightLabel{$\Rightarrow, l$}
\BinaryInfC{$IND[A(x)], A(0), \forall x A(x+1) \vdash_{\mathcal{RE}} \forall x A(x)$}
\UnaryInfC{(*)}
\end{prooftree}
$$

$$
\begin{prooftree}
\AxiomC{$A(1) \vdash_{\mathcal{RE}} A(1)$}
\AxiomC{$A(0), A(1), A(x_1+1), A(x_1+2) \vdash_{\mathcal{RE}} A(x_1+2)$}
\RightLabel{$\forall, l$}
\UnaryInfC{$A(0), A(1), \forall x A(x+2), A(x_1+1) \vdash_{\mathcal{RE}} A(x_1+2)$}
\RightLabel{$\Rightarrow, r$}
\UnaryInfC{$A(0), A(1), \forall x A(x+2) \vdash_{\mathcal{RE}} A(x_1+1) \Rightarrow A(x_1+2)$}
\RightLabel{$\forall, r$}
\UnaryInfC{$A(0), A(1), \forall x A(x+2) \vdash_{\mathcal{RE}} \forall x (A(x+1) \Rightarrow A(x+2))$}
\BinaryInfC{$A(0), A(1), \forall x A(x+2) \vdash_{\mathcal{RE}} A(1) \wedge \forall x (A(x+1) \Rightarrow A(x+2))$}
\AxiomC{(*)}
\UnaryInfC{$IND[A(x)], A(0), A(1), \forall x A(x+2), \forall x A(x+1) \vdash_{\mathcal{RE}} \forall x A(x)$}
\RightLabel{$\Rightarrow, l(IND)$}
\BinaryInfC{$IND[A(x)], IND[A(x+1)], A(0), A(1), \forall x A(x+2) \vdash_{\mathcal{RE}} \forall x A(x)$}
\RightLabel{$\wedge, l$}
\UnaryInfC{$IND[A(x)], IND[A(x+1)], A(0) \wedge A(1), \forall x A(x+2) \vdash_{\mathcal{RE}} \forall x A(x)$}
\RightLabel{$\wedge, l$}
\UnaryInfC{$IND[A(x)], IND[A(x+1)], (A(0) \wedge A(1)) \wedge \forall x A(x+2) \vdash_{\mathcal{RE}} \forall x A(x)$}
\end{prooftree}
$$

Lemma 8: $\forall x(0 < f(x))$

$\Gamma \equiv AX, IND[0 < f(x)], IND[0 < f(x+1)], IND[0 < f(x+2)], 0 < 1$

$$\dfrac{\dfrac{AX, 0 < f(x_2+2) \vdash_{\mathcal{RE}} 0 < f(x_2+2) \qquad AX, 0 < f(x_2+2), 0 < f(x_2+2) + f(x_2+1) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}{AX, 0 < f(x_2+2), 0 < f(x_2+2) \Rightarrow 0 < f(x_2+2) + f(x_2+1) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}\;\Rightarrow, l}{\text{(V)}}$$

$$\dfrac{\dfrac{AX, IND[....] \vdash_{\mathcal{RE}} Lemma3 \qquad \dfrac{\dfrac{\dfrac{\text{(V)}}{AX, 0 < f(x_2+2), \forall z(0 < f(x_2+2) \Rightarrow 0 < f(x_2+2) + z) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}\;\forall, l}{AX, 0 < f(x_2+2), \forall y \forall z(0 < y \Rightarrow 0 < y + z) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}\;\forall, l}{AX, 0 < f(x_2+2), \forall x \forall y \forall z(x < y \Rightarrow x < y + z) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}\;\forall, l}{AX, 0 < f(x_2+2) \vdash_{\mathcal{RE}} 0 < f(x_2+2) + f(x_2+1)}\;cut}{\text{(IV)}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{AX, f(2) = f(1) + f(0) \vdash_{\mathcal{RE}} f(2) = f(1) + f(0)}{AX, AX_3 \vdash_{\mathcal{RE}} f(2) = f(1) + f(0)}\;\forall, l}{AX \vdash_{\mathcal{RE}} f(2) = f(1) + f(0)}\;c, l \qquad \dfrac{AX \vdash_{\mathcal{RE}} f(1) = 1 \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{AX \vdash_{\mathcal{RE}} 0 + f(0) + 1 = 1 + f(0)}{AX \vdash_{\mathcal{RE}} \exists z(0 + z + 1 = 1 + f(0))}\;\exists, r \qquad AX, 0 < 1 + f(0) \vdash_{\mathcal{RE}} 0 < 1 + f(0)}{AX, \exists z(0 + z + 1 = 1 + f(0)) \Rightarrow 0 < 1 + f(0) \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;\Rightarrow, l}{AX, 0 < 1 + f(0) \Rightarrow \exists z(0 + z + 1 = 1 + f(0)), \exists z(0 + z + 1 = 1 + f(0)) \Rightarrow 0 < 1 + f(0) \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;w, l}{AX, 0 < 1 + f(0) \Leftrightarrow \exists z(0 + z + 1 = 1 + f(0)) \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;\wedge, l}{AX, \forall y(0 < y \Leftrightarrow \exists z(0 + z + 1 = y)) \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;\forall, l}{AX, AX_5 \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;\forall, l}{AX \vdash_{\mathcal{RE}} 0 < 1 + f(0)}\;c, l}{AX \vdash_{\mathcal{RE}} 0 < f(1) + f(0)}\;=, r}{AX \vdash_{\mathcal{RE}} 0 < f(0+2)}\;=, r}{\text{(III)}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{(III)}}{AX \vdash_{\mathcal{RE}} 0 < f(0+2)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{AX, f(x_2+3) = f(x_2+2) + f(x_2+1) \vdash_{\mathcal{RE}} f(x_2+3) = f(x_2+2) + f(x_2+1)}{AX, AX_3 \vdash_{\mathcal{RE}} f(x_2+3) = f(x_2+2) + f(x_2+1)}\;\forall, l}{AX \vdash_{\mathcal{RE}} f(x_2+3) = f(x_2+2) + f(x_2+1)}\;c, l \qquad \text{(IV)}}{AX, 0 < f(x_2+2) \vdash_{\mathcal{RE}} 0 < f(x_2+3)}\;=, r}{AX \vdash_{\mathcal{RE}} 0 < f(x_2+2) \Rightarrow 0 < f(x_2+3)}\;\Rightarrow, r}{AX \vdash_{\mathcal{RE}} \forall x(0 < f(x+2) \Rightarrow 0 < f(x+3))}\;\forall, r}{AX \vdash_{\mathcal{RE}} 0 < f(0+2) \wedge \forall x(0 < f(x+2) \Rightarrow 0 < f(x+3))}\;\wedge, r \qquad \dfrac{\dfrac{0 < f(x_1+2) \vdash_{\mathcal{RE}} 0 < f(x_1+2)}{\forall x(0 < f(x+2)) \vdash_{\mathcal{RE}} 0 < f(x_1+2)}\;\forall, l}{\forall x(0 < f(x+2)) \vdash_{\mathcal{RE}} \forall x(0 < f(x+2))}\;\forall, r}{AX, IND[0 < f(x+2)] \vdash_{\mathcal{RE}} \forall x(0 < f(x+2))}}{\text{(II)}}$$

(Lemma 8 ctd.)

$$\dfrac{\dfrac{\text{(II)}}{AX, IND[0 < f(x+2)] \vdash_{\mathcal{RE}} \forall x(0 < f(x+2))} \qquad \dfrac{\dfrac{\text{(Lemma 7 } [0 < f(x)])}{IND[0 < f(x)], IND[0 < f(x+1)], (0 < f(0) \land 0 < f(1)) \land \forall x(0 < f(x+2)) \vdash_{\mathcal{RE}} \forall x(0 < f(x))} \qquad \dfrac{\dfrac{0 < f(x_0) \vdash_{\mathcal{RE}} 0 < f(x_0)}{\forall x(0 < f(x)) \vdash_{\mathcal{RE}} 0 < f(x_0)}\ \forall, l}{\forall x(0 < f(x)) \vdash_{\mathcal{RE}} \forall x(0 < f(x))}\ \forall, r}{\Gamma, 0 < f(0), 0 < f(1), \forall x(0 < f(x+2)) \vdash_{\mathcal{RE}} \forall x(0 < f(x))}\ cut}{\dfrac{\Gamma, 0 < f(0), 0 < f(1) \vdash_{\mathcal{RE}} \forall x(0 < f(x))}{\text{(I)}}}\ cut$$

$$\dfrac{AX \vdash_{\mathcal{RE}} Lemma2 \qquad \dfrac{\dfrac{AX \vdash_{\mathcal{RE}} f(0) = 1 \qquad AX, 0 < 1 \vdash_{\mathcal{RE}} 0 < 1}{AX, 0 < 1 \vdash_{\mathcal{RE}} 0 < f(0)}\ =, r \qquad \dfrac{\dfrac{AX \vdash_{\mathcal{RE}} f(1) = 1 \qquad AX, 0 < 1 \vdash_{\mathcal{RE}} 0 < 1}{AX, 0 < 1 \vdash_{\mathcal{RE}} 0 < f(1)}\ =, r \qquad \dfrac{\text{(I)}}{\Gamma, 0 < f(0), 0 < f(1) \vdash_{\mathcal{RE}} \forall x(0 < f(x))}}{\Gamma, 0 < f(0) \vdash_{\mathcal{RE}} \forall x(0 < f(x))}\ cut}{\Gamma \vdash_{\mathcal{RE}} \forall x(0 < f(x))}\ cut}{AX, IND[0 < f(x)], IND[0 < f(x+1)], IND[0 < f(x+2)] \vdash_{\mathcal{RE}} \forall x(0 < f(x))}\ cut$$

Lemma 9: $\forall x f(x)+1<f(x+3) \Rightarrow f(x+1)+1<f((x+3)+1)$

$\Gamma_2 \equiv AX, IND[0<f(x)], IND[0<f(x+1)], IND[0<f(x+2)]$

$$\dfrac{\dfrac{AX \vdash_{\mathcal{RE}} Lemma8 \qquad \dfrac{AX, 0<f(x_0) \vdash_{\mathcal{RE}} 0<f(x_0)}{AX, Lemma8 \vdash_{\mathcal{RE}} 0<f(x_0)}\;\forall,l}{AX \vdash_{\mathcal{RE}} 0<f(x_0)} \qquad AX, 0+f(x_0+3)<f(x_0)+f(x_0+3) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}{\dfrac{AX, 0<f(x_0) \Rightarrow 0+f(x_0+3)<f(x_0)+f(x_0+3) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}{(****)}}$$

$$\dfrac{AX,1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3) \qquad \dfrac{\dfrac{AX \vdash_{\mathcal{RE}} Lemma6 \qquad \dfrac{\dfrac{\dfrac{\dfrac{(****)}{AX, 0<f(x_0) \Rightarrow 0+f(x_0+3)<f(x_0)+f(x_0+3) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}}{AX, \forall z(0<f(x_0) \Rightarrow 0+z<f(x_0)+z) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}\;\forall,l}{AX, \forall y \forall z(0<y \Rightarrow 0+z<y+z) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}\;\forall,l}{AX, Lemma6 \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}\;\forall,l}{AX \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}\;cut}{AX,1<f(x_0+3) \vdash_{\mathcal{RE}} f(x_0+3)<f(x_0+3)+f(x_0)}\;w,l}{\dfrac{AX,1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3) \wedge f(x_0+3)<f(x_0+3)+f(x_0)}{(***)}}\;\wedge,r$$

$$\dfrac{\dfrac{\dfrac{(***)}{AX,1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3) \wedge f(x_0+3)<f(x_0+3)+f(x_0)} \qquad AX,1<f(x_0+3)+f(x_0) \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}{AX,1<f(x_0+3) \wedge f(x_0+3)<f(x_0+3)+f(x_0) \Rightarrow 1<f(x_0+3)+f(x_0), 1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}\;\Rightarrow,l}{\dfrac{AX, \forall z(1<f(x_0+3) \wedge f(x_0+3)<z \Rightarrow 1<z), 1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}{(**)}}\;\forall,l$$

$$\dfrac{AX, IND[1<f(x+3)] \vdash_{\mathcal{RE}} Lemma4 \qquad \dfrac{\dfrac{\dfrac{\dfrac{(**)}{AX, \forall z(1<f(x_0+3) \wedge f(x_0+3)<z \Rightarrow 1<z), 1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}}{AX, \forall y \forall z(1<y \wedge y<z \Rightarrow 1<z), 1<f(x_0+3) \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}\;\forall,l}{AX, A_8, Lemma4 \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}\;\forall,l}{AX, Lemma4 \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}\;c,l}{\dfrac{AX \vdash_{\mathcal{RE}} 1<f(x_0+3)+f(x_0)}{(**)}}\;cut$$

$$\dfrac{\dfrac{AX \vdash_{\mathcal{RE}} Lemma6 \qquad \dfrac{\dfrac{\dfrac{\dfrac{(**) \qquad AX, 1<f(x_0+3)+f(x_0) \Rightarrow 1+f(x_0+1)<f(x_0+3)+f(x_0)+f(x_0+1) \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}{AX, 1<f(x_0+3)+f(x_0) \Rightarrow 1+f(x_0+1)<f(x_0+3)+f(x_0)+f(x_0+1) \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}\;\Rightarrow,l}{AX, \forall z(1<f(x_0+3)+f(x_0) \Rightarrow 1+z<f(x_0+3)+f(x_0)+z) \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}\;\forall,l}{AX, \forall y \forall z(1<y \Rightarrow 1+z<y+z \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}\;\forall,l}{AX, Lemma6 \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}\;\forall,l}{AX \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}}{\dfrac{AX, f(x_0)+1<f(x_0+3) \vdash_{\mathcal{RE}} f(x_0+1)+1<f(x_0+3)+f(x_0+1)+f(x_0)}{(*)}}\;w,l$$

Lemma 10: $\forall x f(x) + 1 < f(x+3)$

Remark: this must also work without induction, since lemma 9 does not use the induction hypothesis.

$$\dfrac{\dfrac{\dfrac{\vdash_{\mathcal{RE}} Lemma1 \qquad Lemma1 \vdash_{\mathcal{RE}} f(0)+1 < f(3)}{\vdash_{\mathcal{RE}} f(0)+1 < f(3)} \qquad \dfrac{\vdash_{\mathcal{RE}} Lemma9 \qquad Lemma9 \vdash_{\mathcal{RE}} \forall x f(x)+1 < f(x+3) \Rightarrow f(x+1)+1 < f(x+4)}{\vdash_{\mathcal{RE}} \forall x f(x)+1 < f(x+3) \Rightarrow f(x+1)+1 < f(x+4)}}{\vdash_{\mathcal{RE}} f(0)+1 < f(3) \land \forall x f(x)+1 < f(x+3) \Rightarrow f(x+1)+1 < f(x+4)}\land, r \qquad \forall x f(x)+1 < f(x+3) \vdash_{\mathcal{RE}} \forall x f(x)+1 < f(x+3)}{IND[f(x)+1 < f(x+3)] \vdash_{\mathcal{RE}} \forall x f(x)+1 < f(x+3)}\Rightarrow, l$$

Notation

When talking about sets of variables, we sometimes group them by writing them in vector notation (especially if their number is not important). The term $f(x_1, \ldots, x_n)$ then just becomes $f(\overline{x})$. Vectors over the naturals are then just underlined.

$\overline{x}$	vector notation for logical symbols
$\underline{v}$	vector over natural numbers
$a, \ldots, d$	logical constants
e	logical constant for the unit element
f, g	logical function symbols
$i, \ldots, r$	index variables over the natural numbers
s, t	terms of universal algebra and logic
$u, \ldots, z$	logical variables and variables over the naturals
P, Q, R	predicate symbols
$A, \ldots, G$	logical formulas
$\Gamma, \Delta, \Pi, \Lambda$	sets of logical formulas
σ, τ, θ	substitutions
π, ρ	proofs
S	sequent (occurrence)
X	rule name
ξ, β	special vectors (solutions and bases)

List of Tables

List of Figures

Bibliography

[AK92] Mohamed Adi and Claude Kirchner, *Ac-unification race: the system solving approach, implementation and benchmarks*, J. Symb. Comput. **14** (1992), 51–70.

[BCD90] Alexandre Boudet, Evelyne Contejean, and Hervé Devie, *A new ac unification algorithm with an algorithm for solving systems of diophantine equations*, LICS, IEEE Computer Society, 1990, pp. 289–299.

[BEL01] Matthias Baaz, Uwe Egly, and Alexander Leitsch, *Normal form transformations*, in Robinson and Voronkov [RV01], pp. 273–333.

[BHL$^+$06] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter, and Hendrik Spohr, *Proof transformation by ceres*, MKM (Jonathan M. Borwein and William M. Farmer, eds.), Lecture Notes in Computer Science, vol. 4108, Springer, 2006, pp. 82–93.

[BHL$^+$08] ______, *Ceres: An analysis of fürstenberg's proof of the infinity of primes*, Theor. Comput. Sci. **403** (2008), no. 2-3, 160–175.

[Bir35] Garrett Birkhoff, *On the structure of abstract algebras*, Mathematical Proceedings of the Cambridge Philosophical Society, vol. 31, 1935, pp. 433–454.

[BL99] Matthias Baaz and Alexander Leitsch, *Cut normal forms and proof complexity*, Ann. Pure Appl. Logic **97** (1999), no. 1-3, 127–177.

[BL00] ______, *Cut-eleminination and redundancy-elimination by resolution*, J. Symbolic Computation **29** (2000), no. 2.

[BL06] ______, *Towards a clausal analysis of cut-elimination*, J. Symb. Comput. **41** (2006), no. 3-4, 381–410.

[BN98] Franz Baader and Tobias Nipkow, *Term rewriting and all that*, Cambridge University Press, New York, NY, USA, 1998.

[BS81] S. Burris and H. P. Sankappanavar, *A Course in Universal Algebra*, Graduate Texts in Mathematics, vol. 78, Springer-Verlag, New York, 1981.

[BS01] Franz Baader and Wayne Snyder, *Unification theory*, in Robinson and Voronkov [RV01], pp. 445–532.

[Bus98] Samuel R. Buss, *An introduction to proof theory*, Studies in Logic and the Foundations of Mathematics, vol. 137, pp. 36–58, Elsevier, Amsterdam, 1998.

[CF89] Michael Clausen and Albrecht Fortenbacher, *Efficient solution of linear diophantine equations*, J. Symb. Comput. **8** (1989), 201–216.

[Con93] Evelyne Contejean, *Solving linear diophantine constraints incrementally*, Proceedings of the tenth international conference on logic programming on Logic programming (Cambridge, MA, USA), ICLP'93, MIT Press, 1993, pp. 532–549.

[Coq] *Coq website*, http://coq.inria.fr.

[Dav53] Martin Davis, *Arithmetical problems and recursively enumerable predicates*, Journal of Symbolic Logic **18** (1953), no. 1, 33–41.

[DHK03] Gilles Dowek, Thérèse Hardin, and Claude Kirchner, *Theorem proving modulo*, J. Autom. Reasoning **31** (2003), no. 1, 33–72.

[Dio52] Diophantus, *Arithmetik des Diophantos aus Alexandria*, Vandenhoeck & Ruprecht, 1952, Ed. Arthur Czwalina.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud, *Rewrite systems*, 243–320.

[Dom91a] Eric Domenjoud, *Outils pour la déduction automatique dans les théories associatives-commutatives, Thèse de doctorat*, Ph.D. thesis, Université de Nancy I, France, 1991.

[Dom91b] ———, *Solving systems of linear diophantine equations: An algebraic approach*, MFCS, 1991, pp. 141–150.

[Dom92] Eric Domenjoud, *A technical note on ac-unification. the number of minimal unifiers of the equation $\alpha x_1 + \cdots + \alpha x_p \doteq_{AC} \beta y_1 + \cdots + \beta y_q$*, Journal of Automated Reasoning **8** (1992), 39–44, Also available as Technical Report 89-R-2, CRIN, Nancy (France).

[Dow01] Gilles Dowek, *Higher-order unification and matching*, in Robinson and Voronkov [RV01], pp. 1009–1062.

[DPR61] Martin Davis, Hilary Putnam, and Julia Robinson, *The decision problem for exponential diophantine equations*, The Annals of Mathematics **74** (1961), no. 3, 425–436.

[DW03] Gilles Dowek and Benjamin Werner, *Proof normalization modulo*, J. Symb. Log. **68** (2003), no. 4, 1289–1316.

[Eke93] Steven M. Eker, *Improving the efficiency of ac matching and unification*, Tech. report, C.R.I.N. INRIA Lorraine, 1993.

[Ekm94] Jan Ekman, *Normal proofs in set theory*, Ph.D. thesis, Chalmers University and University of Göteborg, 1994.

[Fag84] François Fages, *Associative-commutative unification*, CADE (Robert E. Shostak, ed.), Lecture Notes in Computer Science, vol. 170, Springer, 1984, pp. 194–208.

[For87] Albrecht Fortenbacher, *An algebraic approach to unification under associativity and commutativity*, J. Symb. Comput. **3** (1987), no. 3, 217–229.

[Gen35] Gerhard Gentzen, *Untersuchungen über das logische Schließen. I, II*, Mathematische Zeitschrift **39** (1935), no. 1, 176–210, 405–431.

[God90] Kurt Godden, *Lazy unification*, Proceedings of the 28th annual meeting on Association for Computational Linguistics (Stroudsburg, PA, USA), ACL '90, Association for Computational Linguistics, 1990, pp. 180–187.

[GS93] David Gries and Fred B. Schneider, *A logical approach to discrete math*, Springer-Verlag New York, Inc., New York, NY, USA, 1993.

[Hal83] Lars Hallnäs, *On normalization of proofs in set theory*, Ph.D. thesis, University of Stockholm, 1983.

[Her10] Olivier Hermant, *Resolution is cut-free*, J. Autom. Reasoning **44** (2010), no. 3, 245–276.

[Hil00] David Hilbert, *Mathematische Probleme*, Göttinger Nachrichten (1900), 253–297, an english translation can be found as *Mathematical Problems* in Bulletin of the American Mathematical Society 8 (1902) or online under http://aleph0.clarku.edu/ djoyce/hilbert/problems.html.

[HLWP08] Stefan Hetzl, Alexander Leitsch, Daniel Weller, and Bruno Woltzenlogel Paleo, *Herbrand sequent extraction*, AISC/MKM/Calculemus (Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, eds.), Lecture Notes in Computer Science, vol. 5144, Springer, 2008, pp. 462–477.

[Isa] *Isabelle website*, http://www.cl.cam.ac.uk/research/hvg/Isabelle.

[KN92] Deepak Kapur and Paliath Narendran, *Complexity of unification problems with associative-commutative operators*, J. Autom. Reason. **9** (1992), 261–288.

[Kni89] Kevin Knight, *Unification: a multidisciplinary survey*, ACM Comput. Surv. **21** (1989), 93–124.

[Lan89] Dallas Lankford, *Non-negative integer basis algorithms for linear equations with integer coefficients*, J. Autom. Reasoning **5** (1989), no. 1, 25–35.

[LB11] Alexander Leitsch and Matthias Baaz, *Methods of cut-elmination*, Springer-Verlag New York, Inc., New York, NY, USA, 2011.

[LC89] Patrick Lincoln and Jim Christian, *Adventures in associative-commutative unification*, J. Symb. Comput. **8** (1989), no. 1-2, 217–240.

[Lei94] Daniel Leivant, *A foundational delineation of poly-time*, Inf. Comput. **110** (1994), 391–420.

[LF05] Alexander Leitsch and Christian Fermüller, *The resolution principle*, Handbook of Philosophical Logic, second edition (Dov Gabbay and Franz Guenthner, eds.), vol. 12, Springer, Dordrecht, 2005, pp. 175–278.

[LS76] M. Livesay and Jörg Siekmann, *Unification of A+C-terms (bags) and A+C+I-terms (sets)*, Intern. Ber. 5/76, Universität Karlsruhe, 1976.

[LWWP$^+$] Tomer Libal, Daniel Weller, Bruno Woltzenlogel-Paleo, Tsvetan Dunchev, Mikheil Rukhaia, and Martin Riener, *Generic Architecture for Proofs*, http://code.google.com/p/gapt.

[Mat70] Yuri Matiyasevich, *Enumerable sets are diophantine*, Doklady Akademii Nauk SSSR **191** (1970), no. 2, 279–282, English translation: Soviet Mathematics. Doklady, 11(2):354–358.

[MM82] Alberto Martelli and Ugo Montanari, *An efficient unification algorithm*, ACM Trans. Program. Lang. Syst. **4** (1982), 258–282.

[Pal09] Bruno Woltzenlogel Paleo, *A general analysis of cut-elimination by ceres*, Ph.D. thesis, University of Technology Vienna, 2009.

[Pla93] David A. Plaisted, *Equational reasoning and term rewriting systems*, pp. 274–364, Oxford University Press, Inc., New York, NY, USA, 1993.

[Plo72] Gordon D. Plotkin, *Building-in equational theories*, vol. 7, Edinburgh University Press, 1972, pp. 73–90.

[Pot91] Loic Pottier, *Minimal solutions of linear diophantine systems: Bounds and algorithms*, RTA (Ronald V. Book, ed.), Lecture Notes in Computer Science, vol. 488, Springer, 1991, pp. 162–173.

[Rob52] Julia Robinson, *Existential definability in arithmetic*, Transactions of the American Mathematical Society **72** (1952), no. 3, 437–449, http://www.jstor.org/stable/1990711.

[Rob65] J. A. Robinson, *A machine-oriented logic based on the resolution principle*, J. ACM **12** (1965), 23–41.

[Rub99] Albert Rubio, *A fully syntactic ac-rpo*, RTA (Paliath Narendran and Michaël Rusinowitch, eds.), Lecture Notes in Computer Science, vol. 1631, Springer, 1999, pp. 133–147.

[Rus96] Bertrand Russell, *The Principles of Mathematics*, W.W. Norton & Co., February 1996.

[Rus10] Vlad Rusu, *Combining theorem proving and narrowing for rewriting-logic specifications*, TAP (Gordon Fraser and Angelo Gargantini, eds.), Lecture Notes in Computer Science, vol. 6143, Springer, 2010, pp. 135–150.

[RV01] John Alan Robinson and Andrei Voronkov (eds.), *Handbook of automated reasoning (in 2 volumes)*, Elsevier and MIT Press, 2001.

[SHW] Hendrik Spohr, Stefan Hetzl, and Daniel Weller, *HLK website*, http://logic.at/hlk.

[Sti75] Mark E. Stickel, *A complete unification algorithm for associative-commutative functions*, IJCAI'75: Proceedings of the 4th international joint conference on Artificial intelligence (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1975, pp. 71–76.

[Sti81] ______, *A unification algorithm for associative-commutative functions*, J. ACM **28** (1981), no. 3, 423–434.

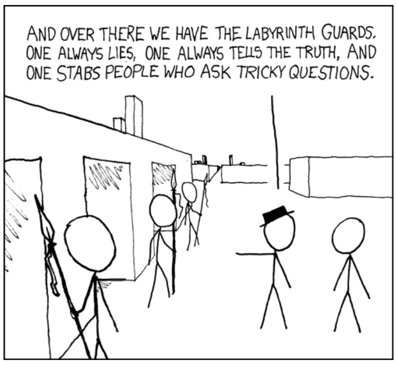
[Tid86] E Tidén, *Unification in combinations of collapse-free theories with disjoint sets of function symbols*, Proc. of the 8th international conference on Automated deduction (New York, NY, USA), Springer-Verlag New York, Inc., 1986, pp. 431–449.

[TS00] A. S. Troelstra and H. Schwichtenberg, *Basic proof theory (2nd ed.)*, Cambridge University Press, New York, NY, USA, 2000.

[Wei] Eric W. Weisstein, *Fermat's last theorem*, From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/FermatsLastTheorem.html.

[Wel10] Daniel Weller, *CERES in higher-order logic*, Ph.D. thesis, Vienna University of Technology, 2010.

[Yel85] Kathy Yelick, *Combining unification algorithms for confined regular equational theories*, Proc. of the first international conference on Rewriting techniques and applications (New York, NY, USA), Springer-Verlag New York, Inc., 1985, pp. 365–380.


AND OVER THERE WE HAVE THE LABYRINTH GUARDS. ONE ALWAYS LIES, ONE ALWAYS TELLS THE TRUTH, AND ONE STABS PEOPLE WHO ASK TRICKY QUESTIONS.

"And the whole setup is just a trap to capture escaping logicians.
None of the doors actually lead out."

by Randall Munroe, with courtesy taken from http://xkcd.com/246